Library Manager User Guide

Version S-2021.06-SP4, December 2021

Copyright and Proprietary Information Notice

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Trademarks

Free and Open-Source Licensing Notices

If applicable, Free and Open-Source Software (FOSS) licensing notices are available in the product installation.

Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

www.synopsys.com

Contents

About This User Guide

The Library Manager tool merges input from a variety of sources, including Liberty logic libraries and physical data in LEF, GDSII, or OASIS® format, to create a single, unified cell library in NDM format that contains all of the physical and logical information for a technology and its library cells.

This user guide is for engineers who use the Library Manager tool to prepare NDM cell libraries.

To use the Library Manager tool, you need to be familiar with the following:

- Logical and physical design principles
- The Linux operating system
- The tool command language (Tcl)

This preface includes the following sections:

- New in This Release
- Related Products, Publications, and Trademarks
- Conventions
- Customer Support

New in This Release

Information about new features, enhancements, and changes, known limitations, and resolved Synopsys Technical Action Requests (STARs) is available in the Library Manager Release Notes on the SolvNetPlus site.

Related Products, Publications, and Trademarks

For additional information about the TestMAX XLBIST tool, see the documentation on the Synopsys SolvNetPlus support site at the following address:

https://solvnetplus.synopsys.com

You might also want to see the documentation for the following related Synopsys products:

- Library Compiler™

Release Notes

Information about new features, enhancements, changes, known limitations, and resolved Synopsys Technical Action Requests (STARs) is available in the *IC Compiler II Release Notes* on the SolvNet site.

To see the *IC Compiler II Release Notes*,

1. Go to the SolvNet Download Center located at the following address:

   https://solvnetplus.synopsys.com

2. Select IC Compiler II, and then select a release in the list that appears.

Conventions

The following conventions are used in Synopsys documentation.

Convention	Description
`Courier`	Indicates syntax, such as `write_file`.
`Courier italic`	Indicates a user-defined value in syntax, such as `write_file` *`design_list`*
`Courier bold`	Indicates user input—text you type verbatim—in examples, such as `prompt>` **`write_file top`**
`Purple`	• Within an example, indicates information of special interest. • Within a command-syntax section, indicates a default, such as `include_enclosing =` **`true`** `| false`
[ ]	Denotes optional arguments in syntax, such as `write_file [-format` *`fmt`*`]`
...	Indicates that arguments can be repeated as many times as needed, such as *`pin1 pin2 ... pinN`*.
|	Indicates a choice among alternatives, such as `low | medium | high`
\	Indicates a continuation of a command line.
/	Indicates levels of directory structure.
Bold	Indicates a graphical user interface (GUI) element that has an action associated with it.

Convention	Description
Edit > Copy	Indicates a path to a menu command, such as opening the **Edit** menu and choosing **Copy**.
Ctrl+C	Indicates a keyboard combination, such as holding down the Ctrl key and pressing C.

Customer Support

Customer support is available through SolvNetPlus.

Accessing SolvNetPlus

The SolvNetPlus site includes a knowledge base of technical articles and answers to frequently asked questions about Synopsys tools. The SolvNetPlus site also gives you access to a wide range of Synopsys online services including software downloads, documentation, and technical support.

To access the SolvNetPlus site, go to the following address:

https://solvnetplus.synopsys.com

If prompted, enter your user name and password. If you do not have a Synopsys user name and password, follow the instructions to sign up for an account.

If you need help using the SolvNetPlus site, click REGISTRATION HELP in the top-right menu bar.

Contacting Customer Support

To contact Customer Support, go to https://solvnetplus.synopsys.com.

1

Working With the Library Manager Tool

Use the Library Manager tool to build an NDM cell library.

To learn how to use the library manager, see the following topics:

- Library Preparation Terminology
- Library Preparation Flow
- User Interfaces
- Entering lm_shell Commands
- Using Application Options
- Using Variables
- Viewing Man Pages
- Using Tcl Scripts
- Setup Files
- Using the Command Log File

Library Preparation Terminology

The following terms are used to describe the library preparation process for the IC Compiler II tool:

- Physical Library
- Logic Library
- Cell Library
- Physical Rules File
- Technology Library
- Aggregate Library
- Library Workspace

- Pane
- Breakpoints
- Scaling Group
- Antenna Properties

Physical Library

A *physical library* is a library that contains only physical information (frame views, and optionally design or layout views) for the library cells. Many vendors provide the physical information about the library cells for a technology in a physical library. In other cases, they provide GDSII, OASIS, or LEF files that contain the physical information for the library cells, and you must use these files to generate the physical library. This type of library is used as an input to generate a cell library.Library Preparation Terminology

See Also

- Cell Library
- Physical Rules File

Logic Library

A *logic library* is a library that contains only logical information, such as timing, power, noise, and functionality, for the library cells. This type of library is used as an input to generate a cell library.

See Also

- Cell Library

Cell Library

A *cell library* is a unified library that contains the logical and physical information for a specific technology and one or more of its library cells. Each library cell is represented as a single object with multiple views that contain various types of information about the cell. Figure 1 shows a conceptual view of a cell library. The views shown in green are the views used by the implementation tools. Table 1 describes each of the view types.

Figure 1 Conceptual View of a Cell Library

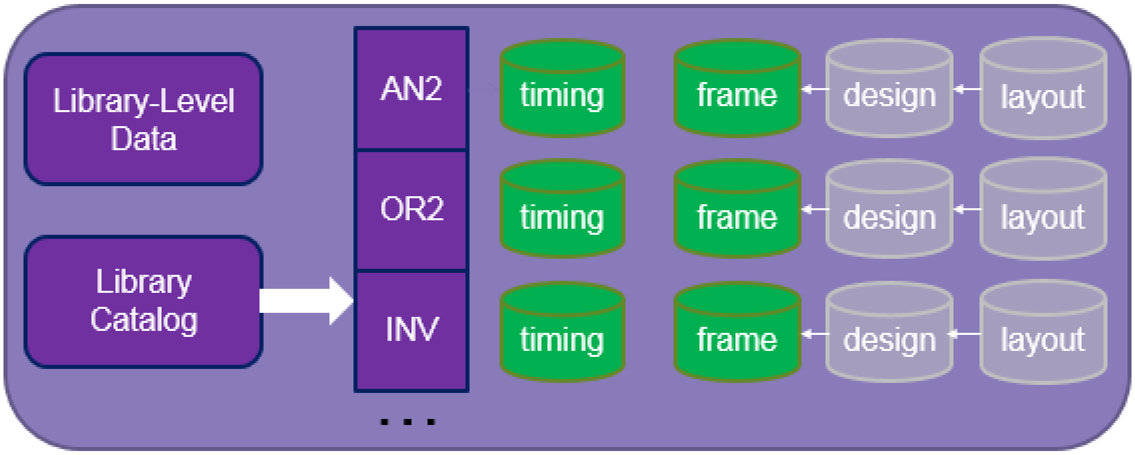
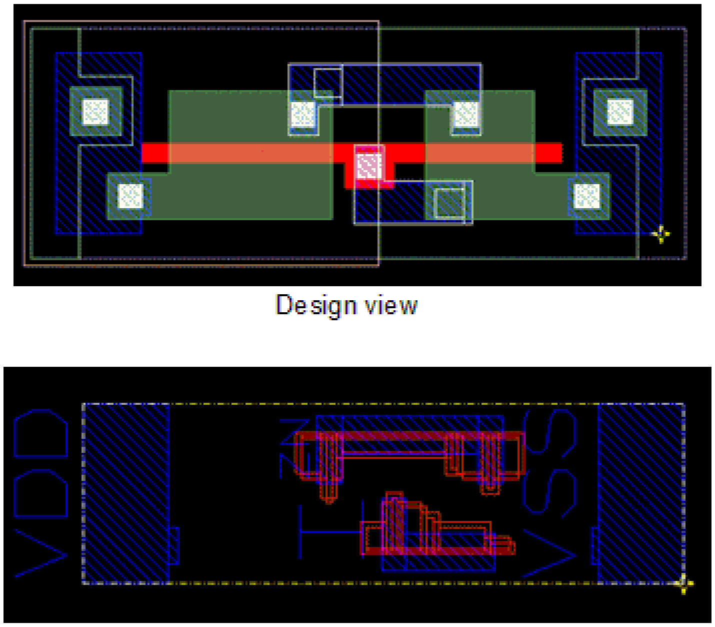


Table 1 Description of Library Cell Views

View name	Description	Usage
Timing view	Contains the timing, power, and functional information for a cell. It should contain timing and power information for all necessary characterization points (voltage and temperature) for a given process setting. The data for each characterization point is stored in a *pane*.	The implementation tools use the timing view to perform static timing analysis, power analysis, and optimization.
Layout view	Contains the physical shape information for a cell, not including connectivity and pins. This view is created only when you read GDSII or Open Artwork System Interchange Standard (OASIS) source libraries.	The library manager uses the layout view to create the design view. The implementation tools use the layout view for mask generation.
Design view	Contains all of the physical shape information for a cell, including connectivity and pins.	The library manager uses the design view to create the frame view.
Frame view	An abstraction of the physical layout of a cell that contains only the information needed for placement and routing, such as the block boundary, pins, via regions, and routing blockages.	The implementation tools use the frame view to perform placement and routing.

Figure 2 shows the difference between a design view and a frame view of a standard cell.

Figure 2 Design and Frame Views of a Standard Cell



Design view

Frame view

You create a cell library by loading information from the following sources:

- The technology file

  The technology file contains information about the routing layers and routing rules for a specific technology. For information about the technology file syntax, see the *Synopsys Technology File and Routing Rules Reference Manual*.

- The logic libraries

  The source files for the logic libraries are Liberty libraries in compiled (.db) format. These files contain the timing, power, and functionality information for the library cells (standard cells or macro cells). Each file represents a characterization point for a specific process, voltage, and temperature.

  Note:

  > This book assumes that the logic libraries use the power and ground (PG) pin syntax. For information about using older logic libraries that use the `rail_connection` attribute to define the PG pin connections or do not define the PG pins, see Appendix C, Using Non-PG Logic Libraries.

- The physical libraries

  The source files for the physical libraries are Library Exchange Format (LEF) files, GDSII files, OASIS files, existing physical libraries, or the FRAM views exported from the IC Compiler tool. These files contain the physical shape data for the library cells.

- TLUPlus files (optional)

  The TLUPlus files contain the RC coefficients for a specific technology. The TLUPlus models enable accurate RC extraction results by including the effects of width, space, density, and temperature on the resistance and capacitance.

  You can store the TLUPlus information either in the cell library or in the design library.

A cell library contains all of the required information, both logical and physical, for a set of cells for a technology. The implementation tools do not directly use the source files used to generate a cell library.

See Also

- Preparing Cell Libraries

Physical Rules File

The *physical rules file* consolidates the placement of cell attribute and rule constraint into an unified file format. For example, when horizontal cell spacing rules are defined through Tcl command and the vertical abutment rules are defined through user attribute and application option, physical rules file consolidates these rules into an unified format.

See Also

- Using Physical Rules File

Technology Library

A *technology library* is a library that contains only the information from the technology file and TLUPlus files. It does not contain any library cells.

See Also

- Cell Library

Aggregate Library

An aggregate library combines multiple cell libraries into a single cell library with a defined search order for the included cell libraries. When you refer to a cell in an aggregate library, the tool uses the defined search order to find the first occurrence of the cell. You can use an aggregate library to group related cell libraries for easier maintenance and distribution.

See Also

- Creating an Aggregate Library

Library Workspace

A library workspace is the in-memory representation of a cell library while it is being built. You use a library workspace to create a new cell library, modify an existing cell library, or create an aggregate cell library. You load the source files into the library workspace, validate the contents, and then commit the workspace to save it as a cell library.

There are two types of library workspaces:

- Root workspace

  This is the main library workspace. Only a single root workspace can be open at one time. If you are creating a single cell library, you work only with a root workspace.

- Subworkspaces

  Subworkspaces are used in the exploration flow to partition source data into multiple cell libraries. Each subworkspace generates a single cell library.

See Also

- Creating a Library Workspace

Pane

A pane is the representation of the timing and power data for a cell. In general, a pane represents the data at a single characterization point (temperature and voltage) for a given process setting. In some cases, the data for multiple characterization points at the same process setting can be merged into a single pane, which is called a *scaling group*.

If you are not using scaling groups, the timing view of a cell contains a pane for each logic library that contains that cell. Each pane is indexed by its voltage and temperature values.

See Also

- Scaling Group

Breakpoints

In a logic library, a timing arc is represented as a lookup table. The index values specified in the lookup table are referred to as the breakpoints for the timing arc. During timing analysis, the tool uses interpolation to determine the timing values for points between the specified breakpoints.

See Also

- Timing Arc Checks

Scaling Group

For a given process, you might have multiple logic libraries that represent different characterization points (voltage and temperature). By default, timing analysis is performed only at these characterization points. If the characterization points for a set of logic libraries form a complete grid, you can create a scaling group, which merges the characterization points into a single pane. A pane that represents a single logic library uses the indexes specified in the logic library for the timing arc, typically input slew and output capacitance. A pane that represents a scaling group has indexes for voltage, temperature, or both, in addition to the indexes specified in the logic library. These higher-dimension delay tables enable very efficient delay calculations in the implementation tools. In addition, they enable interpolation between the voltage and temperature values of the characterization points, which enables efficient multivoltage analysis and optimization.

See Also

- Creating and Using Scaling Groups

Antenna Properties

MOS transistor gate oxides are easily damaged by electrostatic discharge, especially during the manufacturing process. The static charge collected on lower-level wires during multilevel metalization processing can destroy gate oxides and ruin the chip. This issue is called the “antenna problem.”

To protect the MOS device from being damaged, the area of the metal connected to a gate oxide during a given metal mask stage must be limited to a certain value. When the metal area exceeds this value, a reverse-biased diode can be used to provide a discharge path to protect the gate oxide at a cell input.

The diffusion areas of output pins provide protection for gate oxides at input pins. However, at intermediate stages of the manufacturing process, unconnected metal wires extending from the input pins can cause antenna problems to occur. For many process technologies, antenna properties specify the maximum ratio of antenna area (the metal area of dangling connections) to gate area (the gate area of input pins). This ratio can be a constant value or a value derived by an antenna equation.

See Also

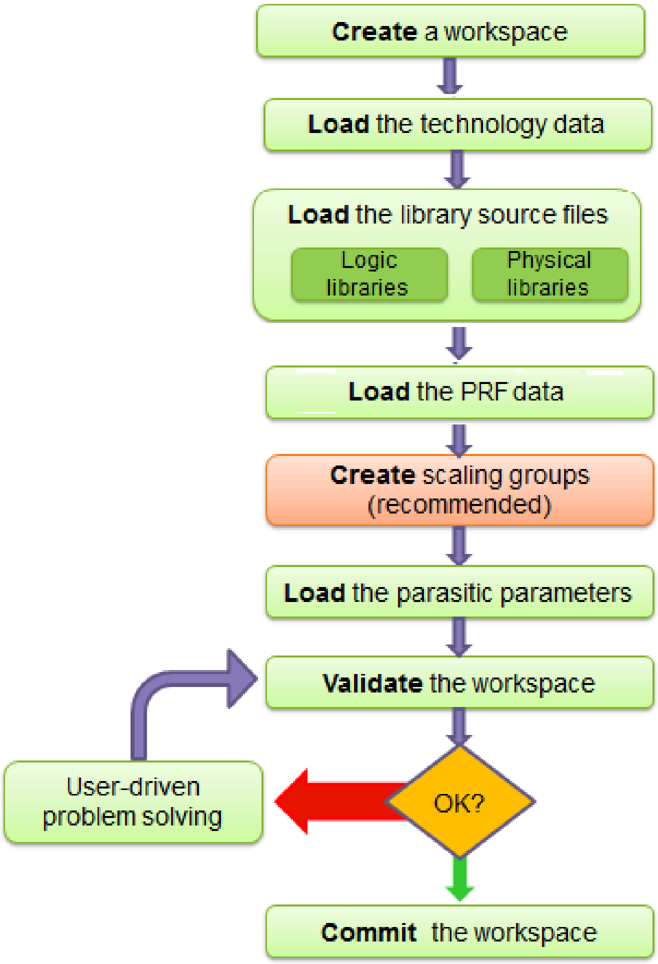
- Automatic Antenna Property Extraction Using the IC Validator Tool
- Defining Antenna Properties on Standard Cells
- Defining Antenna Properties on Hard Macro Cells

Feedback

Library Preparation Flow

Figure 3 shows the library preparation flow used to build a cell library.

Figure 3 Library Preparation Flow



For detailed information about the library preparation flow, see Preparing Cell Libraries.

Feedback

User Interfaces

The Library Manager tool operates in the X windows environment on Linux. It provides a flexible working environment with both a shell command-line interface and a graphical user interface (GUI).

- The shell command-line interface, lm_shell, is a text-only environment in which you enter commands at the command-line prompt. It is typically used for scripts, batch mode, and push-button operations and is always available during a library manager session.
- The GUI provides a wizard to perform library preparation, as well as tools for viewing and editing the cells in your library. The look and feel of the library manager GUI is consistent with the look and feel of other Synopsys GUIs.

The library manager uses the tool command language (Tcl). Using Tcl, you can extend the lm_shell command language by writing reusable procedures and scripts (see the *Using Tcl With Synopsys Tools* manual).

You can start or exit a session in either lm_shell or the GUI, and you can open or close the GUI at any time during a session. The following topics describe how to start and exit the tool using the command-line interface:

- Starting the Command-Line Interface
- Exiting the Library Manager Tool

See Also

- Working With the Library Manager GUI

Starting the Command-Line Interface

To start the lm_shell command-line interface,

1. Include the path to the bin directory in your $PATH variable.
2. Enter the `lm_shell` command in a Linux shell.

   ```
   % lm_shell
   ```

   You can include other options when you start lm_shell. For example,

   - `-file` *`script_file_name`* to execute a script
   - `-x` *`command`* to execute anlm_shell command
   - `-output_log_file` *`file_name`* to create a log file of your session
   - `-h` to display a list of the available options (without starting lm_shell)

At startup, lm_shell performs the following tasks:

1. Creates a command log file.
2. Reads and executes the setup files.
3. Executes any script files or commands specified by the `-f` and `-x` options, respectively, on the command line.
4. Displays the program header and lm_shell> prompt in the shell.

See Also

- Starting the Tool in the GUI
- Using the Command Log File
- Setup Files
- Using Tcl Scripts

Exiting the Library Manager Tool

To exit the Library Manager tool, use the `exit` or `quit` command.

Note:

When you exit the tool from the command line, the tool exits without saving the open library workspace.

See Also

- Exiting the Tool From the GUI

Entering lm_shell Commands

You interact with the library manager by using lm_shell commands, which are based on the tool command language (Tcl) and include certain command extensions needed to implement specific Library Manager functionality. The library manager command language provides capabilities similar to Linux command shells, including variables, conditional execution of commands, and control flow commands. You can

- Enter individual commands interactively at the lm_shell> prompt
- Enter individual commands interactively on the console command line in the GUI
- Run one or more Tcl scripts, which are text files that contain lm_shell commands

When entering a command, an option, or a file name, you can minimize your typing by pressing the Tab key. If you have typed enough characters to specify a unique name, the IC Compiler II library manager completes the remaining characters. If the characters you typed can be used for more than one name, the library manager lists the qualifying names, from which you can select by using the arrow keys and the Enter key.

If you need to reuse a command from the output for a command-line interface, you can copy and paste the portion by selecting it, moving the pointer to the lm_shell command line, and clicking with the middle mouse button.

When you run a command, the library manager echoes the command output (including processing messages and any warnings or error messages) in lm_shell and, if the GUI is open, in the console log view. By default, the output scrolls. To prevent scrolling, enable page mode by setting the `sh_enable_page_mode` variable to `true`.

See Also

- Using Tcl Scripts

Interrupting or Terminating Command Processing

To interrupt command processing and remain in lm_shell, press Ctrl+C.

When you use Ctrl+C, keep the following points in mind:

- If a script file is being processed and you interrupt one of its commands, the script processing is interrupted and no further script commands are processed.
- If you press Ctrl+C three times before a command responds to your interrupt,lm_shell is interrupted and exits with this message:

  ```
  Information: Process terminated by interrupt.
  ```

Some commands and processes cannot be interrupted. To stop these commands or processes, you must terminate lm_shell at the system level. When you terminate a process or the shell, data is not saved.

Getting Information About Commands

The following online information resources are available while you are using the library manager:

- Command help
- Command man pages

Displaying Command Help

Command help consists of either a brief description of an lm_shell command or a list of the options and arguments supported by an lm_shell command.

- To display a brief description of a command, enter the `help` command followed by the command name. For example, to display a brief description of the `read_db` command, use the following command:

  ```
  lm_shell> help read_db
  ```

- To display the options supported by an lm_shell command, enter the command name with the `-help` option. For example, to see the options supported by the `create_workspace` command, use the following command:

  ```
  lm_shell> read_db -help
  ```

Using Application Options

The Library Manager tool uses application options to control the tool behavior. Application options use the following naming convention:

```
category[.subcategory].option_name
```

where *category* is the name of the tool feature affected by the application option. Some application option categories have subcategories to further refine the area affected by the application option.

You use the following commands to work with application options:

- `set_app_options`: Sets one or more application options.
- `reset_app_options`: Resets one or more application options so that they have no value
- `get_app_options`: Lists the available application options.
- `get_app_option_value`: Reports the setting for a specific application option.
- `report_app_options`: Reports application options and their settings.

For more information about working with application options, see Application Options.

Using Variables

In general, the library manager modifies default behavior by using application options rather than application variables; however it does support user-defined Tcl variables, as well as a minimal number of application variables, such as the `search_path` variable.

See Also

- Defining the Search Path
- Using Application Options

Viewing Man Pages

To display the man page for an lm_shell command or application variable, enter the `man` command followed by the command or variable name. For example, to see the man page for the `read_db` command, use the following command:

```
lm_shell> man read_db
```

To display the man page for an lm_shell application option, enter the `man` command followed by the option name. You can also view the following types of summary pages for application options:

- Category summaries

  To view a man page the summarizes all of the application options for a specific category, enter the `man` command followed by *category*_options. For example, to see the man page that summarizes all library manager application options, use the following command:

  ```
  lm_shell> man lm_options
  ```

- Subcategory summaries

  To view a man page the summarizes all of the application options for a specific subcategory, enter the `man` command followed by *category*.*subcategory*_options. For example, to see the man page that summarizes all signoff antenna extraction application options, use the following command:

  ```
  lm_shell> man signoff.antenna_options
  ```

- Command summaries

  To view a man page the summarizes all of the application options for a specific command, enter the `man` command followed by *command*_options. For example, to see the man page that summarizes all application options that affect the `read_db` command, use the following command:

  ```
  lm_shell> man read_db_options
  ```

If you enter the `man` command on the lm_shell command line, the man page is displayed in the shell and in the console log view if the GUI is open. If you enter this command on the console command line in the GUI, the man page is displayed in the GUI man page viewer.

Using Tcl Scripts

You can use Tcl scripts to accomplish routine, repetitive, or complex tasks.

A Tcl script is a text file that contains a sequence of any lm_shell commands. It can also include comments, which are indicated by a pound sign (#) at the beginning of a line.

```
# This is a comment
```

You can use the following methods to run a Tcl script:

- To run a script from the command line, use the `source` command.
- To run a script in the GUI, choose File > Execute Script.
- To run a script when you start the tool, use the `-f` option with the `lm_shell` command.

If an error occurs when running a command in the script, the library manager raises the TCL_ERROR condition, which immediately stops the script execution. To tolerate errors and allow the script to continue executing, either

- Check for TCL_ERROR error conditions with the Tcl `catch` command on the commands that might generate errors.
- Set the `sh_continue_on_error` variable to `true` in the script file.

For information about using Tcl with Synopsys tools, see *Using Tcl With Synopsys Tools*.

See Also

- Starting the Command-Line Interface

Setup Files

A setup file is a special Tcl script that the tool automatically executes at startup. It can contain commands that perform basic tasks, such as initializing application options.

The setup file for the library manager is named .synopsys_lm.setup. The tool looks for this file both in your home directory and in the project directory (the current working directory in which you start the tool). The file is read in the following order:

1. The .synopsys_lm.setup file in your home directory

   Typically the settings in this file define your working environment.

2. The .synopsys_lm.setup file in the project directory

   Typically the settings in this file affect the processing of a specific library.

See Also

- User Interfaces
- Using Application Options
- Using Variables
- Using Tcl Scripts

Using the Command Log File

The command log file records the lm_shell commands processed by the library manager, including setup file commands and application option settings. By default, the tool writes the command log to a file named lm_command.log in the working directory.

To change the name of the command log file, set the `sh_command_log_file` variable in your .synopsys_lm.setup file. If your user-defined or project-specific setup file does not contain this variable, the tool automatically creates the lm_command.log file.

Each session overwrites the existing command log file. To save a command log file, move it or rename it. You can use the command log file to

- Record the library preparation process
- Document any problems in the current session

See Also

- Setup Files

2

Preparing Cell Libraries

To learn how to prepare cell libraries, see the following topics:

- Library Preparation Flows
- Analyzing the Library Source Files
- Building Cell Libraries
- Generating Frame Views
- Using the Exploration Flow
- Allowing Incomplete or Inconsistent Library Data
- Verifying Libraries
- Modifying a Cell Library
- Creating an Aggregate Library
- Modifying an Aggregate Library
- Opening a Cell Library
- Getting Information About Cell Libraries
- Exporting Cell Library Content
- Closing a Cell Library

Library Preparation Flows

The library manager supports the following library preparation flows:

- Normal flow

  Use this flow to create a cell library if you have a single logic library file or a group of logic library files, each of which contains the same set of cells but has timing data for a different characterization point, and physical library source files that contain a superset of the cells in the logic library files. Library vendors typically use this library organization for standard cells and pad cells.

  By default, the generated cell library contains the cells that exist in both the logic and physical library source files. It contains both logical information (timing views) and physical information (frame, design, and layout views) for these cells.

- Technology-only flow

  Use this flow to create a library that contains only the information from the technology file and TLUPlus files. This type of library is referred to as a *technology library*.

- Frame-only flow

  Use this flow to create a library that contains only the physical information for the cells. This type of library is referred to as a *physical library*. It is used as an input to the library preparation process.

  For this flow, you either read only the physical library source files or both the physical and logic library source files. If you read both the physical and logic library source files, the tool uses the logic libraries only to determine the port direction when conflicts occur.

  The generated physical library contains the cells that exist in the physical library source files. It contains only the physical information (frame views and optionally design and layout views) of these cells.

- Extracted timing model (ETM) flow

  Use this flow to create a cell library if you have several extracted timing models, each of which represents a single mode and characterization point for the same design, and a physical library source file that contains the design.

  The generated cell library contains the design represented by the extracted timing model, assuming that physical information is provided for that design. If the physical library source files contain additional designs, they are not included in the generated cell library.

- Physical-only flow

  Use this flow to create a separate physical-only cell library for the cells that exist only in a physical library source file and do not exist in any of the logic library files. A physical-only library typically contains cells such as filler cells, tap cells, flip-chip pad cells, endcap cells, and decoupling capacitor cells.

  The group of files used for this flow includes all of the logic and physical library source files; however, the generated cell library includes only those cells that exist only in the physical library source files.

  Note:

  > You must read at least one logic library when using the physical-only flow; otherwise, workspace validation fails and you cannot save the cell library. To build a library of cells using only physical library source files, use the frame flow.

- Exploration flow

  Use this flow to automatically analyze the library source files and generate a script that you can use to create a cell library.

- Edit flow

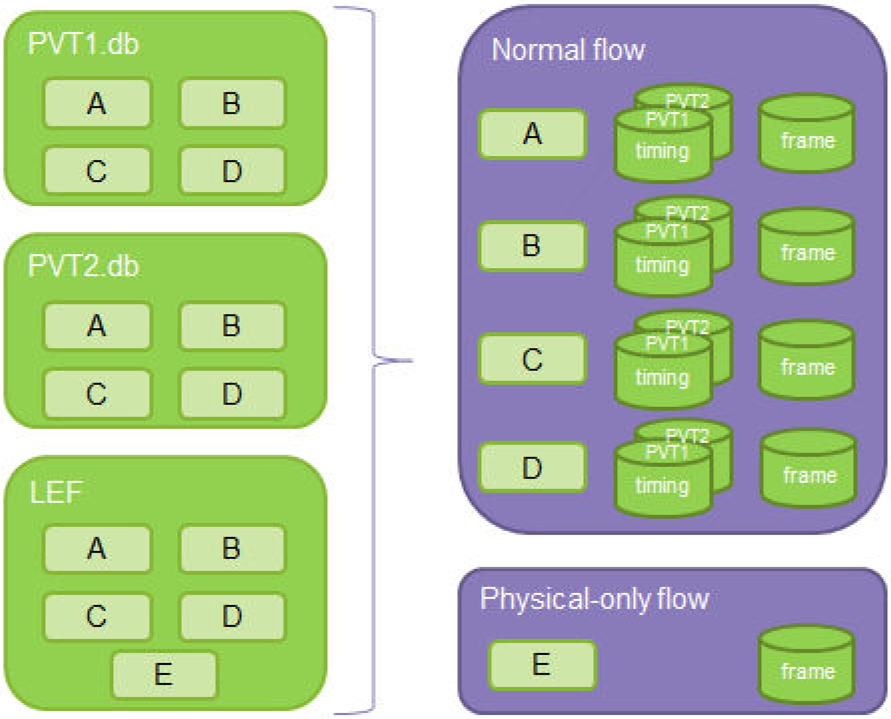
  Use this flow to modify the physical data for cells in an existing cell library.

- Aggregate flow

  Use this flow to combine multiple separate cell libraries into a single cell library.

Figure 4 shows the results of using the normal and physical-only flows on a library workspace that contains two .db files, each of which contains a different characterization point for the same set of cells, and one LEF file.

Figure 4   Normal and Physical-Only Example



See Also

- Cell Library
- Technology Library

Analyzing the Library Source Files

Note:

> This topic describes the manual process for analyzing the library source files. You can use the exploration flow to perform this task automatically; however, you should read this topic to understand the analysis process. For information about using the exploration flow to perform this task, see Using the Exploration Flow.

Before you build the cell libraries, you must analyze your library source files to determine which library preparation flow to use. Depending on your library source files, you might create several cell libraries using various library preparation flows. When you build multiple cell libraries, you need to determine which source files to use for each cell library.

When grouping the library source files, consider the following:

- In general, each cell should be included in both a logic library file and a physical library file.

  The logic library file provides the information for the timing view, while the physical library file provides the information for the layout, design, and frame views.

- If you load multiple logic library files into a library workspace, they must all represent the same process. In addition, the logic library files must meet one of the following conditions:

  - All the logic library files contain the same set of cells but each one provides data for a different temperature and voltage point.
  - Each logic library file contains only a single cell; if the files contain different cells, the group must include a logic library file for each cell at each temperature and voltage point.

- Each cell should exist in only one group, and that group should contain cell data for all necessary temperature and voltage corners.

  Caution:

  > If multiple cell libraries contain a library cell with the same name, the tool uses the data from the first cell library in which it finds the cell, and it ignores that cell's data in all other cell libraries. This phenomenon is referred to as *cell shadowing*.

- A physical library file might belong to several groups, because it might contain more cells than any single logic library file.

  You should ensure that all cells in the physical library files are included in a cell library. Ideally, each cell is included in exactly one cell library; otherwise, users must be very careful when specifying the cell library order for a design library. If a physical library file contains cells that do not exist in any of the logic library files, these cells are considered physical-only cells. You should create a separate cell library for these physical-only cells.

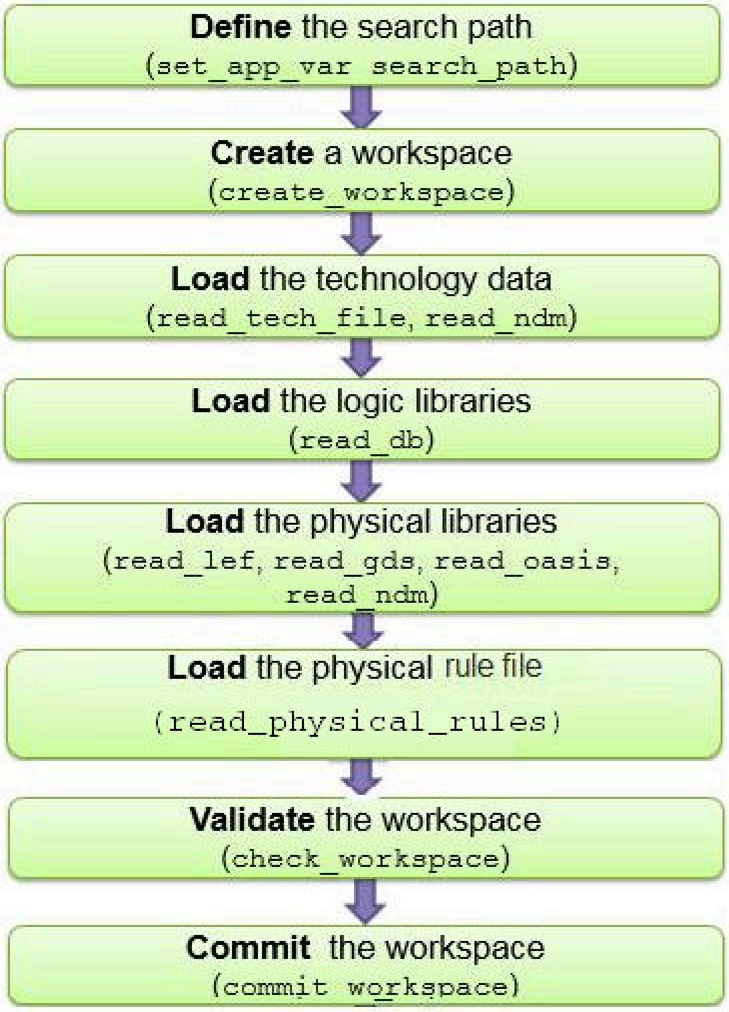
See Also

- Cell Library

Building Cell Libraries

The basic process for building a cell library is the same regardless of the library preparation flow you choose. The only differences are the options that you use when creating the library workspace and the library source files that you load into the workspace. The following figure shows an overview of the library preparation process.

Figure 5 Overview of the Library Preparation Process



To build a technology library, you follow the same flow, but omit steps four through eight, which involve the source files for the library cells.

After a cell library is built, it does not need to be rebuilt unless one of the library source files changes or a new version of the implementation tool requires an updated library.

Feedback

See Also

- Using the Exploration Flow
- Modifying a Cell Library
- Creating an Aggregate Library

Defining the Search Path

The library manager uses a search path to look for files that are specified with a relative path or no path.

To specify the search path, set the `search_path` application variable to the list of directories, in order, in which to look for files. When the library manager looks for a file, it starts searching in the leftmost directory specified in the `search_path` variable and uses the first matching file it finds.

Creating a Library Workspace

To create a library workspace, use the `create_workspace` command. When you run this command to create a new cell library, you must specify the following items:

- The name of the workspace

  For all flows except the edit flow, specify the workspace name without a file extension. For the edit flow, specify the path to an existing cell library.

- The library preparation flow you are using

  By default, the tool creates a workspace for the normal library preparation flow. To specify a different flow, use the `-flow` option, as described in Specifying the Library Preparation Flow.

Note:

> The `create_workspace` command fails if there are any cell libraries in memory. Before running the `create_workspace` command, use the `close_lib` command to close any open libraries.

For example, to use the normal flow to create a cell library named mylib, use the following command to create the library workspace:

```
lm_shell> create_workspace mylib
```

You can optionally specify the following items:

- The technology data for the library

  To provide the technology data when you create the library workspace, use one of the following methods:

  - Load a technology file by using the `-technology` option. This is the preferred method for providing the technology data.
  - Reference a technology library by using the `-use_technology_lib` option.

  For more information, see Loading the Technology Data.

- The scale factor for the library

  The scale factor specifies the number of database units per micron. The default scale factor is 10000, which means that each database unit represents 1 Angstrom. However, the recommended scale factor is the technology length precision as specified by the `lengthPrecision` attribute in the technology file. To use the technology length precision as the scale factor, set the `lib.setting.use_tech_scale_factor` application option to `true` before running the `create_workspace` command.

  For more information about setting the scale factor, see Specifying the Scale Factor.

See Also

- Library Workspace
- Analyzing the Library Source Files
- Using the Exploration Flow
- Modifying a Cell Library
- Creating an Aggregate Library

Specifying the Library Preparation Flow

By default, the `create_workspace` command creates a library workspace for the normal flow. To create a library workspace for a different library preparation flow, use the `-flow` option to specify the flow. Table 2 shows the options to use for the supported library preparation flows.

Table 2 create_workspace Options To Specify the Library Preparation Flow

Library preparation flow	create_workspace -flow keyword
Cell library creation flows	
Normal	`normal`
Technology-only	`normal`
Frame-only	`frame`
Extracted timing model	`etm_moded`
Physical-only	`physical_only`
Exploration	`exploration`
Cell library verification flow	
Verification	`verification`
Cell library modification flow	
Edit	`edit`
Aggregate library creation flow	
Aggregate	`aggregate`

For example, to use the normal flow to create a cell library named mylib, use the following command:

```
lm_shell> create_workspace mylib
```

To use the physical-only flow to create a cell library named mylib_po, use the following command:

```
lm_shell> create_workspace -flow physical_only mylib_po
```

To create a workspace for the exploration flow, use the following command:

```
lm_shell> create_workspace -flow exploration mylib_explore
```

Note:

> Unless you are using the exploration flow, you can have only one library workspace open at a time. To remove the current library workspace from memory without committing it to a saved cell library, use the `remove_workspace` command.

See Also

- Library Workspace
- Analyzing the Library Source Files
- Using the Exploration Flow
- Verifying Libraries
- Modifying a Cell Library
- Creating an Aggregate Library

Specifying the Scale Factor

The scale factor specifies the number of database units per micron. An implementation tool uses the scale factor to convert floating point numbers into integers when storing data in the cell libraries and design libraries. When the tool saves a floating point number, it multiplies the number by the scale factor and then rounds the number to an integer. When the tool retrieves the number, it divides the number by the scale factor. For example, if the scale factor is 1000, the tool stores a value of 0.0032 as 3 (0.0032 * 1000 = 3.2, which rounds to 3). The tool retrieves this value as 0.003 (3 / 1000).

If the scale factor is not large enough, the precision of specified coordinates might be affected. For example, assume that you create a net shape with the following command:

```
prompt> create_shape -shape_type rect -layer M1 \
   -boundary {{0.500 0.500} {0.5053 0.5053}}
```

With a scale factor of 1000, the bounding box is stored as {500 500} {505 505}, instead of {5000 5000} {5053 5053}. With a scale factor of 10000, the bounding box is stored as {5000 5000} {5053 5053}.

The scale factor for a library workspace must be a multiple of both the length precision of the technology file associated with the workspace and the scale factor of the physical library source files loaded into the workspace.

The default scale factor is 10000, which means that each database unit represents 1 Angstrom. However, the recommended scale factor is the technology length precision as specified by the `lengthPrecision` attribute in the technology file. To use the technology length precision as the scale factor, set the `lib.setting.use_tech_scale_factor` application option to `true` before running the `create_workspace` command.

```
lm_shell> set_app_options \
   -name lib.setting.use_tech_scale_factor -value true
```

If neither the default scale factor nor the technology length precision is appropriate for your library, you can explicitly specify the scale factor by using the `-scale_factor` option with the `create_workspace` command.

To determine the scale factor for an existing cell library, query its `scale_factor` attribute.

Loading the Technology Data

You can load the technology data for a cell library either by reading a technology file into the library workspace, which is the preferred method, or by associating a technology library with the library workspace.

- To load a technology file into a library workspace when you create the workspace, use the `-technology` option with the `create_workspace` command.

  For example, to use the normal flow to create a cell library named mylib that gets its technology data from the my.tf technology file, use the following command:

  ```
  lm_shell> create_workspace -technology my.tf mylib
  ```

- To load a technology file into an existing library workspace, use the `read_tech_file` command.

  For example, to load the my.tf technology file into the current library workspace, use the following command:

  ```
  lm_shell> read_tech_file my.tf
  ```

- To load the add-on technology file and merge it to the current library, use the following command:

  ```
  lm_shell> read_tech_file 3dic.tf -merge
  ```

  Where `3dic.tf` is the add-on file which is merged with the `base.tf` file to generate one ndm technology object.

For example, to create a library using multiple technology files in Library Manager flow, use the following commands:

```
create_workspace test -technology base.tf
write_tech_file output0.tf
read_tech_file 3dic1.tf -merge
read_tech_file 3dic2.tf -merge
write_tech_file output.tf
```

To open a library and update incrementally using multiple technology files in IC Compiler II, Fusion Compiler, and Library Compiler flow, use the following commands:

```
open_lib test.ndm ;# test.ndm created using baseline tech file, use
 open_physical_lib in LC
read_tech_file 3dic1.tf -merge
read_tech_file 3dic2.tf -merge
```

Note:

The `write_tech_file` command writes out the merged technology file.

- To associate a technology library with a library workspace when you create the workspace, use the `-use_technology_lib` option with the `create_workspace` command.

  For example, to use the normal flow to create a cell library named mylib that gets its technology data from the mytech.ndm technology library, use the following command:

  ```
  lm_shell> create_workspace -use_technology_lib mytech.ndm mylib
  ```

- To associate a technology library with an existing library workspace, use the `read_ndm` command to read the technology library into the library workspace.

  For example, to associate the mytech.ndm technology library with the current library workspace, use the following command:

  ```
  lm_shell> read_ndm mytech.ndm
  ```

When you read a technology file into a library workspace, the tool performs syntax and semantic checks on the contents of the technology file. The technology file checker has two modes:

- User mode (the default)

  In this mode, the tool downgrades the message severity for suspected errors for the general user.

- Developer mode

  In this mode, the tool increases the message severity for suspected errors for the technology file developer to correct or waive. To enable this mode, set the `file.tech.library_developer_mode` application option to `true`.

When you load technology data into an existing library workspace,

- The new technology data overwrites any existing technology data, whether from a previous technology file or specified with Tcl commands
- The library manager verifies that the frame views in the library workspace are consistent with the new technology data

Caution:

> The technology file does not contain all of the technology data required to perform placement and routing. For information about specifying the missing information, see Using Physical Rules File.

See Also

- Technology Data Access
- Creating a Library Workspace

Replacing Tcl With Physical Rules File

Physical rules file replaces Tcl files for track, site definitions, GDS settings and so on.

The physical rules file supports and consolidates library preparation level and design level physical properties like placement rule constraints, physical cell and pin properties, site and layer properties, and stream-in instructions into a unified file format.

For more information about physical rules file, see Using Physical Rules File.

Completing the Technology Data

The technology file does not contain all of the data required for placement and routing. The following topics describe how to add the missing technology data to a cell library:

- Preparing Site Definitions
- Preparing Routing Layers

Note:

In many cases, the LEF file defines the required attributes. You must manually set these attributes only if you load the physical data by reading a GDSII or OASIS file or if the LEF file does not define these attributes.

Preparing Site Definitions

The following site attributes are required for placement, but are not defined in the technology file:

- `isDefault` site attribute

  This attribute identifies the default site for floorplanning. It must be set only on the default site.

  For example, to set the site definition named unit as the default site for floorplanning, use the following command:

  ```
  lm_shell> set_attribute [get_site_defs unit] is_default true
  ```

- `symmetry` site attribute

  This attribute defines the site symmetry requirement for cell placement. It must be set on all sites defined in the technology file. The attribute value is a list that contains one or more of the following values:

  - `X` - the site can be flipped about the x-axis
  - `Y` - the site can be flipped about the y-axis
  - `R90` - the site can be rotated 90 degrees

  For example, to specify that the site definition named unit can be flipped about the x-axis, set its `symmetry` attribute as follows:

  ```
  lm_shell> set_attribute [get_site_defs unit] symmetry {X}
  ```

  To specify that it can be flipped about both the x- and y-axis and can be rotated 90 degrees, use the following command:

  ```
  lm_shell> set_attribute [get_site_defs unit] symmetry {X Y R90}
  ```

In general, the LEF file defines these site attributes, and the library manager automatically sets them when you read the LEF file. If you load the physical data by reading a GDSII or OASIS file or if the LEF file does not define these attributes, use the `set_attribute` command to set them manually.

To report the site definitions, use the `report_site_defs` command.

Note:

> The cell library contains only single-height site definitions. The Fusion Compiler and IC Compiler II tools use the single-height site definition for both single-height cells and multiple-height cells.

Preparing Routing Layers

The following layer attributes are required for routing, but are not defined in the technology file:

- `routing_direction` layer attribute

  This attribute identifies the preferred routing direction for the layer. Valid values for this attribute are `horizontal`, `vertical`, `diagonal_45`, and `diagonal_135`. For example, to set the preferred routing direction for the M2 layer as horizontal, use the following command:

  ```
  lm_shell> set_attribute [get_layers M2] \
     routing_direction horizontal
  ```

  Note:

  > You can also specify the `routing_direction` layer attribute in the physical rules file. For information about the syntax of the physical rules file, see Using Physical Rules File. For information about loading a physical rules file, see Loading the Physical Rules File.

- `track_offset` layer attribute

  This attribute defines the offset in microns used to create the routing grid for the layer. The first routing track is located at the specified distance from the placement grid origin. For example, to set the track offset for the M2 layer as 0.1 micron, use the following command:

  ```
  lm_shell> set_attribute [get_layers M2] track_offset 0.1
  ```

In general, the LEF file defines these layer attributes, and the library manager automatically sets them when you read the LEF file. If you load the physical data by reading a GDSII or OASIS file or if the LEF file does not define these attributes, use the `set_attribute` command to set them manually.

Loading Parasitic Parameters

The RC coefficients for a specific technology are represented by TLUPlus models, which are stored in a TLUPlus file or a common technology file. The TLUPlus models enable accurate RC extraction results by including the effects of width, space, density, and temperature on the resistance and capacitance coefficients.

To read the TLUPlus models into the library workspace, use the `read_parasitic_tech` command. You must specify the files that contain the TLUPlus models by using the `-tlup` option. If you specify the files with a relative path or with no path, the library manager uses the search path defined with the `search_path` variable to locate the files.

If the layer names in the TLUPlus or common technology file do not match the layer names in the technology file, you must use the `-layermap` option to specify the layer mapping file. The layer mapping file uses the following format:

```
conducting_layers
tf_metal_layer_name1      ITF_metal_layer_name1
...
tf_metal_layer_namen      ITF_metal_layer_namen

via_layers
tf_via_layer_name1      ITF_via_layer_name1
...
tf_via_layer_namen      ITF_via_layer_namen
```

To include comments in the layer mapping file, start the line with an asterisk (*) or pound sign (#).

After you read in a TLUPlus or common technology file, it is identified by its parasitic technology model name. By default, the parasitic technology model name is the base name of the specified file; however, you can specify a different name by using the `-name` option.

For example, to read in a TLUPlus file named my.tlup using a layer mapping file named my.layermap and store it in the workspace with a parasitic technology model name of para1, use the following command:

```
lm_shell> read_parasitic_tech -tlup my.tlup \
   -layermap my.layermap -name para1
```

Feedback

Loading Signal Electromigration Constraints

The signal electromigration constraints are defined in either an Interconnect Technology Format (ITF) file or an Advanced Library Format (ALF) file. To load these constraints into the library workspace, use the `read_signal_em_constraints` command.

- By default, the command assumes that the file is in ITF format.

  For example, to load the electromigration constraints from an ITF file named em.itf, use the following command:

  ```
  lm_shell> read_signal_em_constraints em.itf
  ```

- To read an ALF file, use the `-format ALF` option.

  To load the electromigration constraints from an ALF file named em.alf, use the following command:

  ```
  lm_shell> read_signal_em_constraints -format ALF em.alf
  ```

If the library workspace already contains signal electromigration constraints, they are replaced by the constraints in the new file.

Loading the Logic Data

The logic data is stored in logic libraries, which contain the timing, power, and functionality information for the standard cells and macro cells in the cell library. The supported timing model formats are generic CMOS, nonlinear delay model (NLDM), and Composite Current Source (CCS).

A cell library should contain this information for all necessary characterization points (voltages and temperatures for a given process setting). If these characterization points are represented in multiple logic libraries, you must load all of these logic libraries into a single library workspace.

Caution:

> If multiple cell libraries contain a library cell with the same name, the tool uses the data from the first cell library in which it finds the cell, and it ignores that cell's data in all other cell libraries. This phenomenon is referred to as *cell shadowing*.

Before loading logic libraries into the library workspace, ensure that they have been validated with the Library Compiler `check_library` command. Note that you must have a Library Compiler license to run this command.

To load logic libraries into the library workspace, use the `read_db` command.

For example, to load the db1_pvt1.db file, use the following command:

```
lm_shell> read_db db1_pvt1.db
```

To load all the .db files in the current directory, use the following command:

```
lm_shell> read_db [glob *.db]
```

If you specify the files with a relative path or with no path, the tool uses the search path defined with the `search_path` variable to locate the files.

When you load a logic library into a library workspace, the library manager performs the following tasks:

- Creates timing views in the library workspace for each of the blocks in the library source file
- Saves the logic library attribute settings for the library, library cells, and library cell pins in the library workspace

If you load more than one logic library into a workspace, the first logic library that you read is considered the *base library*. The tool uses the base library for many consistency checks among the logic libraries loaded for a workspace. If an attribute is defined in more than one logic library, the tool uses the value from the first logic library in which the attribute is defined.

By default, the tool processes all of the cells defined in the library source file. To process a subset of the cells, filter the cells as described in Specifying Which Blocks to Process.

By default, the generated cell library contains the timing views generated from all of the logic libraries loaded into the library workspace. To filter the logic libraries based on your design operating corners, use the `set_pvt_configuration` command, as described in Filtering Logic Libraries Based on Operating Corners.

See Also

- Pane
- Defining the Search Path
- Identifying the Process Associated With a Logic Library

Identifying the Process Associated With a Logic Library

When you load a logic library, the library manager uses the following information to identify the process associated with the pane:

- The process label

  The library manager uses the following order of precedence to determine the process label:

  - The `-process_label` option specified with the `read_db` command
  - The `process_label` attribute defined in the `operating_conditions` group for the default operating conditions in the logic library

- The process number

  The library manager determines the process number from the `nom_process` attribute for the library. If this attribute is not specified in the library, the library manager sets the process number to 1.0.

If two or more libraries have the same process label, process number, voltage values, and temperature values, the tool uses only the pane generated for the first library, and ignores the rest. To prevent panes from being ignored, assign a unique process identifier to each pane. To set the process number or process label after loading a logic library, use the `set_process` command.

For example, to read the db1.db logic library and set a process label of PVT1 on the generated pane, use the following command:

```
lm_shell> read_db -process_label PVT1 db1.db
```

To set a process label of PVT2 on the generated pane after reading the logic library, use the following command:

```
lm_shell> set_process -libraries {db1} -label PVT2
```

Filtering Logic Libraries Based on Operating Corners

By default, the generated cell library contains the timing information from all logic libraries loaded into the library workspace. To include only the timing information that matches the valid operating corners (process, voltage, and temperature values) for your design, define a PVT configuration for the cell library.

A PVT configuration is a sequence of rules, each of which specifies the process labels, process numbers, voltages, and temperatures to match. If you want all the data that matches a specific set of operating corners, a single rule is sufficient. If you want to match a specific subset of many operating corners, you must specify multiple rules.

To define a PVT configuration rule, use the set_pvt_configuration command.

- To create a new rule, use the -add option.

  By default, the command uses the rule_*n* naming convention. To specify the rule name, use the -name option.
- To modify an existing rule, use the -rule option to specify the rule name.
- To add filters to a rule, use one or more of the following options: -process_labels, -process_numbers, -voltages, and -temperatures.
- To remove filters from a rule, use the -clear_filter option.

For example, to create a PVT configuration rule that allows any process setting, voltages of 0.65 or 1.32 volts, and a temperature of 25 degrees, use the following command:

```
lm_shell> set_pvt_configuration -add \
   -voltages {0.65 1.32} -temperatures {25}
```

Note that this command does not affect loading of the logic libraries. It affects only which logic libraries are actually used by the cell library. If you change the PVT configuration, the library manager adjusts the logic library usage appropriately.

To see the logic libraries loaded into the library workspace, use the get_libs command. To see which logic libraries are actually used by the library workspace based on the PVT configuration, use the report_workspace command.

Loading Extracted Timing Models

An extracted timing model (ETM) contains the timing information for a design for a single mode and characterization point. The extracted timing model must be in .db format and must meet the following additional requirements:

- It must be generated by the PrimeTime tool.
- It must contain a single design.
- It must not contain a mode definition.
- The dont_touch and dont_use attributes for the design must be set to true.
- The design must have at least one of the following attribute settings:
  - The timing_model_type attribute has a value of extracted
  - The interface_timing attribute has a value of true

To load extracted timing models into the library workspace to create timing views, use the read_db command to load the models in .db format.

You must use the `-mode_label` option with these commands to specify the mode associated with the timing arcs and generated clocks defined in the model.

For example, to read the .db file for the model1 extracted timing model, use the following command:

```
lm_shell> read_db -mode_label functional model1.db
```

If you specify the model files with a relative path or with no path, the tool uses the search path defined with the `search_path` variable to locate the files.

Loading the Physical Data

The physical data includes the geometric shapes for the library cells (standard cells or macro cells). The library manager uses this information to generate the place-and-route information (frame views) for the cells.

The physical data can be in LEF, GDSII, or OASIS format. In addition, you can load the physical data from an existing physical library or cell library, or from the FRAM views exported from the IC Compiler tool. To learn about loading physical libraries, see the following topics:

- Reading LEF Files
- Reading GDSII or OASIS Files
- Loading Physical Data from an Existing Library
- Loading Physical Data from a Design Library
- Importing Milkyway FRAM Views From the IC Compiler Tool

Caution:

> In some cases, the physical library source files do not provide all of the physical information required to perform placement and routing. For information about specifying the missing information, see Using Physical Rules File.

Reading LEF Files

LEF files contain the shape information for blocks, including the block type, associated site definition, pins, and connectivity. To read a LEF file into the library workspace, use the `read_lef` command. This command supports all versions of LEF through version 5.8. When the command reads a LEF file, it creates design views for all of the blocks in the file.

For example, to read the myphys.lef file, use the following command:

```
lm_shell> read_lef myphys.lef
```

If you specify the LEF file with a relative path or with no path, the library manager uses the search path defined with the `search_path` variable to locate the file.

By default, the `read_lef` command

- Processes all of the cells defined in the library source file

  To process a subset of the cells, filter the cells, as described in Specifying Which Blocks to Process.

- Creates a design view for each processed block

  By default, if a block already has a physical representation in the library workspace, the tool creates a design view with a new name. To change this default behavior, either use the `-merge_action` option to change the behavior for all blocks or use a block mapping file to specify the behavior for specific blocks. For details, see Handling Duplicate Physical Blocks.

  By default, the `read_lef` command derives the block boundary from the OVERLAP layer in the LEF file, if it exists. If it does not exist, the command derives the rectangular block boundary from the SIZE statement in the LEF file. To derive the rectangular cell boundary from the SIZE statement even if the OVERLAP layer exists, use the `-cell_boundary by_cell_size` option.

  If a block has PROPERTY definitions in the LEF file, the `read_lef` command saves the properties as user-defined attributes of the block. By default, the command saves all the block properties; to save only specific block properties, specify the property names with the `-properties` option.

  When the library manager reads a LEF file, it stores certain settings as attributes on the blocks or block objects. To use the LEF file only to update the attribute settings on the existing blocks in the library workspace, and not to create design views, use the `-merge_action attributes_only` option.

- Assigns a site definition to each processed block

  By default, the `read_lef` command assigns the site definitions for a block by matching the block's size to the site definition sizes. To match, the height and width of a block must be an integer multiple of the height and width of the site definition. The command uses the following priority to select the site definition for a block:

  1. The default site definition

  2. The block's site definition in the LEF file

  3. The smallest site definition

  To force the command to use the block's site definition in the LEF file, use the `-preserve_lef_cell_site` option.

If the height and width of a block does not match the size of any site definition, the tool issues an error message.

- Fails if the LEF file contains layer information that conflicts with the technology file

See Also

- Defining the Search Path
- Validating the LEF File
- Determining the Block Types When Reading LEF Files
- Determining the Electrical Equivalence for PG Terminals When Reading LEF Files

Validating the LEF File

To analyze an input LEF file before loading it into the workspace, enable check-only mode by using the -syntax_only option. The check-only mode provides diagnostic information about the correctness and integrity of the LEF file. The check-only mode does not load any information into the workspace.

```
lm_shell> read_lef -syntax_only myphys.lef
```

Determining the Block Types When Reading LEF Files

The read_lef command uses the mapping shown in Table 3 to convert the LEF CLASS properties to design_type attributes for the processed blocks.

Table 3 LEF CLASS Property to design_type Attribute Mapping

LEF syntax		design_type attribute
CLASS property	**Subtype**	
BLOCK	N/A	macro
	BLACKBOX	black_box
	SOFT	module
CORE	N/A	lib_cell
	ANTENNACELL	diode
	FEEDTHRU	feedthrough
	SPACER	filler
	TIEHIGH	lib_cell
	TIELOW	lib_cell
	WELLTAP	well_tap
COVER	N/A	cover
	BUMP	flip_chip_pad

Table 3 LEF CLASS Property to design_type Attribute Mapping (Continued)

LEF syntax		design_type attribute
CLASS property	**Subtype**	
ENDCAP	BOTTOMLEFT	corner
	BOTTOMRIGHT	corner
	POST	end_cap
	PRE	end_cap
	TOPLEFT	corner
	TOPRIGHT	corner
PAD	N/A	pad
	AREAIO	flip_chip_driver
	INOUT	pad
	INPUT	pad
	OUTPUT	pad
	POWER	pad
	SPACER	pad_spacer
RING	N/A	macro

Determining the Electrical Equivalence for PG Terminals When Reading LEF Files

By default, the `read_lef` command does not calculate the electrical equivalence (EEQ) setting for the PG terminals. To calculate the EEQ setting for the PG terminals, use the `-create_eeq_setting_for_block_and_pad` option. By default, when the library manager calculates the EEQ setting, it considers the direction of the PG port. To ignore the port direction, use the `-ignore_pg_direction_for_eeq` option.

Resolving Site Definition Conflicts When Reading LEF Files

If the LEF file contains site information that differs from the site information in the technology file loaded into the library workspace, the `read_lef` command fails to load the LEF file. Use one of the following methods to resolve the conflicts so you can load the LEF file:

- If the files contain sites with the same definitions, but different names, you can specify the site name mapping by using the `-convert_sites` option with the `read_lef` command. For example, to change the site name in the myphys.lef file from unit to new_unit, use the following command:

  ```
  lm_shell> read_lef -convert_sites {unit new_unit} myphys.lef
  ```

- If the files contain sites with the same names, but different definitions, you can enable the tool to automatically rename conflicting site names by setting the

Feedback

`file.lef.auto_rename_conflict_sites` application option to `true` before running the `read_lef` command.

When you enable this option, the tool uses the following naming convention to rename the conflicting site names:

```
lef_file_name.site_name[_n]
```

The numeric suffix is added only if required to make the site name unique.

Reading GDSII or OASIS Files

GDSII and OASIS files are similar in that they contain physical shape information for the blocks, but do not contain information about the block type, associated site definition, pins, or connectivity.

- To read one or more GDSII files into the library workspace and perform connectivity analysis, use the `read_gds` command.
- To read one or more OASIS files into the library workspace and perform connectivity analysis, use the `read_oasis` command.

If you specify a library source file with a relative path or with no path, the library manager uses the search path defined with the `search_path` variable to locate the file.

For example, to read the myphys.gds file, use the following command:

```
lm_shell> read_gds myphys.gds
```

When the tool reads a library source file in GDSII or OASIS format, it assumes that all of the blocks are standard cells and all of the pins are signal pins. It generates a layout view for each block in the file. If it finds a duplicate block, it creates a layout view with a new name. The tool then performs connectivity analysis to determine the pins and connectivity information and generates a design view for each block.

Caution:

> GDSII and OASIS files do not provide all of the physical information required to perform placement and routing. At a minimum, you must specify the block type and site definition for the blocks in the GDSII or OASIS file.

The following topics describe how to modify the behavior of the `read_gds` and `read_oasis` commands:

- Resolving Layer Differences
- Specifying Which Blocks to Process
- Handling Duplicate Physical Blocks
- Specifying the Block Types When Reading GDSII or OASIS Files

- Specifying the Site Definitions When Reading GDSII or OASIS Files
- Defining the Standard Cell Boundaries
- Identifying Power and Ground Pins
- Completing the Information for I/O Cells
- Editing Layout Views Before Performing Connectivity Analysis
- Controlling the Connectivity Analysis of Layout Views

Resolving Layer Differences

When you read a GDSII or OASIS file, the tool assumes that the layer names are the same in the library source file and the technology file loaded into the library workspace.

- If the layers are the same, but have different names, you must provide a layer mapping file, as described in Mapping Source File Layers to Technology File Layers.
- If the library source file contains layers that are not defined in the technology file, the tool creates new layers. For information about other ways to handle this situation, see Handling Extra Layers in the Library Source File.

Mapping Source File Layers to Technology File Layers

If the layer names differ between the library source file and the technology file, you must provide a layer mapping file. For information about the syntax of the layer mapping file, see Layer Mapping File Syntax.

Use one of the following methods to specify the layer mapping file:

- Use the `set_layer_map_file -format gds` command to load the layer mapping file into the library workspace

  ```
  lm_shell> set_layer_map_file -format gds -map_file file_name
  ```

  When you load a layer mapping file into the library workspace, it is used by both the `read_gds` and `read_oasis` commands.

- Use the `-layer_map` option with the `read_gds` or `read_oasis` command

  ```
  lm_shell> read_gds -layer_map file_name myphys.gds
  ```

Handling Extra Layers in the Library Source File

To prevent the tool from creating new layers, use the `-read_always false` option with the `read_gds` or `read_oasis` command. When you use this option, the command fails if the library source file contains layers that are not defined in the technology file. To modify this behavior, use one of the following options with the `-read_always false` option:

- `-mapped_layers_only`

  In this case, the command loads only those layers specified in the layer mapping file and ignores all other layers defined in the library source file.

- `-ignore_missing_layers`

  In this case, the command loads only those layers defined in the technology file and ignores all other layers defined in the library source file.

You can also set these options in the layer mapping file, as described in Layer Mapping File Syntax. If you specify these options both on the command line and in the layer mapping file, the settings in the layer mapping file override the command-line settings.

Specifying the Block Types When Reading GDSII or OASIS Files

GDSII and OASIS files do not specify the block type for the blocks defined in the file. When you read a GDSII or OASIS file, the tool assumes all blocks are standard cells and sets their `design_type` attribute to `lib_cell`.

Use one of the following methods to specify the block type for the blocks in a GDSII or OASIS file:

- Use the `cell_type` statement in the physical rules file

  For information about the syntax of the physical rules file, see Using Physical Rules File. For information about loading a physical rules file, see Loading the Physical Rules File.

- Use a block mapping file

  When you run the `read_gds` or `read_oasis` command, use the `-block_map` option to specify the block mapping file. For information about the syntax of the block mapping file, see Block Mapping File Syntax.

- Use the `set_attribute` command to set the `design_type` attribute on the library cell

Specifying the Site Definitions When Reading GDSII or OASIS Files

GDSII and OASIS files do not specify the site definition associated with the blocks defined in the file. To enable successful placement and legalization, you must associate each standard cell with a site definition.

Use one of the following methods to specify the site definition for the blocks in a GDSII or OASIS file:

- Use the `site` statement in the physical rules file

  For information about the syntax of the physical rules file, see Using Physical Rules File. For information about loading a physical rules file, see Loading the Physical Rules File.

- Use a block mapping file

  When you run the `read_gds` or `read_oasis` command, use the `-block_map` option to specify the block mapping file. For information about the syntax of the block mapping file, see Block Mapping File Syntax.

- Use the `set_attribute` command to set the `site_name` attribute on the library cell.

  For example, to associate the site definition named unit with all cells in the workspace, use the following command:

  ```
  lm_shell> set_attribute -objects [get_lib_cells */*/design] \
     -name site_name -value unit
  ```

Note:

> Use the single-height site definition, which is often named *unit*, for both single-height and multiple-height cells. The implementation tool automatically identifies the polarity of the power and ground pins and places the single-height and the multiple-height cells accordingly.

To report the site definitions in the technology data, use the `report_site_defs` command.

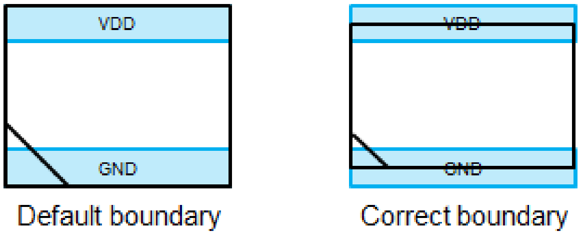
Defining the Standard Cell Boundaries

The height of a standard cell must be the same as the height of its site definition and the width must be an integer multiple of the width of its site definition. When you read a GDSII or OASIS file, the tool determines the boundary for each standard cell. If a boundary layer is defined, the tool uses it to determine the standard cell boundary. Otherwise, the tool generates a default boundary.

In some cases, the cell boundary determined by the tool is not correct and you must adjust it.

- If the boundary for a cell is not specified in the library source file and the cell layout assumes shared power and ground pins, the default boundary generated by the tool is incorrect. You must adjust the cell boundary by reducing it to the center line of the pin geometries. Figure 6 shows the default and adjusted boundaries for a standard cell with shared power and ground pins.

Figure 6 Boundary for Standard Cell With Shared Power and Ground Pins



Use one of the following methods to shrink the default boundary:

- To automatically shrink the default boundary to the center line of the pin geometries for all blocks, use the `-centerline_boundary` option with the `read_gds` or `read_oasis` command.
- To shrink the default boundary for specific blocks, specify the `-centerline_boundary` option in the block mapping file. For information about the syntax of the block mapping file, see Block Mapping File Syntax. To specify the block mapping file, use the `-block_map` option with the `read_gds` or `read_oasis` command.

- If the boundary for a cell is defined incorrectly in the library source file, manually specify the boundary by using the `set_attribute` command to modify the `bbox` attribute of the library cell. For example, the script shown in Example 1 reduces the boundary for a set of library cells by 0.115 units on the left side, 0.051 units on the bottom, 0.115 units on the right side, and 0.109 units on the top.

Example 1 Script to Resize Cell Boundaries

```
set cells_to_resize \
   [get_lib_cells  -include_subcells {libraryname/*/design } ]
set resize_parameters {-0.115 -0.051 -0.115 -0.051}
foreach_in_collection c $cells_to_resize {
   set cname [get_attribute -objects [get_lib_cells $c] -name name]
   puts "INFO: Resize boundary for $cname by \{$resize_parameters\}"
   set existing_boundary \
      [get_attribute -objects [get_lib_cells $c] -name boundary_bbox]
   set new_boundary \
      [get_attribute -objects \
         [get_attribute [resize_polygons \
            -objects [create_poly_rect -boundary $existing_boundary] \
               -size "$resize_parameters"] \
            -name poly_rects] \
         -name point_list]
   set_attribute [get_lib_cells $c] boundary $new_boundary
}
```

Identifying Power and Ground Pins

GDSII and OASIS files do not specify the pin types. When you read a GDSII or OASIS file, the tool assumes that all pins in the library source file are signal pins. To identify power and ground pins, you must map the pin name to its type. The mapping applies to all cells in the GDSII or OASIS files.

You can specify the pin name mapping either in the physical rules file or by setting an application option.

- To specify the pin name mapping in the physical rules file, use the following syntax:

```
stream_in_instructions () {
   type : gds | oasis | both;
   port_type_map(mapping_list);
}
```

- To specify the pin name mapping using an application option, use the following syntax:

```
lm_shell> set_app_options -name file.format.port_type_map \
   -value { mapping_list }
```

  where *format* is either `gds` or `oasis`.

In either method, each item in the *mapping_list* argument uses the following format:

```
"port_type pin_name"
```

where

- *port_type* is one of `power`, `gound`, `pwell`, or `nwell`

In the physical rules file, items in the *mapping_list* argument are separated by a comma. In the application option, they are separated by a space.

For example, to set all pins named VDD as power pins and all pins named VSS as ground pins when reading GDSII files, either

- Include the following statements in the physical rules file:

  ```
  stream_in_instructions () {
     type : gds;
     port_type_map("power VDD", "ground VSS");
  }
  ```

- Use the following command:

  ```
  lm_shell> set_app_options -name file.gds.port_type_map \
     -value { "power VDD" "ground VSS" }
  ```

See Also

- Using Physical Rules File

Completing the Information for I/O Cells

GDSII and OASIS files contain only cell geometry information; therefore, if you load the physical data by reading a GDSII or OASIS file, you must specify additional physical information that is required for placement and routing.

I/O cells must have the following attributes to identify their physical characteristics:

- `design_type` library cell attribute

  This attribute specifies the I/O cell type. I/O cells must have one of the following settings for the `design_type` attribute: `corner`, `pad`, `pad_spacer`, `flip_chip_driver`, or `flip_chip_pad`.

  Use one of the following methods to set this attribute:

  - Use the `cell_type` statement in the `cell` group in the physical rules file
  - Use the `set_attribute` command

- reference_orientation library cell attribute

  This attribute specifies the orientation for proper placement of the I/O cell in the bottom I/O guide. Modify this attribute to correct any I/O orientation issues.

- class terminal attribute

  This attribute specifies the I/O pin type and is required for the flip-chip flow. Valid values for the class attribute are bump, core, and none.

Editing Layout Views Before Performing Connectivity Analysis

In some cases, you might need to edit the layout views before performing connectivity analysis. To do this,

1. Read the library source files without performing connectivity analysis by using the -trace_option none option when you run the read_gds or read_oasis command.
2. Edit the layout views as described in Editing the Physical Layout.
3. Perform connectivity analysis by running the trace_connectivity command.

Controlling the Connectivity Analysis of Layout Views

The tool uses the text labels in the library source file to associate metal shapes with a net or pin. If the text is not on the same layer as the metal shape, you must map the text layers to the metal layers, as described in Mapping Text Layers. If certain layers are used as exclude layers, which define geometries to remove from metal layers during connectivity tracing, you must map the exclude layers to the metal layers, as described in Mapping Exclude Layers.

The tool traces the connectivity of the labeled metal shapes to identify additional metal shapes associated with that net or pin. It can trace connectivity for shapes that overlap pin text, for shapes on the same layer as the pin text, or for shapes on all layers. By default, the tool determines the pin tracing method based on the block type. To explicitly specify the pin tracing method, use the -trace_option option with the read_gds or read_oasis command or the -layer option with the trace_connectivity command. Table 4 describes the supported methods, the keywords used to enable them, and the default method for each block type.

Table 4 Pin Tracing Methods

Tracing method	Cell types for which this is the default tracing method	-trace_option Keyword	-layer Keyword
Create terminal shapes only for shapes that overlap pin text.	Modules, macro cells[1], analog cells, and black box cells	pins_only	pins_only

[1] *If a macro cell has rectilinear pins, you must explicitly set the -trace_option option to same_layer; the pins-only method always extracts rectangular pins.*

Table 4 Pin Tracing Methods (Continued)

Tracing method	Cell types for which this is the default tracing method	-trace_option Keyword	-layer Keyword
Trace the connectivity only on the same layer as the pin text.[2]	Library cells, pad cells, corner cells, pad spacer cells, cover cells, flip-chip pads, flip-chip drivers, physical-only cells, well tap cells, end cap cells, diode cells, filler cells, and metal fill	`same_layer`	`same`
Trace the connectivity across all layers.[2]	N/A	`all_layers`	`all`

Mapping Text Layers

If the text is not on the same layer as the metal shape, you must map the text layers to the metal layers. When you specify a text layer mapping, you can restrict connectivity tracing to only the mapped text layers by setting the `use_only_mapped_text` option.

You can specify the text layer mapping either in the physical rules file or by setting an application option.

- To specify the text layer mapping in the physical rules file, use the following syntax:

```
stream_in_instructions () {
   type : gds | oasis | both;
   text_layer_map(mapping_list);
   use_only_mapped_text : true | false;
}
```

- To specify the text layer mapping using application options, use the following syntax:

```
lm_shell> set_app_options \
   -name file.format.text_layer_map -value { mapping_list }
lm_shell> set_app_options \
   -name file.format.use_only_mapped_text -value true | false}
```

  where *format* is either `gds` or `oasis`.

In either method, each item in the *mapping_list* argument uses the following format:

```
"metal_layer {text_layer1[:data_type1] [text_layer2[:data_type2] ...]"
```

where

- *metal_layer* and *text_layer* are the layer names in the technology file
- *data_type*, which is optional, is the data type number in the technology file

[2] *To reduce runtime, limit the number of geometries traced by setting the `-trace_connectivity_limit` option or enable all-layer tracing only for specific pins by using the `-trace_all_layers_for` option.*

In the physical rules file, items in the *mapping_list* argument are separated by a comma. In the application option, they are separated by a space.

For example, to map the M1PIN text layer to the M1 metal layer and the M2PIN text layer to the M2 metal layer when reading GDSII files, either

- Include the following statements in the physical rules file:

```
stream_in_instructions () {
   type : gds;
   text_layer_map ("M1 M1PIN", "M2 M2PIN");
}
```

- Use the following command:

```
lm_shell> set_app_options -name file.gds.port_type_map \
   -value { "M1 M1PIN" "M2 M2PIN" }
```

See Also

- Using Physical Rules File

Mapping Exclude Layers

If the GDSII or OASIS file contains layers that are used as exclude layers, which define geometries to remove from metal layers during connectivity tracing, you must map the exclude layers to the metal layers.

You can specify the exclude layer mapping either in the physical rules file or by setting an application option.

- To specify the exclude layer mapping in the physical rules file, use the following syntax:

```
   stream_in_instructions () {
      type : gds | oasis | both;
      exclude_layers("mapping_list");
   }
```

- To specify the exclude layer mapping using application options, use the following syntax:

  ```
  lm_shell> set_app_options -name file.format.exclude_layers \
     -value {mapping_list}
  ```

  where *format* is either `gds` or `oasis`.

In either method, each item in the *mapping_list* argument uses the following format:

```
"metal_layer:purpose {exclude_layer[:exclude_purpose][:mask_type]]"
```

where

- *metal_layer* and *exclude_layer* are the layer names in the technology file
- *exclude_purpose*, which is optional, is the purpose number in the technology file
- *mask_type* is the mask constraint of the geometries

  Valid values are `mask_one`, `mask_two`, and `mask_three`.

In the physical rules file, items in the *mapping_list* argument are separated by a comma. In the application option, they are separated by a space.

For example, to subtract the mask_one geometries on the EM1 layer from the mask_one geometries on the M1 layer and the geometries on the EM2 layer from the geometries of the same mask type on the M2 layer when reading GDSII files, either

- Include the following statements in the physical rules file:

  ```
  stream_in_instructions () {
     type : gds;
     exclude_layers ("M1 EM1:mask_one", "M2 EM2");
  }
  ```

- Use the following command:

  ```
  lm_shell> set_app_options -name file.gds.exclude_layers \
     -value { "M1 EM1:mask_one" "M2 EM2" }
  ```

See Also

- Using Physical Rules File

Layer Mapping File Syntax

The syntax used to specify the layer mapping in the layer mapping file is

```
[object_type] tf_layer[:tf_purpose][:use_type]
[:mask_type] stream_layer[:stream_data_type]
```

where

- *object_type* specifies the types of objects to which the mapping applies

  Valid values are `data` (all non-text objects), `text`, and `all`. The default is `all`.
- *tf_layer* is the layer number in the technology file, which is a required argument
- *tf_purpose* is the purpose number in the technology file

  Valid values are either an integer value or the `drawing` keyword.
- *use_type* is the usage of the geometries in the design

  Few values are `power`, `ground`, `signal`, `clock`, `boundary`, `hard_placement_blockage`, `soft_placement_blockage`, `routing_blockage`, `area_fill`, `track`, `bridge_shape`, `keepout_region`, `instance_specific_mask`, `same_net_feedthrough_tsv`, `hi_voltage`, `lo_voltage`, `sealring_boundary`, `sealring_name`, `assembly_die_boundary`, and `stitching_zone`.

  For the complete list of valid values, see man pages.
- *mask_type* is the mask constraint of the geometries

  Valid values for metal layers are `mask_one`, `mask_two`, `mask_three`, and `mask_same`. Via layers have the following additional valid values: `MASK_FOUR`, `MASK_FIVE`, `MASK_SIX`, `MASK_SEVEN`, `MASK_EIGHT`, `MASK_NINE`, `MASK_TEN`, `MASK_ELEVEN`, `MASK_TWELVE`, `MASK_THIRTEEN`, `MASK_FOURTEEN`, and `MASK_FIFTEEN`.
- *stream_layer* is the layer number in the GDSII or OASIS file, which is a required argument
- *stream_data_type* is the data type in the GDSII or OASIS file

To include comment lines in the layer mapping file, start the line with a semicolon (;).

The layer mapping file also supports the following optional statements to further control how the `read_gds` or `read_oasis` command loads the library source files:

```
read_always true|false [-mapped_only] [-ignore_missing_layers]
blockage_as_zero_spacing true|false
cell_prop_attribute attribute_value
net_prop_attribute attribute_value
pin_prop_attribute attribute_value
```

Table 5 describes how these statements affect the behavior of the `read_gds` and `read_oasis` commands. You can also specify most of these settings on the command line. If you set an option both on the command line and in the layer mapping file, the setting in the layer mapping file takes precedence.

Table 5 Additional Layer Mapping File Statements

Mapping file statement	Command-line option	Description
`read_always true`	`-read_always true`	The command loads all layers defined in the library source file. If the library source file contains layers that are not defined in the technology file, the command creates new layers. This is the default behavior.
`read_always false`	`-read_always false`	The command issues an error message and fails if the library source file contains layers that are not defined in the technology file.
`read_always false -mapped_only`	`-read_always false -mapped_layers_only`	The command loads only those layers specified in the layer mapping file and ignores all other layers defined in the library source file.
`read_always false -ignore_missing_layers`	`-read_always false -ignore_missing_layers`	The command loads only those layers defined in the technology file and ignores all other layers defined in the library source file.
`blockage_as_zero_spacing false`	N/A	The command does not mark blockages read from the library source file as zero-spacing blockages. This is the default behavior.
`blockage_as_zero_spacing true`	N/A	The command marks blockages read from the library source file as zero-spacing blockages.
`cell_prop_attribute` *`attribute_value`*	`-cell_property` *`attribute_value`*	The command sets the cell instance names by using the name associated with the specified attribute value.
`net_prop_attribute` *`attribute_value`*	`-net_property` *`attribute_value`*	The command associates geometries with nets by using the specified attribute value.
`pin_prop_attribute` *`attribute_value`*	`-pin_property` *`attribute_value`*	The command associates geometries with pins by using the specified attribute value.

The following example shows a layer mapping file. The comments describe how the statements affect the behavior of the `read_gds` and `read_oasis` commands.

Example 2 Layer Mapping File

```
; Disable new layer creation
read_always false

; Map GDSII or OASIS layer 10 with data type 2 to power net shapes
; on technology file layer 56 with a purpose of 4
data 56:4:power 10:2

; Map GDSII or OASIS layers 31 and 32 with data type 0 to
; technology file layer 31 with a purpose of 0
data 31:0 31:0
data 31:0 32:0
```

Loading Physical Data from an Existing Library

To load the physical data from an existing physical library or cell library into the current library workspace, use the `read_ndm` command. If the library contains technology data, the command also loads this information. By default, the library manager loads all existing physical views (design, frame, and layout) of the cells into the library workspace. To load only specific views, use the `-views` option.

Note:

For the aggregate flow, the library manager ignores the `-views` option and always loads all available views, including the timing views.

For example, to load the physical data from the myphys.frame physical library into the current library workspace, use the following command:

```
lm_shell> read_ndm myphys.frame
```

When you load frame views into the library workspace, the library manager verifies that they are consistent with any existing frame views in the library workspace and issues a warning message if it finds inconsistencies.

Each time you run the command, you can specify only a single library; to load multiple libraries, you must run the command multiple times. You cannot use this command to load aggregate libraries.

You can specify the library path with an absolute path, a relative path, or no path. If you use a relative path or no path, the library manager uses the search path to find the library.

Loading Physical Data from a Design Library

If you are using the ETM flow, you load the physical data for the extracted timing models from the design library. To load the design library into the current library workspace, use the `read_ndm` command.

Note:

> You can load design libraries only when using the ETM flow; if you are using another flow, the tool issues an error.

By default, the library manager loads all existing physical views (design, frame, and layout) of the blocks with an empty label into the library workspace.

- To load only specific views, use the `-views` option.
- To load the blocks with a specific label, use the `-label` option.

For example, to load the physical data for blocks with the empty label from the myblock design library into the current library workspace, use the following command:

```
lm_shell> read_ndm myblock
```

To load the physical data for blocks with a label of *mylabel* from the myblock design library into the current library workspace, use the following command:

```
lm_shell> read_ndm -label mylabel myblock
```

Note:

> The generated library does not contain labels, regardless of the options used with the `read_ndm` command.

When you load frame views from a design library, the library manager verifies that the frame views in the design library are consistent with any existing frame views in the library workspace and issues a warning message if it finds inconsistencies.

Each time you run the command, you can specify only a single library; to load multiple libraries, you must run the command multiple times.

You can specify the library with an absolute path, a relative path, or no path. If you use a relative path or no path, the library manager uses the search path to find the library.

Importing Milkyway FRAM Views From the IC Compiler Tool

If you do not have physical library source files, but do have Milkyway FRAM views in an IC Compiler reference library, you can use these to create a physical library. You can then use this physical library to create a cell library.

To create a physical library by importing Milkyway FRAM views from the IC Compiler tool,

1. In the IC Compiler tool, export the Milkyway FRAM views of a reference library by using the `export_icc2_frame` command.

   For example, to export all FRAM views of all reference libraries of the my_lib design library, use the following commands:

   ```
   icc_shell> open_mw_lib my_lib
   icc_shell> export_icc2_frame
   ```

   The IC Compiler tool writes out a set of files into a new directory named icc2_frame. The generated files include the technology files and a tar file that contains the FRAM views converted to LEF format.

2. In the Library Manager tool, create a library workspace by using the `create_workspace -flow frame` command.

   For example, to create a library workspace using the technology data exported from the IC Compiler tool for the my_reflib reference library, use the following command:

   ```
   lm_shell> create_workspace -flow frame \
     -technology /path/icc2_frame/data/TF/my_reflib.tf my_workspace
   ```

3. Import the converted FRAM views generated by the IC Compiler tool into the library workspace by using the `import_icc_fram` command.

   For example, to import the converted FRAM views exported from the IC Compiler tool for the my_reflib reference library, use the following command:

   ```
   lm_shell> import_icc_fram \
      /path/icc2_frame/data/LEF/my_reflib.tar.gz
   ```

   Note:

   > You cannot mix imported Milkyway FRAM data with other physical data, such as LEF, GDSII, or OASIS data. If you use imported Milkyway FRAM data, all physical data in the library workspace must come from imported Milkyway FRAM data.

4. Validate the workspace, as described in Validating the Workspace.

5. Commit the workspace, as described in Committing the Workspace.

See Also

- Building Cell Libraries
- Loading Physical Data from an Existing Library

Specifying Which Blocks to Process

By default, when you load a library source file, the tool processes all of the cells defined in the file. If you need to process only a subset of the cells, you can

- Exclude cells from processing
  - Use the `lib.workspace.exclude_design_filters` application option to specify the excluded blocks.

  or

  - Load all of the blocks into the library workspace and then remove the unwanted blocks by using the `remove_lib_cell` command.
- Read only specific blocks when loading the source files

  Use the `lib.workspace.include_design_filters` application option to specify the blocks to read when loading the source files.

Handling Duplicate Physical Blocks

To avoid adding duplicate blocks when reading LEF, GDSII, and OASIS files, use the following options to specify how to handle these blocks. You can

- Ignore the duplicate block by using the `-merge_action ignore` option
- Replace the existing block with the new block by using the `-merge_action overwrite` option
- Merge the geometries of the new block with the existing block by using the `-merge_action update` option
- Add the block with a new name by using the `-merge_action add` option, which is the default
- Specify the handling for specific blocks in a block mapping file by using the `-block_map` *`file_name`* option

  The settings in the block mapping file override the `-merge_action` setting. For information about the block mapping file syntax, see Block Mapping File Syntax.

Block Mapping File Syntax

The syntax used to specify the block mapping in the block mapping file is

```
block_name[:rw_type[, flatten][:levels]] [block_type][-fixed_mask]
 [-site_name site_name] [-centerline_boundary]
```

Note:

> The `read_lef` command supports only the *block_name* and *rw_type* arguments; it ignores all other arguments.

The arguments are defined as follows:

- *block_name* is the name of the block, which is required

  You can use regular expressions to specify the block names; the `read_lef`, `read_gds`, and `read_oasis` commands use `glob`-style pattern matching to identify the blocks.

  When you use a block mapping file with the `read_gds` or `read_oasis` command, any blocks that do not match a pattern in the block mapping file are loaded into the library workspace as standard cells.

- *rw_type* specifies how to load the block into the library workspace

  This argument overrides the default behavior specified by the `-trace_option` option of the `read_gds` or `read_oasis` command for the specified blocks. Valid values for this argument are

  - `add`

    Creates a physical representation for the block; if a physical representation already exists for the block, it creates a physical representation with a new name.

  - `ignore`

    Does not create a physical representation for the block. The `read_gds` and `read_oasis` commands ignore this setting if the block is a subblock of a hierarchical block.

  - `overwrite`

    Creates a new physical representation that replaces the existing physical representation for the block.

  - `update`

    Updates the existing physical representation for the block by merging the new data with the existing data.

- `retain`

  Keeps the instance but does not create the reference library cell.

- `flatten` (supported only by the `read_gds` and `read_oasis` commands)

  Flattens all the instances of the block.

  You can specify `flatten` either as the *rw_type* argument or as a modifier for the *rw_type* argument. If you specify `flatten` as the *rw_type* argument, the tool treats this the same as if you specified `add, flatten`.

  When you specify this keyword, by default, the `read_gds` and `read_oasis` commands perform one level of flattening. To flatten more than one level of hierarchy, specify the *levels* argument.

- *levels* specifies the number of levels to flatten

  This argument is valid only when flattening is enabled.

- *block_type* is the block type; valid values are `3dic`, `abstract`, `analog`, `black_box`, `corner`, `cover`, `diode`, `end_cap`, `feedthrough`, `fill`, `filler`, `flip_chip_driver`, `flip_chip_pad`, `lib_cell`, `macro`, `module`, `pad`, `pad_spacer`, `tsv`, `vib`, and `well_tap`.

  If you do not specify this argument, the `read_gds` and `read_oasis` commands use a type of `lib_cell`.

  Note:

  > You can also specify the block type for a library cell by using the `cell_type` statement in the `cell` group in the physical rules file or by using the `set_attribute` command to set its `design_type` attribute

  For more information about setting block types, see Specifying the Block Types When Reading GDSII or OASIS Files.

- `-fixed_mask` sets the `is_mask_shiftable` attribute for the block to `false`

  This argument overrides the `-mask_fixed` option of the `read_gds` or `read_oasis` command for the specified blocks.

- `-site_name` sets the `site_name` attribute for the block to the specified value

  Each standard cell in a library source file must be associated with a site definition. Use this option to override the default site definition for a specific cell.

Note:

> You can also specify the site definition for a library cell by using the site statement in the cell group in the physical rules file or by using the set_attribute command to set its site_name attribute

For more information about setting site definitions, see Specifying the Site Definitions When Reading GDSII or OASIS Files.

- -centerline_boundary shrinks the default boundary to the center line of the pin geometries for the block

  This argument overrides the -centerline_boundary option of the read_gds or read_oasis command for the specified blocks.

  For more information about adjusting the block boundary, see Defining the Standard Cell Boundaries.

To include comment lines in the block mapping file, start the line with a semicolon (;).

The following example shows a block mapping file. The comments describe how the statements affect the behavior of the read_gds and read_oasis commands.

Example 3 Block Mapping File

```
; Load all blocks with CORE in their name as macro cells
*CORE*:add  macro

; Flatten all blocks with filler in their name
*filler*:flatten

; Flatten all blocks with MEM in their name two levels deep
*MEM*:flatten:2

; Ignore the rest of the blocks, they will be loaded only when they
; happen to be in a *CORE* block’s hierarchy
*:ignore
```

Using Physical Rules File

The physical rules file consolidates the placement of cell attribute and rule constraint into an unified file format. For example, when horizontal cell spacing rules are defined through a Tcl command and the vertical abutment rules are defined through an user-defined attribute and application option, the physical rules file consolidates these rules into a unified format.

To generate a physical library (*.frame* file) in the library preparation flow, the Library Manager tool uses the physical rules file. However, if you have an existing *.frame* file, you can read the physical rules file by using the edit flow. Ensure that the placement rule and physical cell or pin properties are read into *.frame* file.

The Figure 7 and Figure 8 show the recommended physical rules file preparation flows:

Figure 7 Building Cell Libraries From Scratch

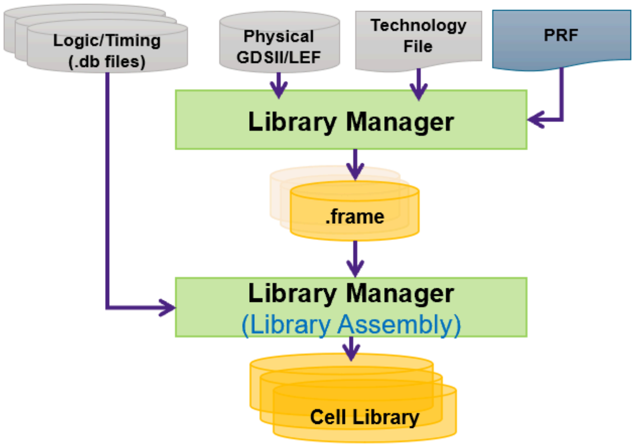


Figure 8 Updating Existing Cell Libraries

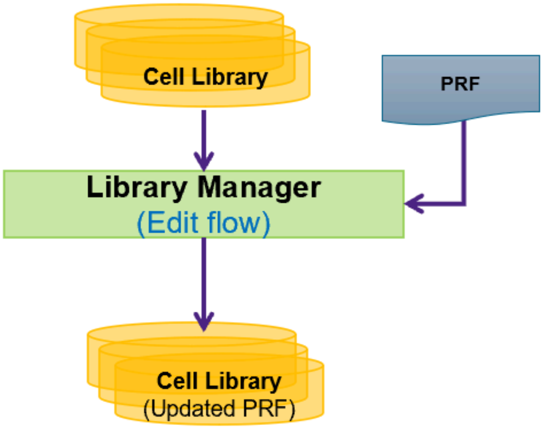
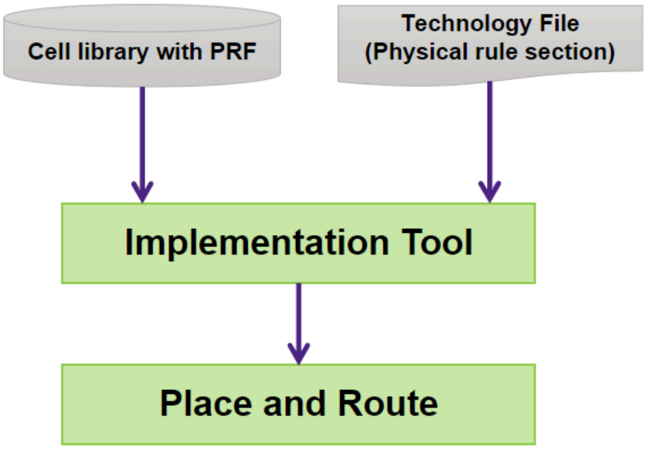


Figure 9 Using Physical Rules File in Design Libraries



The Figure 9 shows the usage of physical rules file in design libraries.

- The reference cell library with physical rules file includes placement rule and physical cell or pin properties.
- The legalizer uses the placement rule and physical cell or pin properties from *.CLIB* in the design library.

Note:

Legacy Tcl commands such as `set_placement_spacing_rule` are no longer required. For site and layer properties that are defined in the physical rule section in a technology file, the required definition for a design library is set during the `read_tech_file` command.

You can perform runtime overwrite under a design library by using the `read_physical_rule` command to read modified physical rules file.

To learn about the physical rules file, see the following topics:

- Physical Rules File Syntax
- base_layer_density_view Group Example
- Horizontal Cell Spacing Rule
- Vertical Abutment Rule
- Library Cell Attributes
- Routing Shape Geometries
- Library Pin Attributes

- Site Properties
- Pin Terminal Geometries
- Commands to Use Physical Rules File With Design Tools

Physical Rules File Syntax

Stream file loading instructions and physical rules for a cell library are stored in a file called the *physical rules file*. A physical rules file has the following syntax:

```
library (library_name) {
   distance_unit : 1um | 1mm;
   length_precision : 1 | 10 | 100 | 1000 | 10000;
   unit_site_width : float;

   base_layer_density_view () {
         saved_layers("layer1, layer2, ..., layerN");
         derived_layer(derived_layer_name) {
            and(layer1, layer2);
            or(layer1, layer2);
            xor(layer1, layer2);
            not(layer1, layer2;
            resize(layer1, float_list);
         }
   }

   stream_in_instructions () {
      type : gds | oasis | both;
      exclude_layers("mapping_list");
      port_type_map("mapping_list");
      text_layer_map("mapping_list");
      trace_copy_overlap_shape_from_sub_cell : true | false;
      trace_terminal_length : float;
      trace_terminal_type : pg | signal | all;
      trace_unmapped_text : true | false;
      use_only_mapped_text : true | false;
   }

   site(site_name) {
      width : float;
      height : float;
      type : core | pad;
      symmetry ("list_of_symmetries");
      is_default: true | false;
   }

   layer(layer_name) {
      routing_direction: unknown | horizontal | vertical |
                         diagonal_45 | diagonal_135)
      number_of_masks: 0 | 1 | 2 | 3 | 4;
   }
```

```
placement_rules(group_name) {
   forbidden_horizontal_site_spacing(label1, label2, min, max);
   forbidden_vertical_abutment_pairs(label1, "label1_1,
     label1_2, ...");
}

cell (cell_name) {
   allowable_orientation : ("list_of_orientations");
   cell_type : type;
   is_mask_shiftable : true | false;
   site : site_name;
   horizontal_spacing_labels ("left_label, right_label", ...);
   vertical_abutment_labels_top ("label_t1, label_t2, ...");
   vertical_abutment_labels_bottom ("label_b1, label_b2, ...");

   shape ( ) {
      shape_use : detail_route;
      layer_name : layer_name;
      rectangle("llx, lly", "urx, ury");
      polygon ("x1, y1", "x2, y2", …,  "xn, yn");
   }

   pin (pin_name) {
      direction : inout | input | internal | output;
      port_type : signal | tie_high | tie_low | clock | reset |
         analog_signal | scan | power | ground | analog_power |
         analog_ground | nwell | pwell | deep_nwell | deep_pwell;
      pg_type : primary | backup | internal;
      is_secondary_pg : true | false;
      connect_within_pin : none | via | via_wire;
      is_diode : true | false;
      is_em_via_ladder_required : true | false;
      must_join_group("layer_name1, layer_name2...", group_name);
      must_join_pin : pin_name;
      pattern_must_join : true | false;

      terminal ( ) {
        layer_name : layer_name;
        rectangle("llx, lly", "urx, ury");
        polygon ("x1, y1", "x2, y2", ...,  "xn, yn");
}

   }

   tap_rule(rule_name) {
        default_tap_distance : float;
        tap_distance_layer_rule (layer_name, float);
   }
   tap_boundary_wall_cell_properties(prop_name) {
   type : enum (p_tap:0,
   n_tap:1, p_tb_wall:2,
   n_tb_wall:3, p_fill_wall:4,
      n_fill_wall:5, p_tap_wall:6,
      n_tap_wall:7, p_tb_tap_wall:8,
      n_tb_tap_wall:9, left_boundary:10, right_boundary:11,
 p_tb_boundary:12,
      n_tb_boundary:13, p_tb_corner_boundary:14,
      n_tb_corner_boundary:15, p_inner_corner_boundary:16,
      n_inner_corner_boundary:17, p_left_tap:18,
      n_left_tap:19, p_right_tap:20,
      n_right_tap:21, p_tb_tap:22,
      n_tb_tap:23, p_tb_corner_tap:24,
      n_tb_corner_tap:25, left_p_gap:26, left_n_gap:27, right_p_gap:28,
 right_n_gap:29, tap_cell:30, fill_wall:31,
      p_tap_p_inner_corner_boundary:32,
      n_tap_p_inner_corner_boundary:33,
      p_tap_n_inner_corner_boundary:34,
      n_tap_n_inner_corner_boundary:35,
      p_fill_wall_replacement:36,
      n_fill_wall_replacement:37, fill_wall_replacement:38) ;

      tap_distance(string_array);
      min_wall_distance: float;
      max_wall_distance: float;
     }
      cell ( <cell_name> ) {
      physical_only_cell_class:
      power_type: normal / VPP / VBB ;
      vt_type: ELVT / LVT / LVTLL / SVT / ULVT / ULVTLL ;
      tap_boundary_wall_cell_properties: string;
      delta_tap_distance (float_array);
      }
     }
tap_cell_properties() {
         type : both | nwell| pwell;
         coverage_distance(left_value, right_value);
         min_row_edge_distance: float;
         coverage_pattern_left: left_full | left_nwell_only |
                                 left_pwell_only | left_none;
         coverage_pattern_right: right_full | right_nwell_only |
                                 right_pwell_only | right_none;
      }
   }

}
```

distance_unit

The distance_unit statement defines the linear distance unit. Valid values are 1um or 1mm. The value must correlate with the value specified for the unitLengthName attribute in the technology file.

Feedback

length_precision

The length_precision statement specifies the number of database units per distance unit, thereby defining the precision of distance measurements stored in the database. Valid values are 1, 10, 100, 1000, and 10000. For example, if you set distance_unit to 1um and length_precision to 1000, the database unit is 0.001 micron. Thus, all distance measurements are rounded to the nearest 0.001 micron. The value must match the value specified for the lengthPrecision attribute in the technology file.

unit_site_width

The unit_site_width statement specifies the placement site width. This value is used by the horizontal cell spacing rule. If you specify horizontal cell spacing rules but do not specify this attribute, the legalizer determines the placement site width.

base_layer_density_view

The base_layer_density_view group specifies the layer density syntax for a cell library including the layer geometry and area. You can have only one base_layer_density_view group. For information about querying layer geometry and area, see Querying Cell Area and Geometry of Cell Libraries.

The following table describes the statements available in the base_layer_density_view group. For an example of the base_layer_density_view group syntax, see base_layer_density_view Group Example.

Table 6 Statements in the base_layer_density_view Group

Statement	Description
saved_layers	Specifies the layers to process and save. These layers can only be the base layers defined in the technology file and the derived auxiliary layers specified by the derived_layer groups. Otherwise, the tool ignores the layers and issues a warning message. The saved_layers attribute is a required attribute that you can define only once in the base_layer_density_view group.
derived_layer	Specifies the auxiliary layers that are derived from the base layers defined in the technology file. An auxiliary layer is not defined in the technology file and is usually a combination of two layers. You can define multiple derived_layer groups, each with a different name. The derived_layer group includes the and, or, xor, not, or resize statements, which are mutually exclusive and one of them must be defined.

Table 6 Statements in the base_layer_density_view Group (Continued)

Statement	Description
`and`	Obtains the region covered by geometries on both *`layer1`* and *`layer2`*.
`or`	Obtains the region covered by geometries on either *`layer1`* or *`layer2`*.
`xor`	Obtains the region covered by geometries on *`layer1`* or *`layer2`* but not both.
`not`	Obtains the region covered by geometries on *`layer1`* but not *`layer2`*.
`resize`	Adjusts the edges of geometries on *`layer1`* and the operator. *`float_list`* is a list of one, two, or four floating-point numbers that specify the distance to expand the edges for resizing. • If one value is provided, the left, bottom, right, and top edges of the layer are each expanded by the value. • If two values are provided, the left and right edges are each expanded by the first value and the top and bottom edges are each expanded by the second value. • If four values are provided, the left, bottom, right, and top edges are expanded by the respective values.

`stream_in_instructions`

The `stream_in_instructions` group specifies how to process the information when loading GDSII or OASIS files. You can have at most two `stream_in_instructions` groups, one for GDSII (`type : gds`) and one for OASIS (`type : oasis`). The following table describes the statements available in the `stream_in_instructions` group.

Table 7 Statements in the stream_in_instructions Group

Statement	Description
`exclude_layers`	Specifies the rules for mapping between metal layers and the layers that are used to exclude geometries from the specified layers. For more information about this statement, see Mapping Exclude Layers.
`port_type_map`	Identifies the power and ground pins by mapping the pin name to its type. The mapping applies to all cells in the library source file. For more information about this statement, see Identifying Power and Ground Pins.

Table 7 Statements in the stream_in_instructions Group (Continued)

Statement	Description
`text_layer_map`	Maps the text layers to the metal layers so that the tool can associate metal shapes with a net or pin when the text is not on the same layer as the metal shapes. For more information about this statement, see Mapping Text Layers.
`trace_copy_overlap_shape_from_sub_cell`	Controls whether the commands duplicate a shape in a subblock to the top-level block when the shape is overlapped by terminals.
`trace_terminal_length`	Controls whether the commands split long pins on the block boundary into a terminal and a net shape. By default (0), the commands do not split long pins. When set to a positive integer, the commands split pins that are longer than the specified value. After splitting, the new terminal is the specified length; the rest of the pin is converted to a net shape.
`trace_terminal_type`	Specifies the types of long pins that are considered for splitting. By default, both PG and signal pins are considered.
`trace_unmapped_text`	Controls whether the commands trace all routing layers for text that is not mapped to any routing layer.
`use_only_mapped_text`	Controls whether the commands use only the text specified by the `text_layer_map` statement when tracing ports.

`site`

The `site` group defines a new site definition or modifies an existing site definition. A physical rules file can contain multiple `site` groups. Each group must have a unique name. If the site group name matches an existing site definition in the library, the tool updates the existing definition. Otherwise, it creates a new definition. To create a new definition, you must specify at least the `height` and `width` statements.

The following table describes the statements available in the `site` group.

Table 8 Statements in the site Group

Statement	Site definition attribute	Description
Name of the `site` group	`name`	Specifies the name of the site definition.

Table 8 Statements in the site Group (Continued)

Statement	Site definition attribute	Description
`width`	`width`	Specifies the width of the site definition. This statement is required when defining a new site definition.
`height`	`height`	Specifies the height of the site definition. This statement is required when defining a new site definition.
`type`	`type`	Specifies the type of the site definition. Valid values are `core` and `pad`. The default is `core`.
`symmetry`	`symmetry`	Specifies whether the site definition is symmetrical about the X-axis, Y-axis, or a 90-degree rotation. You can specify one or more of the following case-insensitive values: `x`, `y`, and `r90`. For example, `"x"` or `"y r90"`. By default there is no symmetry.
`is_default`	`is_default`	Specifies whether the site definition is the default site definition. There can be only one default site definition for a library. Valid values are `true` and `false`. The default is `false`.

`layer`

The `layer` group specifies layer-specific information. The following table describes the layer attributes you can specify in the physical rules file. The group's name must match an existing layer name in the technology data of the library.

Table 9 Statements to Set Layer Attributes

Statement	Layer attribute	Description
`routing_direction`	`routing_direction`	Specifies the routing direction for the layer. Valid values are `horizontal`, `vertical`, `diagonal_45`, and `diagonal_135`. The default is `unknown`.
`number_of_masks`	`number_of_masks`	Specifies the number of multipatterning masks supported on the layer. Valid values are 0, 1, 2, 3, or 4. The default is 0.

`placement_rules`

The `placement_rules` group specifies a placement rule for the library. A physical rules file can contain multiple `placement_rules` groups. Each group must have a unique name.

You can specify the following types of placement rules in the physical rules file:

- Horizontal cell spacing rule

  The `forbidden_horizontal_site_spacing` statement defines a horizontal spacing rule. You can specify multiple horizontal spacing rules. Each rule is identified by a unique label. The rules are assigned to specific cells by setting the `horizontal_spacing_labels` attribute in the cell group. For detailed information, see Horizontal Cell Spacing Rule.

- Vertical pin abutment rule

  The `forbidden_vertical_abutment_pairs` statement defines a vertical abutment rule. You can specify multiple vertical abutment rules. Each rule is identified by a unique label. The rules are assigned to specific cells by setting the `vertical_abutment_labels` attribute in the cell group. For detailed information, see Vertical Abutment Rule.

`cell`

The `cell` group specifies cell-specific information, including library cell attributes, cell-specific settings for placement rules, and library pin attributes.

The following topics describe specific attributes and geometries in the physical rules file syntax:

- Library Cell Attributes
- Routing Shape Geometries
- Library Pin Attributes
- Pin Terminal Geometries

`tap_rule`

The `tap_rule` statement defines a pre-defined group of tap rule. The `default_tap_distance` attribute specifies the default tap distance in microns.

The `tap_distance_layer_rule` attribute is a complex attribute which can be defined multiple times in a `tap_rule()` group. It specifies the name-value pair, where:

- The name is a marker referring to layer name
- The value is the tap distance under that marker

`tap_distance`

The `tap_distance` statement specifies the distance between two columns of tap cells in microns. The distance is measured from the middle of one tap cell to the middle of the other tap cell. Half-row definition for the left and right cells is supported.

`delta_tap_distance`

The `delta_tap_distance` statement specifies the shift of well pick-up (NTAP) or well break point (PTAP) from center of cell toward edge in microns.

base_layer_density_view Group Example

An example of the `base_layer_density_view` syntax in a physical rules file file is as follows:

```
library(my_lib) {
   ...
   base_layer_density_view() {
      saved_layers("poly, pdiff, ndiff, exposed_pdiff, exposed_ndiff");
         derived_layer("derived_layer1") {
            resize(poly, 0.005, 0.00);
         }
         derived_layer("exposed_pdiff") {
            not(pdiff, derived_layer1);
         }
         derived_layer("exposed_ndiff") {
            not(ndiff, derived_layer1);
         }
      }
   }
}
```

Horizontal Cell Spacing Rule

A horizontal cell spacing rule defines the valid spacing between standard cells.

Horizontal cell spacing rules are implemented by attaching labels, which are similar to attributes, to the left and right sides of library cells, assuming that the cell is in its north (R0) orientation, and specifying the invalid spacings between these labels.

To define horizontal cell spacing rules,

1. Add labels to the library cells that have spacing constraints by using the `horizontal_spacing_labels` statement in the affected cell groups. This statement has the following syntax:

   ```
   horizontal_spacing_labels (left_label, right_label, ...);
   ```

   You must specify the labels as left-right pairs. If you specify only one pair, the labels apply to all rows. Otherwise, you must specify the label pairs for all rows, starting with the bottom row in R0 orientation.

   If there is no horizontal spacing requirement on a side, specify `NONE` as the label. You can assign multiple labels to a library cell.

   For example, to assign a label named X to the right side of the cellA library cell and the left side of the cellB and cellC library cells and a label named Y to the right side of the cellC library cell, use the following statements:

   ```
   cell (cellA) {
      horizontal_spacing_labels (NONE, X);
   }
   cell (cellB) {
      horizontal_spacing_labels (X, NONE);
   }
   cell (cellC) {
      horizontal_spacing_labels (X, Y);
   }
   ```

2. Define the spacing requirements between the labels by using `forbidden_horizontal_site_spacing` statements in the `placement_rules` group. This statement has the following syntax:

   ```
   forbidden_horizontal_site_spacing(label1, label2, min, max);
   ```

   You must specify exactly two labels in each `forbidden_horizontal_site_spacing` statement. You can specify any of the labels defined by the `horizontal_spacing_labels` statements. The two labels can be the same or different.

   You must also specify the range of invalid spacings, in number of placement sites. The maximum value must be greater than or equal to the minimum value. By default, the spacing requirements are applied to cells in the same row. If you specify the `adjacent_row_only` keyword, the spacing requirements are applied to cells in adjacent rows.

Feedback

For example, the following statement specifies that there must be at least one placement site between labels X and Y (they cannot abut):

```
placement_rules (group1) {
   forbidden_horizontal_site_spacing(X, Y, 0, 0);
}
```

The following statement specifies that there must be at least one placement site between X labels and any boundary:

```
placement_rules (group1) {
   forbidden_horizontal_site_spacing(X, SNPS_BOUNDARY, 0, 0);
}
```

The following statement specifies that two X labels cannot have a spacing of two placement sites:

```
placement_rules (group1) {
   forbidden_horizontal_site_spacing(X, X, 2, 2);
}
```

The following statement specifies that labels X and Z must have a spacing of less than two placement sites or more than four placement sites:

```
placement_rules (group1) {
   forbidden_horizontal_site_spacing(X, Z, 2, 4);
}
```

See Also

- Defining Cell Spacing Constraints for Legalization

Vertical Abutment Rule

A vertical abutment rule defines illegal site alignments for vertically abutted standard cells to avoid DRC violations or pin access issues.

Vertical abutment rules are implemented by attaching label patterns to the top and bottom sides of library cells, assuming that the cell is in its north (R0) orientation, and specifying pairs of labels that cannot abut.

To define vertical pin abutment rules,

1. Define the illegal label pairs by using `forbidden_vertical_abutment_pairs` statements in the `placement_rules` group. Each statement specifies the labels that cannot abut a specific label. This statement has the following syntax:

```
forbidden_vertical_abutment_pairs(label,
   "illegal_label1,illegal_label2, ...");
```

For example, the following statement specifies that sites with label 1 cannot abut sites with label 1 or 2:

```
placement_rules (rule2) {
   forbidden_vertical_abutment_pairs (1, "1, 2");
}
```

The following statement specifies that sites with label 2 cannot abut sites with label 2:

```
placement_rules (rule2) {
   forbidden_vertical_abutment_pairs (2, "2");
}
```

2. Add label patterns to the library cells that have abutment constraints by using the `vertical_abutment_labels_top` and `vertical_abutment_labels_bottom` statements in the affected cell groups. These statements have the following syntax:

```
vertical_abutment_labels_top ("label_t1, label_t2, ...");
vertical_abutment_labels_bottom ("label_b1, label_b2, ...");
```

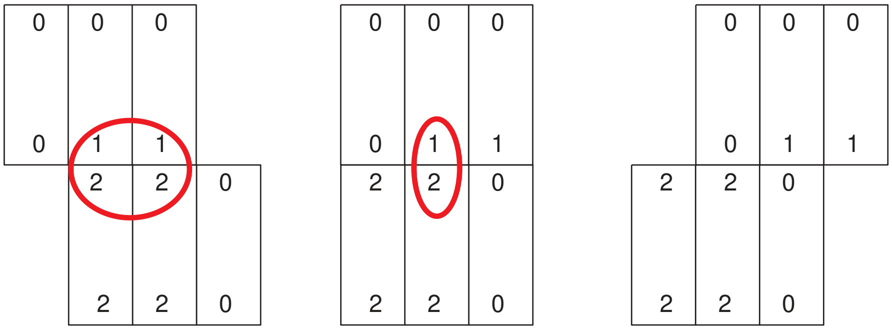
The number of labels in the pattern depends on the cell width in placement sites. For example, if a cell is three placement sites wide, the pattern includes three labels. You can specify multiple patterns for a cell by specifying the statements multiple times.

For example, to assign a pattern of 2 2 0 to the top and bottom of the cellA library cell, a pattern of 0 0 0 to the top of the cellB library cell, and a pattern of 0 1 1 to the bottom of the cellB library cell, use the following statements:

```
   placement_rules (rule2) {
      forbidden_vertical_abutment_pairs (1, "1, 2");
      forbidden_vertical_abutment_pairs (2, "2");
   }
...
   cell (cellA) {
      vertical_abutment_labels_top ("2, 2, 0");
      vertical_abutment_labels_bottom ("2, 2, 0");
   }
   cell (cellB) {
      vertical_abutment_labels_top ("0, 0, 0");
      vertical_abutment_labels_bottom ("0, 1, 1");
   }
```

Figure 10 shows some illegal and legal abutments between the cellA and cellB library cells based on this definition.

Figure 10   Vertical Abutment Rule



Illegal abutment   Illegal abutment   Legal abutment

Library Cell Attributes

The following table describes the statements available in the `cell` group for setting library cell attributes.

Table 10   Statements to Set Cell Attributes

Statement	Library cell attribute	Description
`allowable_orientation`	`allowable_orientation`	Specifies the allowable orientations for the library cell. Valid values are `R0`, `R90`, `R180`, `R270`, `MX`, `MXR90`, `MY`, and `MYR90`.
`cell_type`	`design_type`	Specifies the design type of the library cell. Valid values are `3dic`, `abstract`, `analog`, `black_box`, `corner`, `cover`, `diode`, `end_cap`, `feedthrough`, `fill`, `filler`, `flip_chip_driver`, `flip_chip_pad`, `lib_cell`, `macro`, `module`, `pad`, `pad_spacer`, `tsv`, `vib`, and `well_tap`.
`is_mask_shiftable`	`is_mask_shiftable`	Specifies whether the library cell is mask shiftable. Valid values are `true` and `false`.
`mask_shift_layers`	`mask_shift_layers`	Defines the library cell attribute of the mask shift layers.
`site`	`site_name`	Specifies the site definition associated with the library cell. The value must be one of the site definitions in the technology data for the library. To report the site definitions in the technology data, use the `report_site_defs` command.

Table 10 Statements to Set Cell Attributes (Continued)

Statement	Library cell attribute	Description
`cell_group`	`cell_group`	Defines the cell group of the library cell that is used by the variant-aware legalization.
`start_site_id`	`start_site_id`	Defines the start site ID of the library cell that is used by site ID legalization flow.
`share_timing_view`	`share_timing_view`	Specifies whether timing view of the library cell can be shared with another cell within the same cell group.
`routing_blockage`	`routing_blockage`	Specifies the routing blockages. Can be specified multiple times with same geometry. The number of routing blockages with the same layer name and geometry must be the same as the number of routing blockages in NDM. You cannot use routing blockage group in physical rules file to create a new routing blockage in NDM. The physical rules file uses layer name and geometry (rectangle or polygon) to match the routing blockage in frame or design view. Valid attributes are `mask_constraint`, `mask_type`, `net_types`, and `is_zero_spacing`. However, you can use the routing blockage group to update the property of the existing routing blockage.

Routing Shape Geometries

The following table describes the statements available in the `shape` group to define routing shapes. A `cell` group can contain multiple `shape` groups.

Note:

> The `write_physical_rules` command outputs the `shape` group syntax into a physical rules file only when the `shape_use` attribute is set to `detail_route`. See Writing the Physical Rules File.
>
> The `read_physical_rules` command ignores the `shape` group syntax.

Table 11 Statements in the shape Group

Statement	Shape attribute	Description
shape_use	shape_use	Specifies the usage of the shape. The only valid value is detail_route and specifies that the shape is used for detail routed signal nets.
layer_name	layer_name	Specifies the layer on which the shape exists.
rectangle		Specifies the bounding box of a rectangular shape. (*llx*, *lly*) are the lower-left coordinates and (*urx*, *ury*) are the upper-right coordinates. A shape group contains either a rectangle or a polygon statement.
polygon		Specifies the boundary of a rectilinear shape. (*x1*, *y1*), (*x2*, *y2*) , …, (*xn*, *yn*) are the polygon coordinates. A shape group contains either a rectangle or a polygon statement.

Library Pin Attributes

The following table describes the statements available in the pin groups of a cell group for setting library pin attributes.

Table 12 Statements to Set Cell Pin Attributes

Statement	Library pin attribute	Description
direction	direction	Specifies the direction of the library pin. Valid values are inout, input, internal, or output.
port_type	port_type	Specifies the type of the library pin. Valid values are signal, tie_high, tie_low, clock, reset, analog_signal, scan, power, ground, analog_power, analog_ground, nwell, pwell, deep_nwell, or deep_pwell.
pg_type	pg_type	Specifies the PG type of the library pin. Valid values are primary, backup, or internal.

Table 12 Statements to Set Cell Pin Attributes (Continued)

Statement	Library pin attribute	Description
`is_secondary_pg`	`is_secondary_pg`	Specifies whether the library pin is a secondary PG pin. Valid values are `true` and `false`.
`connect_within_pin`	`connect_within_pin`	Specifies the conditions under which a pin connection must be made within the pin shape. Valid values are `none`, `via`, and `via_wire`.
`is_diode`	`is_diode`	Specifies whether the library pin is a diode port. Valid values are `true` and `false`.
`is_em_via_ladder_required`	`is_em_via_ladder_required`	Specifies whether the library pin requires an electromigration via ladder candidate. Valid values are `true` and `false`.
`must_join_group`	`must_join_group` **Note:** This attribute is set on the terminals of the library pin, rather than on the library pin.	Specifies the must-join group to which the pin belongs and the layers on which this setting applies. To set this attribute on the terminals on all layers, use * as the layer name.
`must_join_pin`	`must_join_port`	Specifies the must-join port of the library pin.
`pattern_must_join`	`pattern_must_join`	Specifies whether the library pin requires a pattern-must-join connection. Valid values are `true` and `false`.

Example 4 Physical Pin Properties

```
library (test) {
cell (cell1) {
     pin(pin1) {
       // set must_join_group for terminals on layer m1 and v1.
        must_join_group (“m1, v1”, grp1);
     }
     pin(pin2) {
       // set must_join_group for terminals on all layers
        must_join_group (“*”, grp2);
     }
    pin(pin3) {
       // set must_join_group for terminals on m1 layer
        must_join_group (“m1”, grp3);
     }
}
```

Use the following syntax to specify the direction:

```
library ( * ) {
    cell (<cell_name>) {
 pin (<pin_name>) {
  direction : enum (inout | input | internal | output | tristate);
           port_type : enum (signal | tie_high | tie_low  | clock | reset
 | analog_signal | scan | power | ground | analog_power | analog_ground |
 nwell | pwell | deep_nwell | deep_pwell);
           pg_type : enum( primary | backup | internal);
           is_secondary_pg : true | false;
     }
   }
}
```

Specify the following attributes for the physical pin name:

- `direction`: Values are `inout | input | internal | output | tristate`.
- `port_type`: Values are `signal | tie_high | tie_low | clock | reset | analog_signal | scan | power | ground | analog_power | analog_ground | nwell | pwell | deep_nwell | deep_pwell`.
- `pg_type`: Values are `primary | backup | internal`.
- `is_secondary_pg`: Values are true or false.

Site Properties

You can define site properties in multiple groups with different names, but only one group can be a default group. The `read_PRF` command implements the same functionality as the `create_site_def` command when creating a site definition.

Syntax

```
library (<library_name>) {
     site(<name>) {
        width : float;
        height : float;
        type : enum (core | pad);
        symmetry  (list);
        is_default : true | false;
     }
}
```

Specify the following attributes for the site name:

- `width`: A float value in user unit.
- `height`: A float value in user unit. If you do not specify `width` or `height`, physical rules file updates the existing site name and issues a warning message.

- `type`: Values are `core | pad`. Core is the default.
- `symmetry`: Values are `x | y | r90`. The values are case-insensitive.
- `is_default`: Values are `true` or `false`, false is the default.

  If different default site properties exist, the tool displays an error message. If the same default sites exist, the tool updates the default site and issues a warning message.

Example 5 Site Properties

```
library(lib) {
     site(site_1) {
         width: 2;
         height: 3;
         type: pad;
         symmetry ("x  y r90");
         is_default: true;
     }
     site(site_2) {
         width: 2;
         height: 3;
         symmetry ("x");
     }
     site(site_3) {
         type: pad;
         symmetry ("y  r90");
     }
  }
```

Row Pattern

Row pattern is defined under library group. There can be multiple row patterns in a library, differentiated by the row pattern name.

```
library (<library_name>) {
    row_pattern# <row_pattern_name> # {
   site_list ("<site1>, <site2>, …, <siteN>");
   last_row_site_index: integer;
   first_row_orientation: enum (R0, or MX);
   overlapping_site_alignment(<overlappingSite>, "<site1>, <site2>",
 <orientation>);
        track_pattern (<layer_name >) {};    // see section 2.3
      }
}
```

Specify the following attributes for the row_pattern name:

- `site_list`: Specify at least two different sites for the row pattern. Otherwise, the tool issues a warning message and ignores the user specified group.
- `last_row_site_index`: Specify an integer with range [0, number of `site_list`-1]. -1 is the default.

- `first_row_orientation enum`: Specify one of the R0 or MX values.
  - `orientation`: The orientation that is set for the first site `(-site1)`.
  - `R0`: The orientation that is set for `site1, site2, site3, site4` are `R0, MX, R0, MX` respectively.
  - `MX`: The orientation that is set for `site1, site2, site3, site4` are `MX, R0, MX, R0` respectively.
- `overlapping_site_alignment`: Define overlapping site alignment multiple times with different overlapping sites.
  - `overlappingSite`: Composed of multi-height site composed with `site1 and site2,…,siteM'` with orientations. `site1 and site2,…,siteM'` must be one of the sites defined in the site list.
  - `orientation`: Specify one of the R0 or MX values.
- `track_pattern`: Define for different layers by the `layer_name` attribute.

Track Pattern

You can define track patterns only in the *site* group multiple times with different layer names. The layer name must be an existing layer; otherwise the tool ignores the site group and issues a warning message.

If you define a non-uniform track pattern in a site, mirroring occurs in the first site group and the tracks in the second site definition are the mirrors of tracks in the first site definition.

If you define a non-uniform track pattern in a row pattern, mirroring does not occur.

```
track_pattern (<layer_name>) {
      type: enum (uniform, or non_uniform);
      direction: enum (unknown, horizontal, or vertical);
      mask_pattern(list);
      spacing: float (> 0.0);
      offsets (list);
      widths (list);
      reserved_width_flags (list);
      grid_low_offsets (list);
      grid_high_offsets (list);
      grid_low_steps (list);
      grid_high_steps (list);
    }
site (<site_name>) {
    …
    track_pattern (<layer_name>) {}
    …
 }
 row_pattern (<row_pattern_name>)
```

```
        …
        track_pattern (<layer_name>) {}
        …
     }
```

Specify the following attributes for the track_pattern layer name:

- `type`: Specify the type of the track pattern. Valid values are `uniform | non_uniform`. `uniform` is the default.
  - `uniform`: The location of each track and the wire width of each track in the track pattern are the same.
  - `non_uniform`: The location of each track and the wire width of each track in the track pattern are different.
- `direction`: Specify the direction of each track pattern. Valid values are `horizontal | vertical`. The default is the preferred routing direction of the layer.
- `mask_pattern`: Valid values are `mask_one | mask_two | mask_three`.
- `spacing`: Specify the pitch distance between each track of the track pattern. You can specify this attribute only when creating a uniform track pattern. The default is the pitch of the layer.
- `offsets`: Specify the location of each track in the track pattern as follows:
  - Non-uniform track pattern: (Mandatory), you must specify the attribute. The offsets list size must be the same as the list size of mask pattern.
  - Uniform track pattern: (Optional), you can specify only one value if needed.
- `widths`: Specify the associated wire width for each track of the track pattern as follows:
  - The width value, when specified, must be greater than or equal to the minimum width of the layer.
  - For non-uniform track pattern, the list size must be same as the list size of the mask pattern.
  - For a uniform track pattern, the list size must be 1. If the width is not defined, the default is 0.

Example 6   Uniform Track Pattern

```
library (*) {
   site (unitTile) {
     track_pattern (M1) {
       type: uniform;
       direction: horizontal;
       mask_pattern ("mask_one", "mask_two", "mask_one", "mask_two");
       spacing: 0.2;
       offsets ("0.3");
       widths ("0.1");
       reserved_width_flags("true");
       grid_low_offsets ("0.1, 0.2, 0.3, 0.4");
       grid_high_offsets ("0.2, 0.3, 0.4, 0.5");
       grid_low_steps ("0.3, 0.4", "0.2, 0.4", "0.2, 0.4", "0.3, 0.4");
       grid_high_steps ("0.3, 0.4", "0.2, 0.4", "0.2, 0.4", "0.3, 0.4");
    }
 }
}
```

Example 7   Non-uniform Track Pattern

```
library (*) {
   site (unitTile) {
     track_pattern (M1) {
       type: non_uniform;
       direction: horizontal;
       mask_pattern ("mask_one", "mask_two", "mask_one", "mask_two");
       offsets ("0.3, 0.2, 0.1, 0.2");
       widths ("0.1, 0.2, 0.3, 0.4");
       reserved_width_flags ("true, false, true, false");
       grid_low_offsets ("0.1, 0.2, 0.3, 0.4");
       grid_high_offsets ("0.1, 0.2, 0.3, 0.4");
       grid_low_steps ("0.3, 0.4", "0.2, 0.4", "0.2, 0.4", "0.3, 0.4");
       grid_high_steps ("0.3, 0.4", "0.2, 0.4", "0.2, 0.4", "0.3, 0.4");
     }
  }
}
```

Consecutive Cell Placement Rule

The following syntax shows how to use the consecutive cell placement rule:

```
library ( <library_name> ) {
  placement_rules (rule_name) {
         # For existing CPODE check
         max_vertical_cell_edge_length : float ;
         max_vertical_cell_edge_exception_cells : string_array ;

         max_vertical_cell_edge_parallel_length : float ;
         max_vertical_cell_edge_parallel_length_site_spacing : int ;
         max_vertical_cell_edge_parallel_length_exception_cells :
 string_array ;

         max_horizontal_filler1 : int ;
  }
}
```

Specify the following attributes for the placement _rules name:

- max_vertical_cell_edge_length: Specifies the maximum length of consecutive vertical cell edge in microns for the existing CPODE check.
- max_vertical_cell_edge_exception_cells: Specifies the name of the cells that are the *except cells* for the existing CPODE check.
- max_vertical_cell_edge_parallel_length: Specifies the maximum parallel length of the consecutive vertical cell edge in microns for the edge check.
- max_vertical_cell_edge_parallel_length_site_spacing: Specifies the number of site gaps for N3 P5 edge check.
- max_vertical_cell_edge_parallel_length_exception_cells: Specifies the name of the cells that are the *except cells* for the edge check.
- max_horizontal_filler1: Specifies the maximum number of consecutive filler.

Example 8   Placement Rules

```
library ( prf ) {
    placement_rules ( ) {
      max_vertical_cell_edge_length : -50 ;
      max_vertical_cell_edge_exception_cells ("BOUNDARY", "TAP",
 "FILL1BWP7D5T16P96CPD") ;
      max_vertical_cell_edge_parallel_length : -30 ;  /* non-negative
 check, LBDB-1159 */
      max_vertical_cell_edge_parallel_length_site_spacing : 1 ;
      max_vertical_cell_edge_parallel_length_exception_cells ("BOUNDARY",
 "TAP") ;
      max_horizontal_filler1 : 3 ;
    }
}
```

Pin Terminal Geometries

The following table describes the statements available in the terminal group to define pin terminal geometries. A pin group can contain multiple termianl groups.

Note:

The read_physical_rules command ignores the terminal group syntax.

The write_physical_rules command can output the terminal group syntax into a physical rules file. See Writing the Physical Rules File.

Table 13 Statements to Set Terminal Attributes

Statement	Terminal attribute	Description
`layer_name`	`layer_name`	Specifies the layer on which the terminal exists.
`rectangle`		Specifies the bounding box of a rectangular terminal. (*llx*, *lly*) are the lower-left coordinates and (*urx*, *ury*) are the upper-right coordinates. A `terminal` group contains either a `rectangle` or a `polygon` statement.
`polygon`		Specifies the boundary of a rectilinear terminal. (*x1*, *y1*), (*x2*, *y2*) , …, (*xn*, *yn*) are the polygon coordinates. A `terminal` group contains either a `rectangle` or a `polygon` statement.

Commands to Use Physical Rules File With Design Tools

To know the commands used in physical rules file with design tools, see:

- Commands Used in icc2_shell or fc_shell
- Commands Used in icc2_lm_shell

Commands Used in icc2_shell or fc_shell

The commands usage in icc2_shell or fc_shell is as follows:

`read_physical_rules`:

- By default, the tool reads the physical rules file into the library specified by the library statement in the physical rules file or the current library if the specified library is not found.
- If you use the `-library` option to specify a target library name, the tool updates the runtime data of the reference library cell.
- For placement rules, the `read_physical_rules` command updates both design library attributes and library cell attributes.
- For physical properties, the `read_physical_rules` command updates the settable physical properties.

`remove_physical_rules`: The command does not remove physical properties because they may be defined using the `set_attribute` command.

`write_physical_rules`: Writes out all supported physical properties and stream-in application options, defined by physical rules file and Tcl commands.

Commands Used in icc2_lm_shell

The commands usage in icc2_lm_shell is as follows:

`read_physical_rules`

- For stream-in instructions, run the `read_physical_rules` command before the `read_gds` or `read_oasis` commands, so that the application options are set.
- For physical properties, the cell and pin attributes are automatically set during the `check_workspace` command, regardless of when you run the `read_physical_rules` command.

`remove_physical_rules`:

- For stream-in instructions, the `remove_physical_rules` command does not reset the application options because they may be defined using the `set_app_options` command.
- For physical properties, the cell and pin attributes are not removed because they may be defined using the `set_attribute` command.

`write_physical_rules`: Writes out all supported physical properties and stream-in application options, defined by physical rules file and Tcl commands.

Loading the Physical Rules File

A physical rules file specifies stream-in instructions, library-level rules, and cell- and pin-level rules and attributes. For details about the syntax of the physical rules file, see Using Physical Rules File.

You must load the data from the physical rules file before performing the step in which the data is used.

- The stream-in instructions are used when you read GDSII or OASIS files.
- The rules and attributes specified in the physical rules file are applied to the library when you run the `check_workspace` command.

To load the physical rules file,

1. Specify the location of the Library Compiler executable by setting the `file.lib.library_compiler_exec_script` application option.
2. Read the physical rules file by using the `read_physical_rules` command.

For example,

```
lm_shell> set_app_options \
   -name file.lib.library_compiler_exec_script \
   -value /LC/linux64/bin/lc_shell
lm_shell> read_physical_rules new.prf
```

By default, the `read_physical_rules` command reads the library-level rules and the cell- and pin-level rules and attributes. You should load this information after loading the physical data. To control the content loaded by the `read_physical_rules` command, use the `-include` option, which takes one or more of the following values:

- `all`

  Loads all information from the physical rules file.

- `cell`

  Loads only the cell- and pin-level rules and attributes from the physical rules file.

- `density`

  Loads the layer density related syntax into the cell library from the physical rules file. The generated base layer geometry and area information is loaded first in the design view and then in the frame view.

  For information about querying base layer geometry and area, see Querying Cell Area and Geometry of Cell Libraries.

- `instructions`

  Loads only the stream-in instructions from the physical rules file. You must load this information before reading GDSII or OASIS files.

- `layer`

  Loads only the layer attributes from the physical rules file.

- `library`

  Loads only the library-level rules from the physical rules file.

- `pin_attr`

Loads the direction, port_type, pg_type, and is_secondary_pg pin attributes. When you specify the pin_attr value, you must also specify the cell value; otherwise, the tool issues a warning message.

- placement_rule

  Loads only the placement rules from the physical rules file.

- site

  Loads only the site definitions from the physical rules file.

To write the library information from the library workspace into a physical rules file, use the write_physical_rules command. See Writing the Physical Rules File.

Writing the Physical Rules File

The write_physical_rules command writes the library, cell, and pin-level rules and attributes in the library workspace into a physical rules file. For details about the syntax of the physical rules file, see Using Physical Rules File.

To control the content written by the write_physical_rules command, use the -include option, which takes one or more of the following values:

- all

  Writes all information into the physical rules file.

- cell

  Writes only the cell- and pin-level rules and attributes into the physical rules file.

- density

  Writes the layer density related syntax from the cell library into the physical rules file.

- instructions

  Writes only the stream-in instructions into the physical rules file.

- layer

  Writes only the layer attributes into the physical rules file.

- library

  Writes only the library-level rules into the physical rules file.

- pin_attr

Writes the `direction`, `port_type`, `pg_type`, and `is_secondary_pg` pin attributes into the physical rules file.

- `placement_rule`

  Writes only the placement rules into the physical rules file.

- `routing_blockage`

  Writes only the routing blockage rules into the physical rules file.

- `shape`

  Writes only the routing shapes into the physical rules file.

- `site`

  Writes only the site definitions into the physical rules file.

- `terminal`

  Writes only the terminals into the physical rules file.

To read the information from a physical rules file into the library workspace, use the `read_physical_rules` command. See Loading the Physical Rules File.

Editing a Physical Rules File

To edit the library cell attributes, use the `read_physical_rules` command to update the library cell attributes using physical rules file in `icc2_lm_shell` as follows:

```
create_workspace -flow edit std.ndm
setenv SYNOPSYS_LC_ROOT /LC/linux64/lc/
read_physical_rules new.prf
check_workspace
commit_workspace -output new.ndm
```

To edit the design library attributes, use the `read_physical_rules` command to update the design library attributes using physical rules file in `icc2_shell` or `fc_shell` as follows:

```
open_lib TOP.ndm
open_block place_opt.design
setenv SYNOPSYS_LC_ROOT /LC/linux64/lc/
read_physical_rules new.prf
```

For library cell and pin attributes, the `read_physical_rules` command supports runtime override in the design library.

```
open_lib TOP.ndm
open_block place_opt.design
setenv SYNOPSYS_LC_ROOT /LC/linux64/lc/
```

```
read_physical_rules new.prf -include {cell}
```

To edit the library cell attributes, use the `update_physical_properties` command to update library cell attributes using physical rules file in `lc_shell` as follows:

```
open_physical_lib std.ndm
update_physical_properties -format prf -files new.prf
write_physical_lib -output new.ndm
```

Annotating Antenna Properties on Hard Macro Cells

To enable antenna checking on hard macro cells, use one of the following methods to annotate hierarchical antenna properties on their ports:

- Automatically extract the antenna properties by enabling IC Validator antenna extraction during the `check_workspace` command, as described in Automatic Antenna Property Extraction Using the IC Validator Tool.
- Manually annotate the antenna properties on the hard macro ports by using the `set_port_antenna_property` command, as described in Defining Antenna Properties on Hard Macro Cells.

Automatic Antenna Property Extraction Using the IC Validator Tool

To enable IC Validator antenna extraction during the `check_workspace` command, perform the following tasks before running the `check_workspace` command:

- Specify the location of the IC Validator executable by setting the `ICV_HOME_DIR` environment variable.

  You can set this variable in your .cshrc file. To specify the location of the IC Validator executable, use commands similar to those shown in the following example:

  ```
  % setenv ICV_HOME_DIR /root_dir/icv
  % set path = ($path $ICV_HOME_DIR/bin/AMD.64)
  ```

  Make sure that the version of the IC Validator executable that you specify is compatible with the Library Manager version that you are using.

  For more information about the IC Validator tool, see the IC Validator documentation, which is available on SolvNet.
- Set the `signoff.antenna.enabled` application option to `true`.
- Set `signoff.antenna` application options to control the extraction behavior.

  For a summary of these options, see Antenna Extraction Application Options.

Querying the Workspace

To see the contents of a library workspace, use the `report_workspace` command. By default, the command reports the following information:

- The workspace name
- The technology file loaded into the workspace
- The logic libraries loaded into the workspace

  You can optionally report the PVT and power rail information for each logic library by using the `-panes` option.
- The physical library source files loaded into the workspace
- The TLUPlus files loaded into the workspace
- The validation status of the workspace

For example,

```
lm_shell> report_workspace

Workspace name: mylib

...Checked? No

...Technology file: /usr/LIBRARIES/TECH/my.tf

...DB/Liberty files:
      1. /usr/LIBRARIES/DB/my_ff0p95v125c.db
      2. /usr/LIBRARIES/DB/my_ff0p95vn40c.db
      3. /usr/LIBRARIES/DB/my_ff1p16v125c.db
      4. /usr/LIBRARIES/DB/my_ff1p16vn40c.db

...Physical Source files:
      1. /usr/LIBRARIES/LEF/my.lef

...TLUP files: /usr/LIBRARIES/TLU/my.tlup
```

Validating the Workspace

To validate the contents of the library workspace, use the `check_workspace` command.

```
lm_shell> check_workspace
```

Note:

This step is not required when creating a technology-only library. No checks are performed when you run this command for a library workspace that contains only technology data.

In addition to validating the workspace, the `check_workspace` command performs the following tasks:

- Generates frame views for the physical cells in the workspace

  For information about generating frame views, see Generating Frame Views.

- Sets the `is_secondary_pg` attribute for PG pins

  For information about the secondary PG settings, see Identifying Secondary PG Pins.

- (Optional) Extracts the antenna properties

  For information about extracting antenna properties, see Automatic Antenna Property Extraction Using the IC Validator Tool.

When you run the `check_workspace` command, the tool performs the checks described in the following topics:

- Library Checks (includes checks for unique characterization points)
- Cell Checks (includes checks for missing cells, differences in logic function, and differences in the cell attributes)
- Pin Checks (includes checks for missing pins, and differences in the pin direction, type, and order)
- PG Rail Checks (includes checks for differences in the number, names, and order)
- Timing Arc Checks (includes checks for missing arcs, incorrect arcs, and differences in breakpoints)
- Leakage-Power Checks (includes checks for missing leakage-power information and differences in power conditions)

In many cases, the library manager can automatically fix the issues detected by these checks.

Note:

The checks performed by the `check_workspace` command validate the consistency of the library source files loaded into the workspace; however,

Feedback

the workspace might not contain all of the information required to perform placement and routing.

To generate a cell library from a library workspace, the workspace must pass these checks. However, in some cases you can generate a cell library using incomplete or inconsistent source libraries by defining a mismatch configuration, as described in Allowing Incomplete or Inconsistent Library Data. Cell libraries created with a nondefault mismatch configuration are not valid for implementation and can be used only for preroute feasibility analysis; these types of libraries are referred to as *prototype libraries*.

You can view the messages issued by the `check_workspace` command in the message browser, as described in Using the Message Browser Window.

To generate more detailed messages during the validation, use the `-details` option with the `check_workspace` command to specify one or more of the following categories: `arcs_progress`, `breakpoint_comparisons`, `combine_physical`, `leakage_progress`, `physical_only_cells`, or `all`.

Note:

Using the `-details` option can result in a very large log file. In general, you should run the `check_workspace` command with the `-details` option only to debug a specific issue.

Library Checks

The `check_workspace` command performs the following library checks:

- At least one physical library source file is loaded into the library workspace
- Each library logic loaded into the workspace has a unique characterization point (process, voltage, and temperature).

The workspace must pass these checks before it can be committed.

PG Rail Checks

This topic describes the PG rail checks for logic libraries that use the PG-pin syntax. For information about PG rail checks for older logic libraries that do not use the PG-pin syntax, see Using Non-PG Logic Libraries.

Note:

You cannot mix logic libraries with and without PG-pin syntax in a library workspace.

Logic libraries that use the PG-pin syntax define the PG pin connections by using library-level `voltage_map` attributes and cell-level `pg_pin` groups. The `voltage_map` attributes define the power rail names. The `pg_pin` groups define the PG pin connections for a cell; they use the `voltage_name` attribute to assign one of the defined power rails to each

PG pin. The signal pins of a cell are associated with the PG rails by using the pin-level `related_power_pin` and `related_ground_pin` attributes. Example 9 shows the use of these attributes to define the power connections in a .lib logic library file.

Example 9 Logic Library Example With voltage_map Attributes

```
library (my_lib) {
  ...
  voltage_map(VDD085, 0.85);
  voltage_map(VDD105, 1.05);
  voltage_map(VDD120, 1.20);
  ...
  cell(LS_085_105) {
    pg_pin (VDDH) {
      voltage_name : VDD105;
      pg_type : primary_power;
    }
    pg_pin (VDDL) {
      voltage_name : VDD085;
      pg_type : primary_power;
    }
    ...
    pin (in) {
      direction : input;
      related_power_pin : VDDL;
      related_ground_pin : VSS;
      ...
    }
    pin (out) {
      direction : output;
      related_power_pin : VDDH;
      related_ground_pin : VSS;
      ...
    }
    ...
  } /* end cell group*/
  ...
} /* end library group*/
```

The `check_workspace` command verifies that each logic library has the same number of rails and that the rails have the same names and types. However, the tool ignores rails that are defined in a `voltage_map` attribute but not used.

If the rail order differs, the tool automatically updates the order using the information from the base library.

By default, if a rail name differs between the logic libraries, the tool issues an LM-043 error. If the mismatch is caused only by rail names and positions, and not rail type differences, you can use the `rename_rail` command to make the rail names match. For example, assume logic library A has a `voltage_map` setting of `(VDD1.0, 1.0)` and logic library B has a voltage_map setting of (VDD1.1, 1.1). You could use the rename_rail command to align these rails, as shown in the following example:

```
lm_shell> rename_rail -library A -from VDD1.0 -to VDD
lm_shell> rename_rail -library B -from VDD1.1 -to VDD
```

In some cases, it might be difficult to find a combination of rename_rail commands that result in a consistent set of rail names. In these situations, you can have the tool assign the rail names instead of using the rail names from the logic libraries.

To enable the tool to assign the rail names, set the lib.logic_model.use_db_rail_names application option to false. In that case, the tool analyzes the voltage values for all of the power and ground pins on the library cells in all logic libraries and assigns a rail name to each unique combination. The generated rail names use the format *type_n*, where *type* is the rail type, such as power or ground, and *n* is an integer that is incremented to make each name unique. You can use the rename_rail command to change the generated rail names to names that are more meaningful to you.

Note:

> If the logic libraries contain duplicate rails, the number of rails generated by the tool will be less than the number of rails in the logic libraries. Because of this difference, the check_workspace command issues errors; you can ignore these errors, as they are followed by a message stating that rail mismatches have been fixed.

Cell Checks

To learn about the cell checks performed by the check_workspace commands, see the following topics:

- Missing Cell Checks
- Logic Function Check
- Cell Attribute Checks

Missing Cell Checks

The check_workspace command checks for the following types of missing cells:

- A cell that exists in some, but not all logic libraries, and you are using the normal flow

  The tool issues an error message if it finds missing or extra cells in a logic library, as compared to the set of cells in the base library.

  Note:

  > The cell name checks are case-sensitive. You can create a prototype library using case-insensitive checks by defining a mismatch configuration, as described in Allowing Incomplete or Inconsistent Library Data.

  To create a cell library, you must do one of the following:

  - Manually fix the source libraries and then rerun the check_workspace command
  - Rerun the check_workspace command with the -allow_missing option to enable the tool to add the missing cells to the generated cell library

    The generated cell library will contain all cells that exist in at least one logic library; if a cell does not exist in a logic library, it will have missing data in the pane associated with that logic library.

- A cell that exists in the logic libraries, but does not have physical data

  If a cell exists in the logic libraries, but does not have any physical data, the tool issues an error message and you cannot commit the workspace.

  To create a cell library, you must either

  - Set the lib.logic_model.auto_remove_timing_only_designs application option to true to automatically remove the cells from the workspace
  - Manually remove the cells from the workspace by using the remove_lib_cell command
  - Update the library source files

  To create a prototype library that includes placeholder cells for the missing physical cells, define a mismatch configuration, as described in Allowing Incomplete or Inconsistent Library Data.

- A physical cell that is not in the logic libraries

  If a physical cell is not in any logic library, the tool behavior depends on the flow you selected.

  - For the normal flow, the cell is not included in the generated cell library.
  - For the physical-only flow, the cell is included in the generated cell library.

  For detailed information about physical-only cells detected during workspace validation, specify the physical_only_cells keyword with the -details option.

Logic Function Check

The check_workspace command verifies that the logic function for a cell is equivalent in all of the logic libraries.

Cells with the same name in different logic libraries must have an equivalent logic function and the pin_equal and pin_opposite relationships must match; otherwise, the tool issues an error message. You must manually correct this error before you can continue with library preparation. You can correct this error either by updating the library source file and reloading it, or by using the remove_lib_cell command to remove the cell.

Cell Attribute Checks

The check_workspace command checks the dont_touch, dont_use, and use_for_size_only attributes for each cell in the logic libraries.

By default, if a cell with the same name in different logic libraries has different values for any of these attributes, the tool issues a warning message and stores the attribute value from the base library in the cell library. You can resolve conflicts by using the set_attribute command to change the attribute values.

To require a cell to have the same value for these attributes in all logic libraries, set the lib.logic_model.require_same_opt_attrs application option to true.

Pin Checks

To learn about the pin checks performed by the check_workspace commands, see the following topics:

- Missing Pin Checks
- Pin Definition Checks

Missing Pin Checks

The check_workspace command compares the pins defined for a cell in each logic and physical library source file against the pins defined for the cell in the base library. If a cell in the workspace is missing one or more pins as compared to the base library, the tool issues an error message.

Note:

> The pin name checks are case-sensitive. You can create a prototype library using case-insensitive checks by defining a mismatch configuration, as described in Allowing Incomplete or Inconsistent Library Data.

The tool automatically fixes the following issues:

- If a cell in the physical library source file has power or ground pins that are not defined in the logic libraries, the tool automatically adds these pins to the logic cell.
- If a pin is defined as a signal pin in the logic library, but is defined as a PG pin in the physical library source file, the tool issues a warning, redefines the pin as a PG pin, and removes all timing information from the pin.

To create a cell library, you must manually fix any other types of errors in the source libraries. If the pins are not actually missing, but the pin names differ between libraries, you can fix this issue by using the `set_attribute` command to update the `name` attribute for the pins.

To create a prototype library that includes placeholder pins for the missing pins, define a mismatch configuration, as described in Allowing Incomplete or Inconsistent Library Data.

Pin Definition Checks

The `check_workspace` command checks the following aspects of the pin definition:

- Pin direction

  If a pin on any cell in the workspace has a different direction as compared to the base library, the tool issues an error message, with the following exceptions:

  - The pin is defined as a bidirectional power or ground (PG) pin in the physical library source file and is defined as an input or output PG pin in the base library. In this case, the tool uses the direction defined in the base library.
  - The pin does not have a direction in the physical library source file, as is the case when you read GDSII or OASIS files. In this case, the tool uses the pin direction from the logic libraries.

  To create a cell library, you must manually fix the source libraries.

  To create a prototype library that uses the pin direction from the base library in the case of mismatches, define a mismatch configuration, as described in Allowing Incomplete or Inconsistent Library Data.

- Pin type

  The tool automatically fixes the following issues:

  - If a pin does not have a pin type in the physical library source file, as is the case when you read GDSII or OASIS files, the tool uses the pin type from the logic libraries.
  - If a pin is defined as a signal pin in the logic library, but is defined as a PG pin in the physical library source file, the tool issues a warning, redefines the pin as a PG pin, and removes all timing information from the pin.

  To generate a cell library, you must manually fix the following issues:

  - A pin is defined as a signal pin in all logic libraries, but the `bit_width`, `is_pad`, `is_diode`, or `is_scan` attribute setting differs

    To fix this issue, you can use the `set_attribute` command to change the attribute values.
  - A pin is defined as a signal pin in one logic library, but is defined as a PG pin in another logic library

    If a pin is incorrectly defined as a signal pin and is not used in the logic function definition, you can fix this issue by using the `set_attribute` command to change the `port_type` attribute of the pin.

    Note:

    > Changing a pin from a signal pin to a PG pin might cause the tool to remove timing arcs for that pin.
    >
    > You can use the `set_attribute` command only to change a signal pin to a PG pin; you cannot change a PG pin to a signal pin.
  - A pin is defined as a PG pin in all logic libraries, but the `pg_type` attribute differs

    To fix this issue, you can use the `set_attribute` command to change the `pg_type` attribute of the pin.

  To create a prototype library that uses the pin type from the base library in the case of mismatches, define a mismatch configuration, as described in Allowing Incomplete or Inconsistent Library Data.

  Note:

  > The mismatch configuration can only change a signal pin to a PG pin; it cannot change a PG pin to a signal pin.

- PG pin connection

  If the rail name specified for the voltage_name attribute in a pg_pin group is not defined in a library-level voltage_map attribute or the voltage_name attribute for a PG pin differs between libraries, the tool issues an error message. To fix this error, you can use the set_attribute command to change the rail_name attribute. To see the rail names defined for a library workspace, use the report_workspace -panes command.

- Related PG pin

  Signal pins have a related power pin and a related ground pin; power pins can have a related bias pin. Typically, these relationships are defined in the logic libraries.

  If the logic libraries do not define these relationships, either because they do not define the PG pins (and the PG pins are inherited from the physical library source file) or they define the PG pins, but not the relationships, the check_workspace command tries to derive these relationships as follows:

  - If a cell has a single power pin and a single ground pin, the tool automatically derives the related PG pin information.

  - If a cell has multiple power pins, you must guide the tool to derive the related PG pin information.

    To guide the tool, use the set_attribute command to set the related_power_pin_hint and related_ground_pin_hint attributes on the pins with the missing information.

    ```
    lm_shell> set_attribute [get_lib_pins lib/cell/pin] \
       related_power_pin_hint PGpin_name
    ```

  If the related PG pin information for a pin exists but is incorrect, you can correct it by using the set_attribute command to set the related_power_pin_name, related_ground_pin_name, or related_bias_pin_name attribute. For example,

  ```
  lm_shell> set_attribute [get_lib_pins lib/cell/pin] \
     related_power_pin_name PGpin_name
  ```

  Note:

  By default, the check_workspace command ignores missing power and ground relationships during the workspace validation, which might affect analysis performed by the implementation tool. To require signal pins to have at least one related power or ground pin, set the lib.workspace.allow_missing_related_pg_pins application option to false before running the check_workspace command.

- Pin order

  For each cell, the tool determines the pin order in each logic and physical library source file that contains the cell, and verifies that it is the same in each library.

  In a logic library, the tool determines the pin order from the logic function, followed by other non-signal pins, such as internal pins, power pins, ground pins, p-well pins, and n-well pins. In a physical library source file, the pin order is determined by the order in which they are specified.

  Note:

  > Although the tool can resolve many pin ordering issues, these issues might be caused by out-of-date libraries. If the tool reports pin ordering issues, you should verify that you are using the correct set of libraries before continuing.

  If the pin order differs between a logic library and the base library, the result depends on whether the mismatched pins contribute to the logic function.

  - If so, the tool issues an error message and does not load the affected cell. You must manually correct this problem to continue with library preparation.

  - If not, the tool reorders the pins to match the order in the base library.

  If the pin order differs between a physical library source file and the base library, the tool automatically updates the pin order using the information from the base library.

Timing Arc Checks

For each cell with the same name in different logic libraries, the `check_workspace` command verifies that the cell has the same timing arcs as those defined in the base library, including the timing arcs that specify timing constraints, such as setup and hold.

In the following cases, the tool can often resolve the mismatches:

- The libraries use different representations for the same multimode timing arcs.

  A multimode timing arc can be represented as a single arc with multiple modes or as multiple arcs with a single mode. The tool tries to resolve mismatches of multimode arcs by considering both of these representations.

- The timing arcs use different breakpoints (indexes into the delay lookup table) in the various logic libraries.

- The breakpoints for complementary attributes (`cell_rise` and `cell_fall`, `rise_transition` and `fall_transition`, `rise_setup` and `fall_setup`, or `rise_hold` and `fall_hold`) vary within a library or among libraries.

  The breakpoints for complimentary attributes must be functions of the same variables, such as `input_net_transition` (T_{inp}) and `total_output_net_capacitance` (C_{out}).

However, if one of the attributes is defined as a function of both variables, but the other is defined as a function of only one variable, the tool can resolve this difference.

The tool can also resolve differences in the complementary attributes for setup or hold when one attribute is defined as a function of `constrained_pin_transition` (T_{dat}) and the other is defined as a function of `related_pin_transition` (T_{clk}).

- A timing arc is represented as a table in one library and as a scalar in another.
- A library is missing `rise_transition` or `fall_transition` attributes.

  The tool resolves this by setting the unspecified transition time equal to the specified one for the arcs that are missing these attributes.

If the tool cannot resolve the mismatches, it issues the following message in addition to messages providing information about the timing arc mismatch.

```
Error: Could not fix all missing arcs (LM-025)
```

In the messages, the tool represents a timing arc using the following format:

```
cell from_pin to_pin [related_pin] sense [object_name] [{w:when_cond}]
 [MIN] [{m:modes}] [DUPn]
```

The cell, from pin, to pin, and arc sense are always specified. The other fields are reported only when relevant. The related pin is reported only when the delay table is dependent on a related capacitance. If the library specifies a name for the timing group, that name is reported in the *object_name* field. The *when_cond* field is reported for state-dependent timing arcs and conditional timing checks. The `MIN` keyword indicates a minimum delay arc and the `DUP` keyword indicates a duplicate arc.

For example, a negative-unate timing arc from pin A to pin Z on the NAND3 cell is represented as “NAND3 A Z negative_unate”.

To pass workspace validation, you must either fix the errors, provide guidance to the tool about how to handle them, or remove the cell with mismatched timing arcs from all logic libraries.

If the tool finds missing or extra timing arcs for a cell in a logic library, as compared to those in the base library, it issues a warning.

To add arcs that are not defined in the base library, but are defined in another logic library, rerun the `check_workspace` command with the `-allow_missing` option. The arcs are added only if the nodes in the timing graph are the same as in the base library.

Note:

If you choose to override these errors with the `-allow_missing` option, the cell library will have missing data for the cell in some panes.

To remove cells with mismatched timing arcs from the logic libraries, use the `remove_lib_cell` command. You must remove the cell from all logic libraries; otherwise, this results in a missing cell error, which is described in Missing Cell Checks.

For detailed messages during the timing arc checks, specify the `arcs_progress` keyword with the `-details` option. For detailed messages during the breakpoint comparisons, specify the `breakpoint_comparisons` keyword with the `-details` option.

Note:

> Using the `-details` option can result in a very large log file. In general, you should run the `check_workspace` command with the `-details` option only to debug a specific issue.

Leakage-Power Checks

For each cell with the same name in different logic libraries, the `check_workspace` command verifies that it has the same leakage-power information, as defined in the `cell_leakage_power` attribute and the `leakage_power` group.

If the tool finds missing or extra leakage-power information for a cell in a logic library, as compared to those in the base library, it issues a warning. To get detailed information about the missing leakage-power information, use the `-details leakage_progress` option when you run the `check_workspace` command. In the messages, the tool represents leakage-power information using the following format:

```
cell [related_power_pin] [{w:when_condition}]
```

To add leakage-power information that is not defined in the base library, but is defined in another logic library, rerun the `check_workspace` command with the `-allow_missing` option.

Note:

> If you choose to override this error with the `-allow_missing` option, the cell library will have missing data for the cell in some panes.

If the `when` statement in the `leakage_power` group is not identical between logic libraries, the tool checks the logic equivalence of the conditions.

If the conditions are logically equivalent, the tool uses the condition from the base library. If the conditions are not logically equivalent, the tool issues an error message and you cannot commit the workspace.

Identifying Secondary PG Pins

The router uses the is_secondary_pg attribute to identify the PG and bias pins on which to perform secondary PG pin routing. The check_workspace command sets this attribute on the pins based on certain pin and block attribute settings.

- For PG pins, the setting is based on the pg_type and std_cell_main_rail attributes of the PG pin, which are defined in the logic library source files, and the design_type attribute of the block, which is derived from the physical library source files. Table 14 shows the is_secondary_pg settings for the various combinations of these attributes.
- For bias pins, the setting is based on the pg_type and physical_connection attributes of the bias pin, which are defined in the logic library source files. Table 15 shows the is_secondary_pg settings for the various combinations of these attributes.

Table 14 Determining the is_secondary_pg Setting for PG Pins

Logic library PG pin attribute		**Cell library cell attribute**	**Cell library pin attribute**
pg_type	**std_cell_main_rail**	**design_type**	**is_secondary_pg**
primary_power or primary_ground	true	all	false
primary_power or primary_ground	false or unset	all	true
backup_power or backup_ground	any value	all	true

Table 15 Determining the is_secondary_pg Setting for Bias Pins

Logic library PG pin attribute		**Cell library pin attribute**
pg_type	**physical_connection**	**is_secondary_pg**
pwell, nwell, deeppwell, or deepnwell	routing_pin	true
pwell, nwell, deeppwell, or deepnwell	device_layer	false

Committing the Workspace

After the workspace passes validation with the check_workspace command, you can commit the workspace, which saves the workspace on-disk as a cell library. To commit the workspace, use the `commit_workspace` command. By default, the tool saves the cell library in a directory named *workspace_name*.ndm in the current directory using the current schema revision.

```
lm_shell> commit_workspace
```

To override the default behavior, use the following options:

- `-output` *`dir_name`*

  This option saves the cell library in the specified directory. For example, to save the cell library in a directory named mylib.ndm, use the following command:

  ```
  lm_shell> commit_workspace -output mylib.ndm
  ```

  The `-output` option specifies the name of both the library and its root directory. If you use this option, both the library and its root directory are named *dir_name*, including any specified extension. If you do not use this option, the library name is *workspace_name*, while its root directory is named *workspace_name*.ndm. The following table shows the library and root directory names for the cell library generated from the mylib workspace with and without the `-output` option:

	Library name	**Root directory name**
Without `-output` option	mylib	mylib.ndm
With `-output mylib.ndm` option	mylib.ndm	mylib.ndm

- `-version` *`version_string`*

  This option saves the cell library using the schema revision for an older tool version. To see the available version strings, use the `report_versions` command.

By default, the command fails if the specified cell library already exists. To overwrite an existing cell library, use the `-force` option. For example,

```
lm_shell> commit_workspace -output mylib.ndm
Error: File 'mylib.ndm' already exists (FILE-009)
lm_shell> commit_workspace -force -output mylib.ndm
```

To enable all `commit_workspace` commands to overwrite an existing cell library without using the `-force` option, set the `lib.workspace.allow_commit_workspace_overwrite` application option to `true`.

```
lm_shell> set_app_options \
   -name lib.workspace.allow_commit_workspace_overwrite -value true
```

The views saved in the generated cell library depend on the library preparation flow you enabled and the settings of the related application options.

- For the normal and ETM library preparation flows, the timing and frame views are saved
- For the frame-only and physical-only library preparation flows, only the frame views are saved
- To save the design views, set the `lib.workspace.save_design_views` application option to `true`.
- To save the layout views, set the `lib.workspace.save_layout_views` application option to `true`.

You can have only a single workspace in memory at one time. After you commit a workspace, it is removed from memory so that you can prepare another cell library.

See Also

- Library Preparation Flows

Generating Frame Views

If a library workspace passes validation, the `check_workspace` command creates frame views for the blocks in the workspace. You do not need to run a separate command to create the frame views.

The Library Manager tool uses the `lib.physical_model` application options to control the frame view generation, as described in the following topics:

- Generating Blockages During Frame View Generation
- Creating Via Regions
- Deriving Must-Join Pins
- Creating Frame Views for Hierarchical Cells
- Creating Colored Frame Views
- Mode-Based Configuration of Frame View Generation Options

Note:

When you create frame views with the `create_frame` command in an implementation tool, you control the frame view generation with command options, which have the same names as the lm_shell application options. For example, the `lib.physical_model.block_all` application option in lm_shell

Feedback

and the `create_frame -block_all` command option in an implementation tool have the same behavior.

You must set these application options before running the `check_workspace` command. When you commit the workspace, the library manager saves these application option settings with the cell library.

To update frame views for an existing cell library, such as when the technology file or tool version is updated, use the process described in Modifying a Cell Library.

See Also

- Building Cell Libraries
- Frame View Generation Application Options

Generating Blockages During Frame View Generation

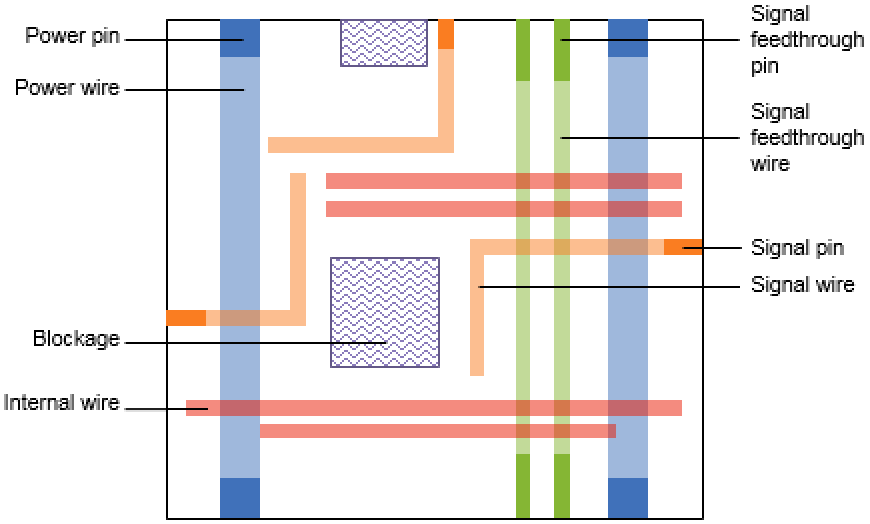
A blockage prevents routing or vias within the blockage region. The implementation tools support the blockage types shown in the following table:

Table 16 Blockage Types

Blockage type	Attribute setting	Description
Regular blockage	`is_zero_spacing=false`	The router treats a regular blockage as real metal and applies all design rules to the blockage.
Zero-spacing blockage	`is_zero_spacing=true`	The router checks zero-spacing blockages only for overlaps, and does not apply design rules to them.
Design rule via blockage	`is_design_rule_blockage=true`	Design rule via blockages are similar to zero-spacing via blockages, but they honor the rules defined for the via blockage layer in the `DesignRule` sections of the technology file.

Figure 11 shows an example design view for a standard cell. This example is used to illustrate the various controls over blockage generation.

Figure 11 Example Design View for a Standard Cell

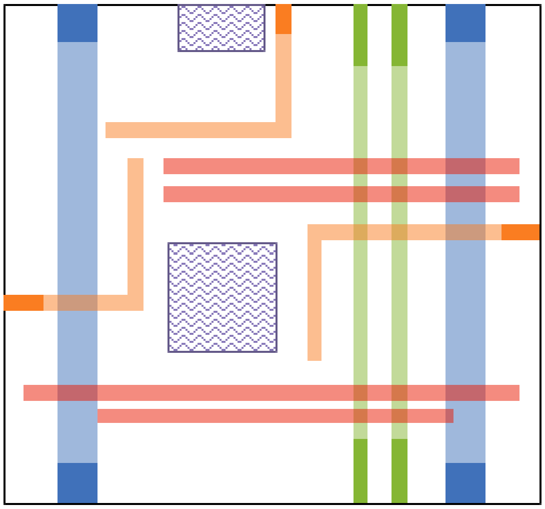


By default, when the library manager generates a frame view for a standard cell, it preserves the blockages that exist within the cell and does not generate additional blockages. Figure 12 shows the resulting frame view for the design view shown in Figure 11, which includes all metal shapes from the design view.

Note:

> By default, when you create a frame view for a macro by using the `create_frame` command in the implementation tool, the tool trims the metal shapes and regular blockages that touch a pin, as shown inFigure 12 .

Figure 12 Default Frame View for a Standard Cell



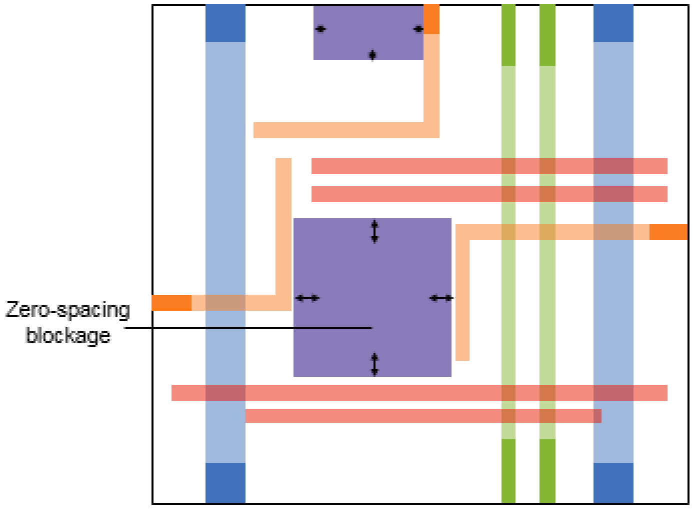
Converting Regular Blockages to Zero-Spacing Blockages

In some cases, particularly when a blockage touches a pin, regular blockages can cause pin access issues. To improve pin access, you can convert regular blockages to zero-spacing blockages during frame view generation. To enable the conversion, set the `lib.physical_model.convert_metal_blockage_to_zero_spacing` application option. When the tool converts the regular blockage to a zero-spacing blockage, it enlarges the blockage boundary by a specified amount. You can specify different conversion parameters for each layer depending on whether the blockage touches a pin. The syntax to set this application option is

```
set_app_options
   -name lib.physical_model.convert_metal_blockage_to_zero_spacing
   -value { {layer1 spacing1 [all|touch_pin|no_touch_pin]}
            {layer2 spacing2 [all|touch_pin|no_touch_pin]} ...
            {layern spacingn [all|touch_pin|no_touch_pin]} }
```

Figure 13 shows the frame view, where the purple region represents the zero-spacing blockage that replaced the regular blockage. The arrows represent the increased size of the blockage.

Figure 13 Converting Regular Blockages to Zero-Spacing Blockages



For via blockages, you can convert zero-spacing blockages to design rule blockages on specific layers by setting the `lib.physical_model.design_rule_via_blockage_layers` application option.

Trimming Blockages Around Pins

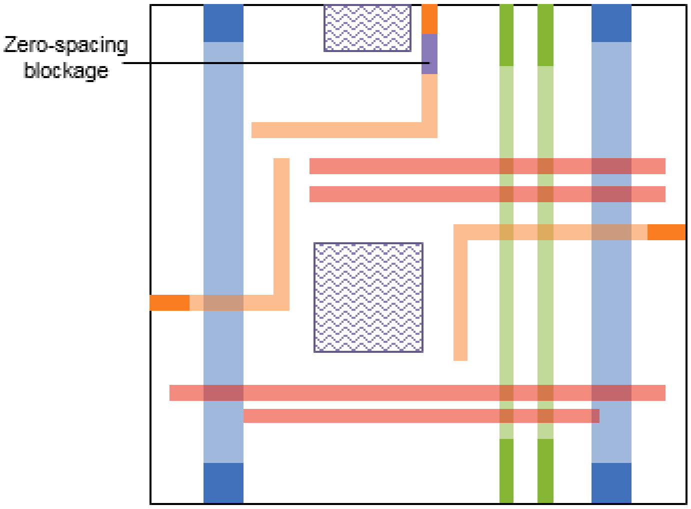
You can improve pin access by converting portions of metal shapes and regular blockages near the pins to zero-spacing blockages. This process is called *trimming*. The trimming distance is the largest fat table spacing defined for the layer in the technology file.

To trim the metal shapes and regular blockages that touch a pin for all layers, set the `lib.physical_model.preserve_metal_blockage` application to `false`. Figure 14 shows the resulting zero-spacing blockage.

Note:

> When you create a frame view for a macro by using the `create_frame` command in the implementation tool, this is the default behavior.

Figure 14 Trimming Touching Blockages Around Pins

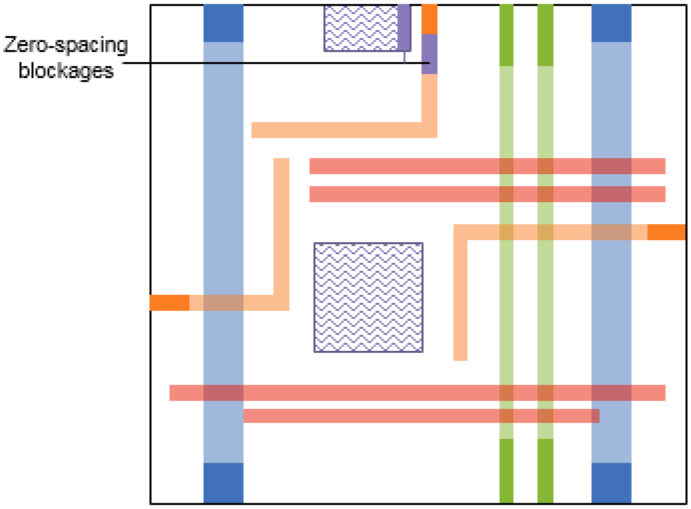


To trim the metal shapes and regular blockages for specific layers, set the `lib.physical_model.trim_metal_blockage_around_pin` application option. When you set this option, you can choose to trim only those metal shapes and regular blockages that touch the pin (`touch`) or all metal shapes and regular blockages within the minimum spacing of a pin (`all`). The syntax to set this application option is

```
set_app_options
   -name lib.physical_model.trim_metal_blockage_around_pin
   -value { {layer1 all|touch|none}
            {layer2 all|touch|none} ...
            {layern all|touch|none} }
```

Figure 15 shows the resulting zero-spacing blockages when all metal shapes and regular blockages in the vicinity of the pin are trimmed.

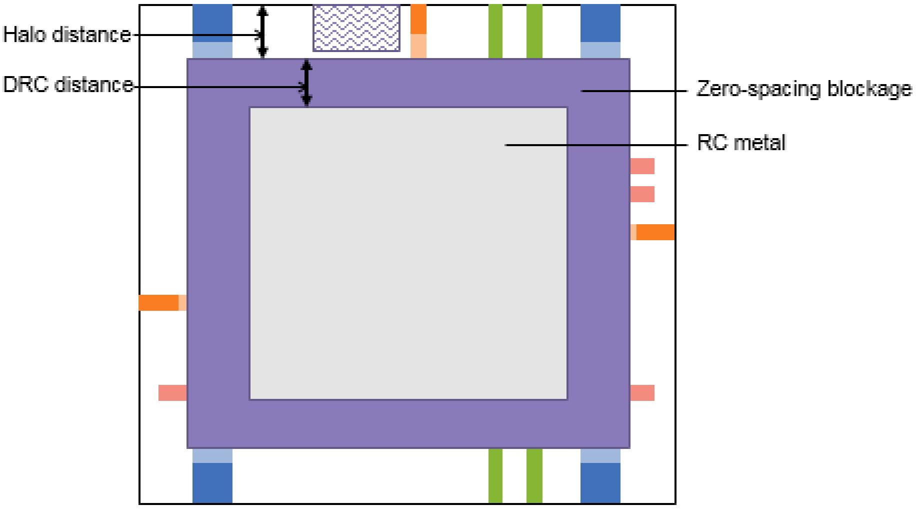
Figure 15 Trimming All Blockages Around Pins



Reducing the Physical Data by Generating a Core Blockage

For place and route accuracy, only the shapes around the pins and block boundary are required. To reduce the size of the frame views, the tool can generate a zero-spacing routing blockage over the core area (a core blockage). Figure 16 shows the parameters used to create the core blockage.

Figure 16 Core Blockage



The tool keeps all of the shapes within the halo distance of the cell boundary and does not create a blockage for this region. The tool creates a zero-spacing routing blockage for the region deeper than the DRC distance from the cell boundary. The tool keeps all of the shapes between the halo distance and the DRC distance. It discards the shapes that are deeper than the DRC distance and replaces them with a large RC metal shape.

If a pin is closer to the cell boundary than the channel distance, it is considered a *shallow pin*; otherwise, it is considered a *deep pin*. If a shallow pin is within the zero-spacing blockage, the tool cuts a channel to the pin to enable same-layer routing to the pin.

To enable creation of a core blockage, set the `lib.physical_model.block_all` application option to one of the following values:

- `true`

  The tool creates a core blockage on all metal layers, including routing layers and base layers.

- `used_layers`

  The tool creates a core blockage only on metal layers used by the cell.

- *`top_layer_name`*

  The tool creates a core blockage on the specified metal layer and all metal layers below it.

- `auto`

  The tool behavior depends on the design type of the cell.

  - For analog, black-box, and module design types, the tool creates a core blockage on all metal layers.
  - For macro cells, the tool creates a core blockage on the metal layers used by the cell.
  - For all other cells, the tool does not create a core blockage.

By default, the halo distance is 0 and the zero-spacing blockages extend to the cell boundary. To specify the halo distance, set the `lib.physical_model.block_core_margin` application option. The syntax to set this application option is

```
set_app_options
   -name lib.physical_model.block_core_margin
   -value { {layer1 halo_distance1}
            {layer2 halo_distance2} ...
            {layern halo_distancen} }
```

Note:

If you set the halo distance for a layer, the tool creates a core blockage for that layer, regardless of the setting of the `lib.physical_model.block_all` application option.

By default, the DRC distance is derived from the largest fat table spacing value defined for the layer in the technology file. To specify the DRC distance, set the `lib.physical_model.drc_distances` application option. The syntax to set this application option is

```
set_app_options
   -name lib.physical_model.drc_distances
   -value { {layer1 drc_distance1}
            {layer2 drc_distance2} ...
            {layern drc_distancen} }
```

By default, the channel distance is the `minSpacing` value plus the `minWidth` value specified for the layer in the technology file. To specify the channel distance, set the `lib.physical_model.pin_channel_distances` application option. The syntax to set this application option is

```
set_app_options
   -name lib.physical_model.pin_channel_distances
   -value { {layer1 channel_distance1}
            {layer2 channel_distance2} ...
            {layern channel_distancen} }
```

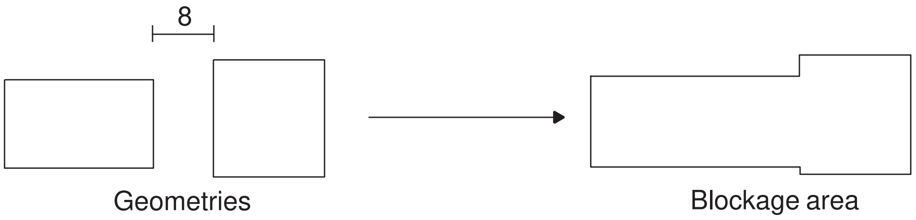
To further reduce the number of shapes kept in the frame view, the tool can

- Merge neighboring shapes if the space between them is less than twice the `minSpacing` value plus the `minWidth` value in the technology file (the merging threshold)

  When you set the `lib.physical_model.merge_metal_blockage` application option to `true`, the tool fills the space between neighboring shapes to create a single larger shape if the space between them is less than the merging threshold.

  For example, if the technology file specifies a `minSpacing` of 2 and a `minWidth` of 4 for a layer, the merging threshold is 2*2 + 4 = 8. Therefore, in Figure 17, the tool merges the geometries on the left to create the blockage area on the right.

Figure 17 Merging Same-Layer Geometries



- Remove non-pin shapes

By default, the tool keeps the non-pin shapes in the frame view. When you set the `lib.physical_model.remove_non_pin_shapes` to `all`, the tool removes all non-pin shapes from the frame view. When you set it to `core`, the tool removes the non-pin shapes that are completely within the core area.

Creating Zero-Spacing Blockages Around Pins

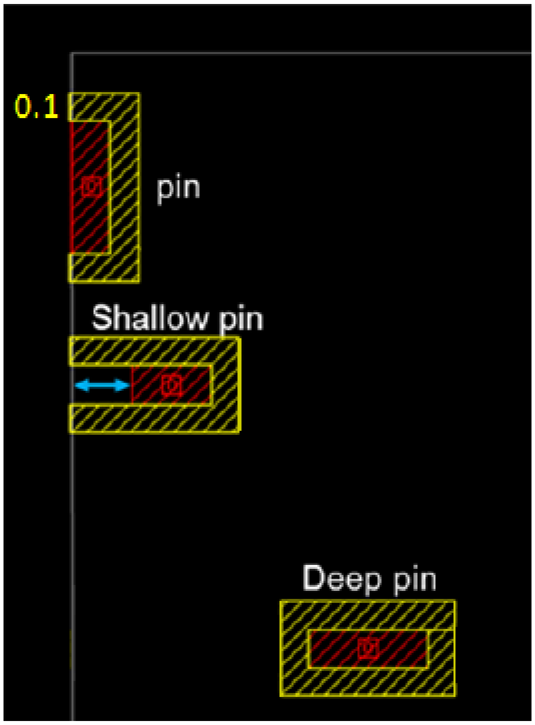
When you specify a positive halo distance for the core blockage, it can be difficult for the router to access the pins between the cell boundary and the core blockage because they can be accessed only from inside the cell. If these pins are not colored or placed off-track from the top-level design, routing DRC violations can occur.

To improve the accessibility by enabling access from outside the cell, the tool can create zero-spacing blockages around the pins. To create zero-spacing blockages around the pins, set the `lib.physical_model.create_zero_spacing_blockages_around_pins` application option. The syntax to set this application option is

```
set_app_options
   -name lib.physical_model.create_zero_spacing_blockages_around_pins
   -value { {layer1 blockage_width1}
            {layer2 blockage_width2} ...
            {layern blockage_widthn} }
```

If a pin is on the cell boundary or within the DRC distance of the cell boundary it is considered a shallow pin, and the tool does not create a blockage on the boundary side. Otherwise, the pin is considered a deep pin, and the tool creates a blockage of the specified width on all sides of the pin, as shown in Figure 18.

Figure 18 Zero-Spacing Blockages Around Pins



Creating Zero-Spacing Blockages Around Non-Pin Shapes

If your technology has voltage spacing requirements, you can enforce these requirements by creating zero-spacing blockages around fixed non-pin shapes in the frame view. The non-pin shapes include metal shapes, via shapes, and regular blockage shapes.

To create zero-spacing blockages around non-pin shapes, set the `lib.physical_model.keepout_spacing_for_non_pin_shapes` application option. The syntax to set this application option is

```
set_app_options
   -name lib.physical_model.keepout_spacing_for_non_pin_shapes
   -value {{layer1
 {edge_threshold1 ext_spacing1 corner_keepout_spacing1}}
          {layer2 {edge_threshold2 ext_spacing2 corner_keepout_spacing2}}
          ...
          {layern {edge_thresholdn ext_spacingn corner_keepout_spacingn}}
         }
```

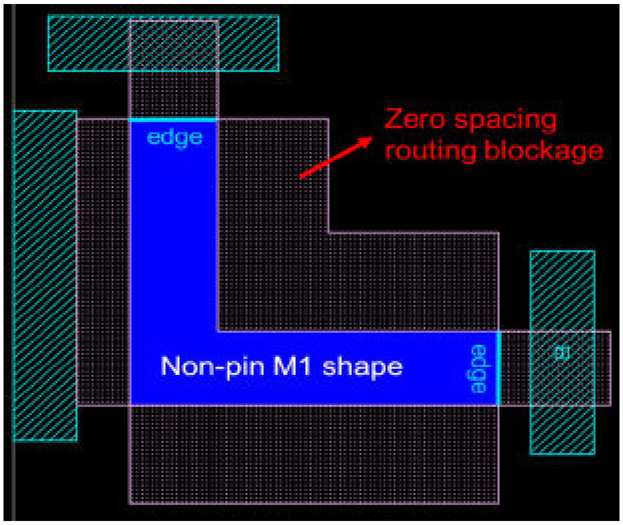
The *edge_threshold* argument defines the threshold for the length of an edge. An *edge* is a segment along the contour of a connected non-pin shape whose length is less than or equal to the specified value, and is not part of a concave corner. All other segments of the connected non-pin shape are *sides*. When generating the frame view, the tool trims sides by pins, but does not trim edges.

The *ext_spacing* argument defines the width of the zero-spacing blockage on the top, bottom, right, and left sides of the non-pin shapes. The *corner_keepout_spacing* argument defines the width of the blockages at the corners of the non-pin shapes.

For example, the following command creates the zero-spacing blockage shown in Figure 19.

```
lm_shell> set_app_options \
   -name lib.physical_model.keepout_spacing_for_non_pin_shapes \
   -value {M1 {0.55 0.7 0}}
```

Figure 19   Zero-Spacing Blockage Around Non-Pin Shape



Note:

To use this feature, you must ensure that core blockage creation is disabled for the affected layers.

Creating Via Regions

In a frame view, a via region is an area that consists of one or more rectangles, which might be overlapping, over a pin where the router can make a connection to a port. The via region provides guidance to the router about where to drop a via to make a connection to prevent DRC violations.

By default, the tool

- Generates via regions for cells with the following design types: library cells, diodes, end caps, fill cells, filler cells, physical-only cells, and well taps

  For all other design types, the tool generates pin access edges and directions, which guide the router to connect to the pins properly on the same layer.

  To generate via regions for all design types, set the `lib.physical_model.enable_via_regions_for_all_design_types` application option to `true`.

- Requires the router to place the via enclosure inside the pin shape when inserting vias in the via region

  This “within-pin” connection ensures that the router will not introduce any DRC violations in the lower metal layer of the cell being connected.

  In certain situations, such as when a pin is small and the minimum enclosure around the via is large, the “within-pin” connection is too restrictive. To allow the router to place the via enclosure outside the pin shape, set the `lib.physical_model.connect_within_pin` application option to `false`. When this option is `false` and the router cannot fit the lower-metal enclosure within the pin, it makes the connection anyway, which causes the lower-metal enclosure to extend beyond the pin shape. This type of connection is said to “change the pin shape.” In that case, the router must spend time checking for and fixing lower-metal design rule violations in the cell, which increases routing runtime.

  For large pins, you might want to restrict the connection region to a portion of the pin, which is referred to as the *must-connect area*. For information about creating must-connect areas, see Creating Pin Must-Connect Areas.

- Considers only the default vias and their rotations for all ports

  To consider nondefault vias when generating the via regions, specify the contact codes by setting the `lib.physical_model.include_nondefault_via` application option.

  To specify the contact codes to use for specific ports, set the `lib.physical_model.port_contact_selections` application option.

- Does not generate via regions for power and ground ports

  To generate via regions for specific power and ground ports, specify the ports by setting the `lib.physical_model.include_routing_pg_ports` application option.

See Also

- Updating Via Regions

Creating Pin Must-Connect Areas

To define a pin must-connect area, use one of the following methods:

- Explicitly Defining a Must-Connect Area
- Specifying Must-Connect Width Thresholds

Explicitly Defining a Must-Connect Area

This method uses shapes drawn on a spare layer to define the must-connect area. During frame view generation, if a pin shape overlaps the must-connect area, the tool creates zero-spacing blockages on the pin shapes, except in the must-connect area.

To use this method, define the must-connect area on a spare layer, and then associate the spare layers with the pin layers. To associate the layers, set the `lib.physical_model.pin_must_connect_area_layers` application option.

For example, to associate the spare CB layer with the M2 layer, use the following command:

```
lm_shell> set_app_options \
   -name lib.physical_model.pin_must_connect_area_layers \
   -value { {M2 CB} }
```

Figure 20 shows the pin shapes on the M2 layer, as well as the must-connect area defined on a spare layer (the CB layer). Figure 21 shows the resulting frame view, with the zero-spacing blockages shown in purple.

Figure 20 Pin Shapes and Must-Connect Area

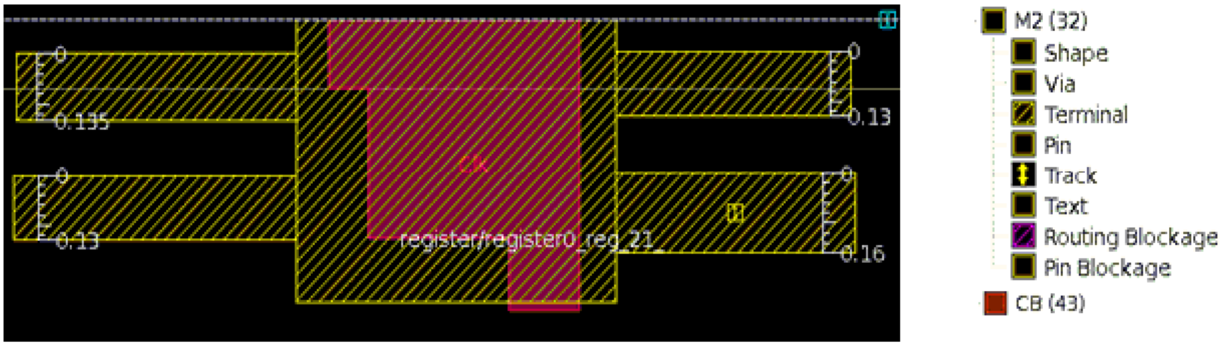
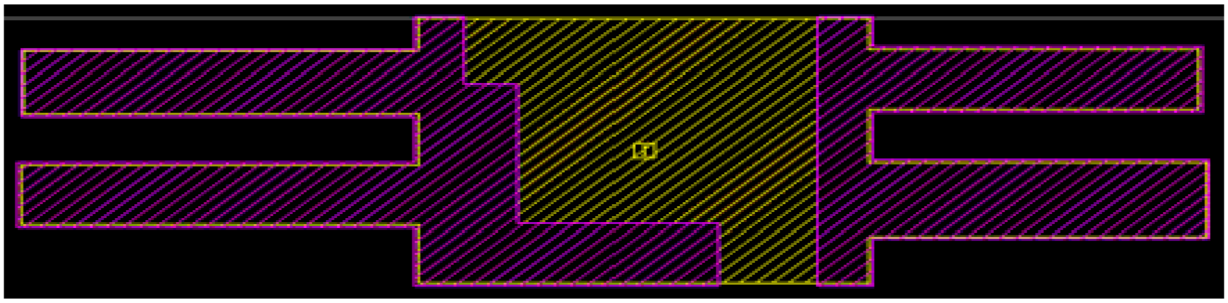


Figure 21 Frame View With Must-Connect Area



Specifying Must-Connect Width Thresholds

This method uses a width threshold to prevent pin connections on narrow shapes. During frame view generation, if the width of a pin shape is less than the specified threshold for that layer, the tool creates zero-spacing blockages on the pin shapes.

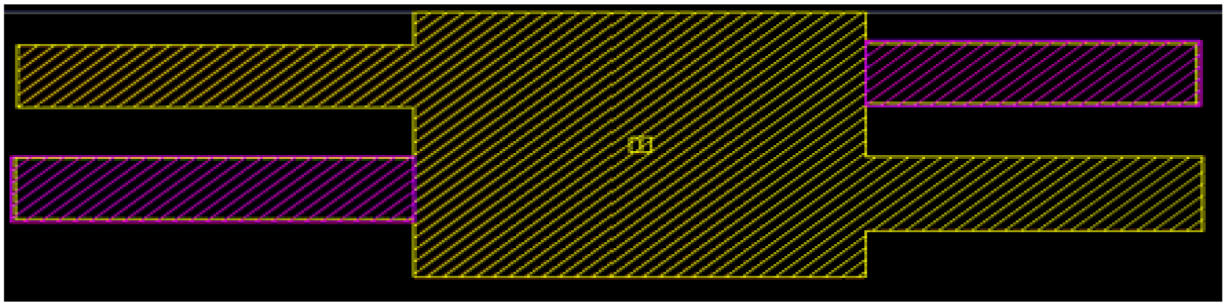
To define the width thresholds, set the `lib.physical_model.pin_must_connect_area_thresholds` application option.

For example, to specify a width threshold of 0.135 microns for the M2 layer, use the following command:

```
lm_shell> set_app_options \
   -name lib.physical_model.pin_must_connect_area_thresholds \
   -value { {M2 0.135} }
```

Figure 22 shows the resulting frame view, with the zero-spacing blockages shown in purple.

Figure 22 Frame View With Must-Connect Threshold



Deriving Must-Join Pins

When all terminals of a pin must be connected as a single net by the router, the pin is referred to as a *must-join pin* and is identified by a `pattern_must_join` attribute setting of `true`.

By default, frame view generation does not consider must-join pins. If you require that pins with multiple terminals on a layer be routed as must-join pins, set the `lib.physical_model.derive_pattern_must_join` application option to one of the following values:

- `true`

  Sets the `pattern_must_join` attribute to `true` on all pins with multiple terminals on the layer that meet one of the following criteria:

  - The `port_type` attribute of the pin is not set to one of the following power-related values: `power`, `ground`, `nwell`, `pwell`, `deep_nwell`, or `deep_pwell`.
  - The `port_type` attribute is set to one of the power-related values and the `is_secondary_pg` attribute is set to `true`.
  - The `port_type` attribute is set to one of the power-related values and the pin is specified in the `lib.physical_model.include_routing_pg_ports` application option.

- `exclude_pg`

  Sets the `pattern_must_join` attribute to `true` on all pins with multiple terminals on the layer whose `port_type` attribute is not set to one of the following power-related values: `power`, `ground`, `nwell`, `pwell`, `deep_nwell`, or `deep_pwell`.

Creating Frame Views for Hierarchical Cells

By default, when you create a frame view for a hierarchical cell, the tool

- Creates a frame view only for the top-level block

  To create frame views for the subblocks, set the `lib.physical_model.create_frame_for_subblocks` application option to `true`.

- Extracts the geometries in the subblocks

  To ignore the geometries in the subblocks, set the `lib.physical_model.hierarchical` application option to `false`.

Creating Colored Frame Views

To enable color-based frame view generation, set the `lib.physical_model.color_based_dpt_flow` application option to one of the following values:

- `true`

  Always use color-based frame view generation.

- `auto`

  Use color-based frame view generation only when the technology file contains double-patterning rules.

When color-based frame view generation is enabled, the tool represents the RC metal of each metal layer as color one shapes that are trimmed by both the colored and uncolored shapes with the maximum spacing specified in the technology file.

For example, Figure 23 shows the design view for a colored cell. Figure 24 shows the frame view generated for the block without the color-based flow. Figure 25 shows the frame view generated for the block with the color-based flow.

Figure 23 Design View for Colored Cell

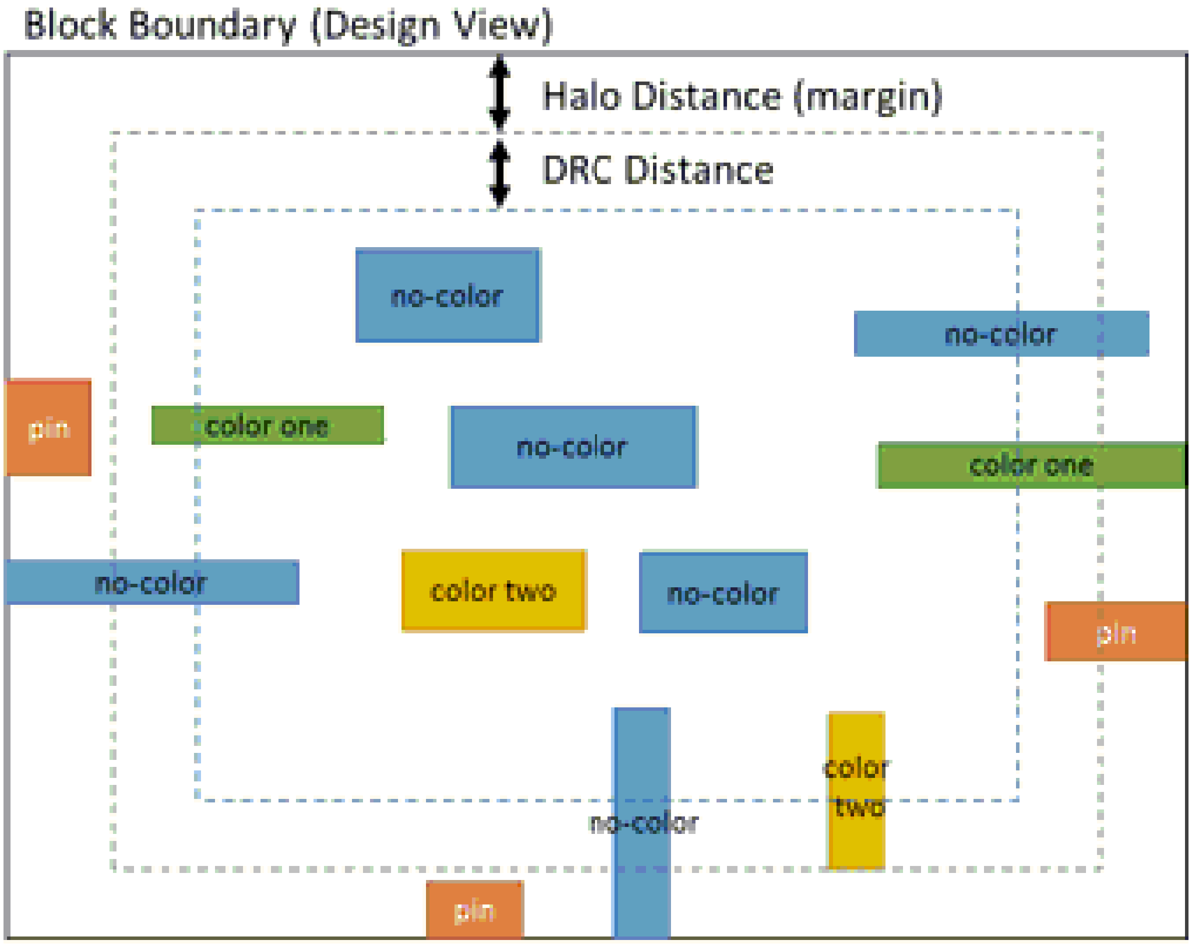


Figure 24 Frame View Without Color-Based Flow

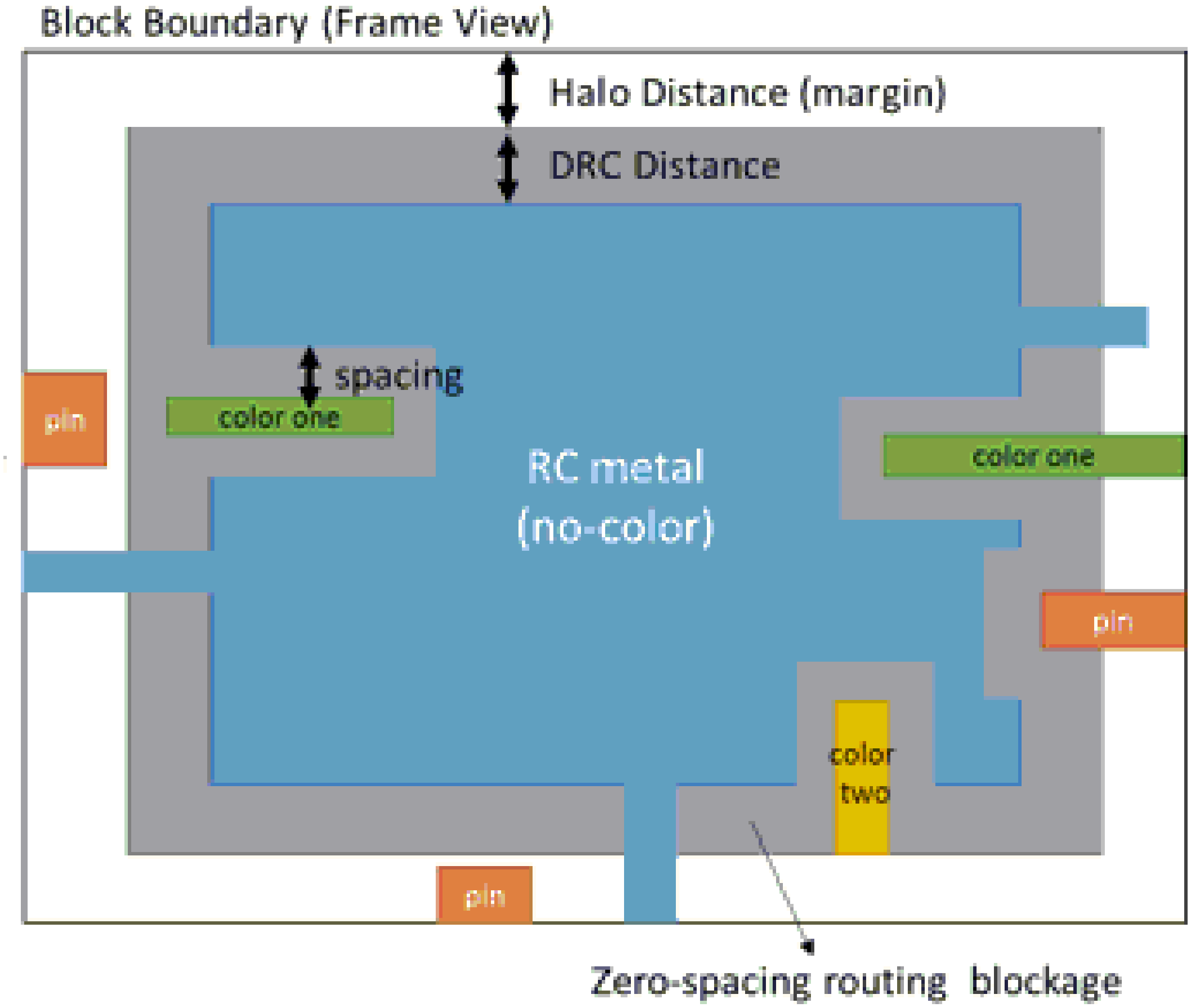
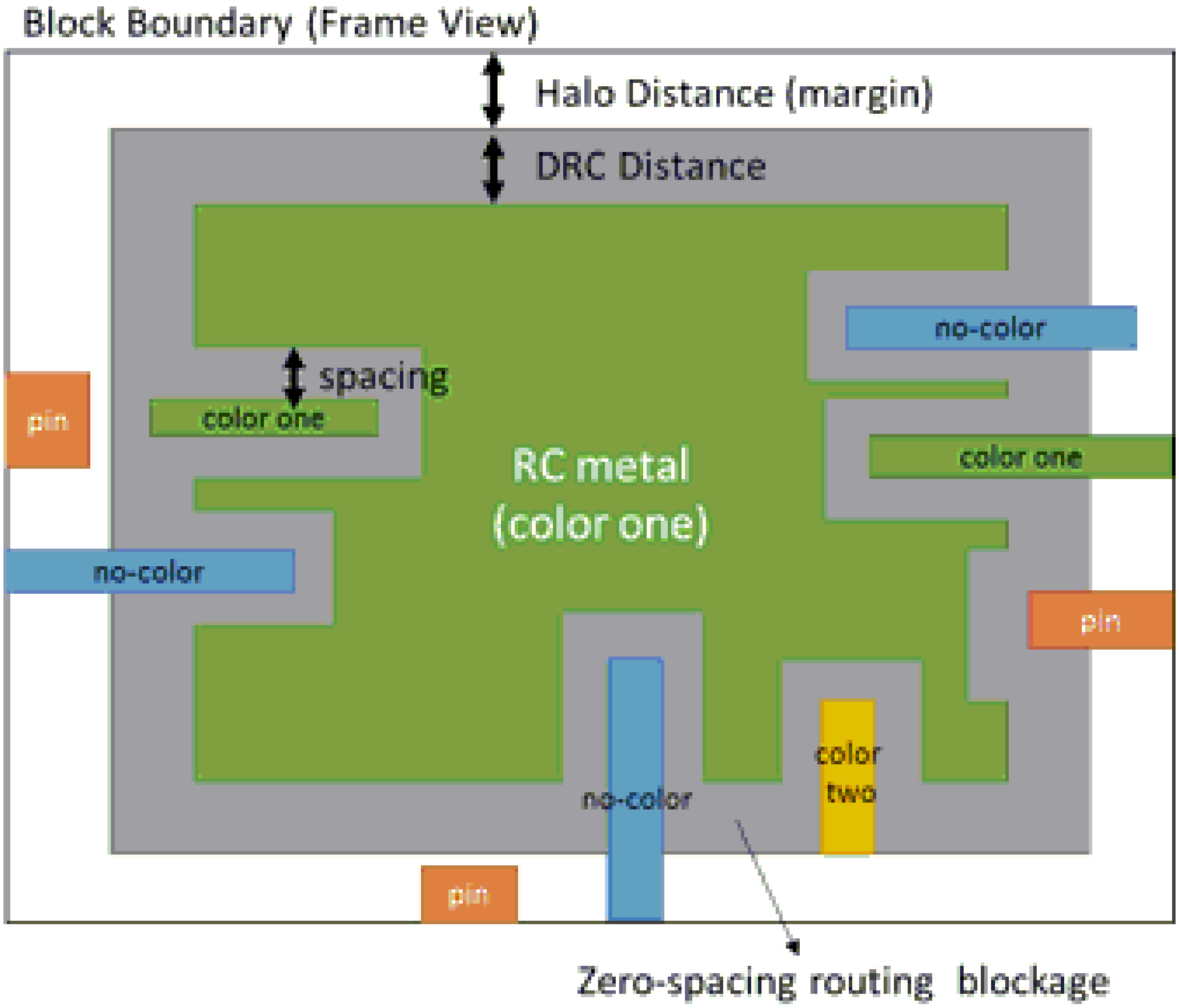


Figure 25 Frame View With Color-Based Flow



Mode-Based Configuration of Frame View Generation Options

To configure the frame generation options, you can either use the `configure_frame_options` command or the `read_lef -configure_frame_options` command.

When you use the `-configure_frame_options` option with the `read_lef` command, the tool calls the `configure_frame_options` command at the end of the `read_lef` run. The `read_lef` command automatically builds a frame view from the input LEF by determining appropriate frame view creation settings.

Feedback

The tool recognizes a design with a large obstruction as *abstract*, and a design without a large obstruction as *detailed*. An obstruction is considered large, the width and height of the blockage are larger than half the design boundary width and height, respectively. You can configure the frame generation options for the following situations by using the `configure_frame_options` command:

- A detailed standard cell contains a non-zero spacing obstruction that overlaps a pin

  In this case, use the `preserve_and_convert` configuration mode.
- A detailed standard cell without any obstruction issues

  In this case, use the `preserve_all` configuration mode.
- A detailed macro contains a non-zero-spacing obstruction that overlaps a pin, or has minimum spacing or less from a pin, or is without any obstruction issues

  In this case, use the `block_and_trim` configuration mode.
- An abstract macro contains an obstruction that you want to keep as-is in the frame view

  In this case, use the `preserve_all` configuration mode.
- An abstract macro cell contains a non-zero spacing obstruction that overlaps a pin

  In this case, use the `keep_obs_and_trim_touch_pin` configuration mode.
- An abstract macro cell where the spacing between the obstruction and the pin is less than or equal to the minimum spacing

  In this case, use the `keep_obs_and_trim_all_pin` mode.

In all other cases, use the default frame view generation option settings or individually set the frame view generation options, as needed.

Before using the `configure_frame_options` command, reset the frame view generation options to their default settings by using the following command:

```
lm_shell> reset_app_options -user_default {lib.physical_model.*}
```

Note that the `lib.physical_model.create_frame_for_subblocks` application option, which has a global scope, is not reset by this command.

Table 17 shows the option settings for the supported configuration modes.

Table 17 Frame View Generation Options for Configuration Modes

lib.physical_model application option	Configuration mode		
	block_and_trim	preserve_and_convert	preserve_all
`block_all`	`auto`	`false`	`false`
`block_core_margin`	`{used_layers auto_bloat}`	`{}`	`{}`
`preserve_metal_blockage`	`false`	`true`	`true`
`trim_metal_blockage_around_pin`	`{layer all}`	`{}`	`{}`
`convert_metal_blockage_to_zero_spacing`	`{}`	`{layer min_grid touch_pin}`	`{}`
`include_nondefault_via`	`auto`	`auto`	`auto`
`enable_via_regions_for_all_design_types`	`true`	`true`	`true`

Using the Exploration Flow

The exploration flow automatically analyzes your library source files and generates a script that you can use to create a cell library.

To use the exploration flow to build a cell library,

1. Define the search path, as described in Defining the Search Path.
2. Create the root library workspace for the exploration flow, as described in Creating a Library Workspace.
3. Load the logic library source files into the root workspace, as described in Loading the Logic Data.
4. Load the physical library source files into the root workspace, as described in Loading the Physical Data.
5. Analyze the library source files to automatically create the subworkspaces, as described in Performing Automated Library Analysis.
6. (Optional) Edit the subworkspaces.

7. (Optional) Enable aggregate library creation.

   To create an aggregate library from the subworkspaces, set the `lib.workspace.explore_create_aggregate` application option to `true`. For information about aggregate libraries, see Creating an Aggregate Library.

8. Validate the library workspace and commit it to disk.

   Note:

   In the exploration flow, library validation always allows missing library data. The validation behavior is similar to using the `check_workspace -allow_missing` command.

   You can use either of the following methods to validate and commit the library workspace when you use the exploration flow:

   - Validate and commit the exploration flow library workspace by running the `process_workspaces` command on it.

     When you run this command, the tool validates each subworkspace in the root workspace and, if the workspace passes validation, commits it to a cell library. This command performs the same functions on each subworkspace as the `check_workspace` and `commit_workspace` commands, but uses less peak memory when processing multiple subworkspaces.

     For more information about workspace validation, see Validating the Workspace. To specify validation options for the `process_workspaces` command, use the `-check_options` option.

     For more information about committing a workspace, see Committing the Workspace. The `process_workspaces` command accepts the same options as the `commit_workspace` command to control the commit process.

   - Generate a library preparation script and source it to create the cell library, as described in Using a Library Preparation Script.

To reduce the runtime of the exploration flow, use the fast exploration flow, as described in Using the Fast Exploration Flow.

Performing Automated Library Analysis

To analyze the library source files in the root workspace and partition them into a set of subworkspaces, use the `group_libs` command.

```
lm_shell> group_libs
```

This command partitions the library source files into subworkspaces based on the names of the library cells. It analyzes the logic libraries to identify both single-cell and multiple-cell logic libraries. The `group_libs` command checks only the library cell names; it does not check the pins, function, or timing arcs of the library cells.

For each subworkspace, the `group_libs` command adds the physical library source files that include at least the same set of cells. If the physical library source files contain cells that are not in any of the logic libraries, the `group_libs` command also creates a physical-only subworkspace.

By default, the `group_libs` command

- Checks for cell shadowing across subworkspaces, and fixes any detected issues

  Cell shadowing occurs when a library cell with the same name exists in multiple cell libraries. In this case, the tool uses the data from the first cell library in which it finds the cell, and it ignores that cell's data in all other cell libraries.

  By default, the `group_libs` command adjusts the grouping such that each cell exists in only one generated cell library, which includes all the PVT panes for the cell. To disable these fixes, set the `lib.workspace.group_libs_fix_cell_shadowing` application option to `false`.

- Tries to put as many macro cells as possible into a single subworkspace

  When using this strategy, the command groups all macro cells with the same set of characterization points into a single subworkspace. This strategy creates the smallest number of subworkspaces.

  To create subworkspaces that each contain a single macro cell with all of its characterization points, set the `lib.workspace.group_libs_macro_grouping_strategy` application option to `single_cell_per_lib` before running the `group_libs` command. Using this strategy can generate a large number of subworkspaces.

Feedback

- Assigns a name of <*root_workspace*>_physical_only to the physical-only subworkspace, if there is one

  To specify a name for the physical-only subworkspace, set the `lib.workspace.group_libs_physical_only_name` application option.

- Assigns names to the other subworkspaces by using a unique combination of logic library prefix and suffix characters

  To enable other naming strategies, set the `lib.workspace.group_libs_naming_strategies` application option.

  To report the names of the subworkspaces, use the `get_workspaces` command. To change a workspace name, use the `rename_workspace` command.

Note:

> If you change the set of source libraries in the root workspace or change the contents of any library in the root workspace, you must rerun the `group_libs` command to re-create the subworkspaces; otherwise, the subworkspaces might contain obsolete data.

Using a Library Preparation Script

When you use the exploration flow, the tool can generate a library preparation script to create the cell libraries. The generated script file performs the following tasks for each workspace:

1. Creates the library workspace.
2. Loads the logic and physical library source files into the library workspace.
3. Validates the contents of the library workspace.
4. Commits the library workspace to disk using the name of the first logic library as the cell library name.

In addition, if you enable aggregate library creation by setting the `lib.workspace.explore_create_aggregate` application option to `true`, the script creates an aggregate library that contains the cell libraries generated for the workspaces.

Note:

> The generated script file also includes any `set_app_options` and `reset_app_options` commands issued in the lm_shell session before running the `write_workspace` command.

To use this method to create the cell libraries,

1. Generate the library preparation script by using the `write_workspace` command.

   When you run this command, you must specify how to output the script:

   - Generate one script file per workspace

     To write one script file per workspace, use the `-directory` option to specify the directory in which to save the script files. The tool names each script file *workspace_name*.tcl. For example, to save the script files in a directory named libgen_scripts, use the following command:

     ```
     lm_shell> write_workspace -directory libgen_scripts
     ```

   - Generate one script file for all workspaces

     To generate a single script file, use the `-file` option to specify the file name. For example, to save the script in a file named libgen.tcl, use the following command:

     ```
     lm_shell> write_workspace -file libgen.tcl
     ```

   The default behavior of this command depends on the current workspace. If the current workspace is the root workspace, it generates a script for each subworkspace. If the current workspace is a subworkspace, it generates a script only for that workspace. To generate a script for specific workspaces, specify the workspaces as an argument to the command. For example, to generate a Tcl script only for the my_lib workspace, use the following command:

   ```
   lm_shell> write_workspace -file libgen.tcl my_lib
   ```

   You should review the generated script files, and modify as necessary to ensure that it meets your needs before using it to perform your library preparation.

2. Remove the existing root workspace, which also removes all of its subworkspaces, by using the `remove_workspace` command.

   ```
   lm_shell> remove_workspace
   ```

3. Create the cell libraries by sourcing the generated script.

   ```
   lm_shell> source file_name
   ```

   If the `check_workspace` command reports any issues, you must resolve the issues and rerun the script. For information about resolving the issues, see Validating the Workspace. In addition, you might need to modify the generated script to resolve some issues.

Using the Fast Exploration Flow

The fast exploration flow reduces the runtime required for the exploration flow by up to 50X by loading only the information required for library analysis by the `group_libs` command.

To enable this flow, set the `lib.workspace.fast_exploration` application option to `true` before loading the logic libraries.

When you use the fast exploration flow, you cannot use the `process_workspaces` command to check and commit the workspaces. Instead, you must use the `write_workspace` command to create a library preparation script and source this script to create the cell libraries.

The following script shows the commands used in the fast exploration flow:

```
set_app_options -name lib.workspace.fast_exploration -value true
read_db {my_db_files}
group_libs
write_workspace -file libprep.tcl
remove_workspace
source libprep.tcl
```

See Also

- Performing Automated Library Analysis
- Using a Library Preparation Script

Allowing Incomplete or Inconsistent Library Data

By default, the `check_workspace` command performs the checks described in Validating the Workspace. If a library workspace passes these checks, you can commit the workspace to create a cell library; otherwise, you must fix the reported errors before you can commit the workspace.

You can relax the workspace verification to enable the tool to generate a cell library even if the workspace has certain types of incomplete or inconsistent library data by defining a mismatch configuration. A *mismatch configuration* specifies the types of workspace verification errors that can be ignored or fixed by the tool. Cell libraries generated with this relaxed verification are called *prototype libraries*; they can be used only for preroute feasibility analysis.

The library manager provides two predefined mismatch configurations:

- `default` (the default)

  This is the default behavior of the workspace validation and must be used to generate cell libraries that can be used throughout the implementation flow.

- `auto_fix`

  This mismatch configuration enables the `check_workspace` command to repair the mismatches for all of the supported error types, which are shown in Table 18.

Table 18 Mismatch Configuration Error Types

Error type	**Description**
`lib_cell_name_case` Supported repairs: • `record_lib_cell_name_case_insensitivity`	Cell name mismatch due to case-sensitivity. Enables case-insensitivity when comparing cell names. Uses the cell name from the base library.
`lib_missing_logical_port` Supported repairs: • `create_placeholder_logic_lib_port`	Logic library cell is missing a signal port that exists in the base logic library or physical library cell. Creates a placeholder port on the logic library cell.
`lib_missing_physical_port` Supported repairs: • `create_placeholder_physical_lib_port`	Physical library cell is missing a port that exists in the base logic library. Creates a placeholder port on the physical library cell. The placeholder port is added on the M1 layer with a location of (0, 0). The terminal width and height are both equal to the minimum width defined for the M1 layer.
`lib_missing_physical_reference` Supported repairs: • `create_placeholder_physical_lib_cell`	A cell exists in the logic libraries, but not in any physical library source file. Creates a placeholder physical cell that has the same dimensions as the LEF unit site and the same number of ports as the cell in the logic libraries, but no real physical information.
`lib_missing_rail_pg` Supported repairs: • `create_pg_for_rail`	Logic library cell is missing a signal port that exists in the base logic library or physical library cell. Creates a placeholder port on the logic library cell.
`lib_port_name_case` Supported repairs: • `record_lib_port_name_case_insensitivity`	Pin name mismatch due to case-sensitivity. Enables case-insensitivity when comparing pin names. Uses the pin name from the base library.

Table 18 Mismatch Configuration Error Types (Continued)

Error type	Description
`lib_port_direction` Supported repairs: • `change_lib_port_direction`	A library cell pin has a different direction than in the base library. Uses the pin direction from the base library in the case of signal pin mismatches. Uses the pin direction from the physical library cell in the case of PG pin mismatches. **Note:** The library manager fixes many types of pin direction mismatches by default. For details, see Pin Definition Checks.
`lib_port_type` Supported repairs: • `change_lib_port_type`	A library cell pin has a different type than in the base library. Uses the pin type from the base library in the case of signal pin mismatches. Uses the pin type from the physical library cell in the case of PG pin mismatches. **Note:** The library manager fixes many types of pin type mismatches by default. For details, see Pin Definition Checks.

If these predefined configurations do not meet your needs, you can define your own configuration, as described in Creating a Mismatch Configuration.

To set a mismatch configuration, use the `set_current_mismatch_config` command, as described in Setting a Mismatch Configuration. To see the mismatches that were identified and fixed during the `check_workspace` command, use the `report_design_mismatch` command after validating the workspace.

Creating a Mismatch Configuration

To create a mismatch configuration,

1. Create the configuration by using the `create_mismatch_config` command.

   By default, when you create a user-defined mismatch configuration, it is based on the default mismatch configuration, and all of the error types are treated as error conditions. To specify a different mismatch configuration as the basis for the user-defined configuration, use the `-ref_config` option.

2. Specify how to handle a specific error type by setting its `action` attribute for the configuration.

   Specify one of the following values for the `action` attribute: `accept`, `error`, `ignore`, or `repair`. The initial setting of this attribute comes from the reference configuration used to create the user-defined configuration.

   To determine the available error types, use the `get_mismatch_types` command.

3. If you set the `action` attribute to `repair`, specify the repair process to use by setting the error type's `current_repair` attribute for the configuration.

   To determine the available repairs for an error type, query the `available_repairs` attribute of the error type. The Table 18 table describes many of the error types and their available repairs.

For example, the following commands create a user-defined mismatch configuration named user_def that repairs missing physical library cells.

```
lm_shell> create_mismatch_config user_def
lm_shell> set_attribute \
   [get_mismatch_types lib_missing_physical_reference] \
   action(user_def) repair
lm_shell> set_attribute \
   [get_mismatch_types lib_missing_physical_reference] \
   current_repair(user_def) create_placeholder_physical_lib_cell
```

See Also

- Setting a Mismatch Configuration
- Reporting Mismatch Configurations

Setting a Mismatch Configuration

By default, the library manager uses the default mismatch configuration. To set a different mismatch configuration, use the `set_current_mismatch_config` command. You can specify one of the predefined mismatch configurations (`default` or `auto_fix`) or a user-defined mismatch configuration.

For example, to set the mismatch configuration to the predefined auto-fix configuration, use the following command:

```
lm_shell> set_current_mismatch_config auto_fix
```

To see all available mismatch configurations, use the `report_mismatch_configs -all` command.

See Also

- Creating a Mismatch Configuration
- Reporting Mismatch Configurations

Reporting Mismatch Configurations

You can report mismatch configurations by using the commands described in the following table:

Command	Report
get_current_mismatch_config	Reports the name of the current mismatch configuration, such as `default` or `auto_fix`.
report_mismatch_configs	Reports the details of the current mismatch configuration.
report_mismatch_configs -all	Reports all available mismatch configurations
report_mismatch_configs -config_list *list*	Restricts the report to specific mismatch configurations.

In the following example, the tool reports the details of the current mismatch configuration, including the mismatch error types and repair strategies. The report includes mismatch error types that apply to both the library manager and the implementation tool; only the error types with a category of *library* apply to the library preparation process.

```
lm_shell> report_mismatch_configs
****************************************
Report : Reporting Mismatch Configs
****************************************
```

```
Config : default
-------------------------------------------------------------------------------------
Mismatch-Type   Category   Action     Allowed Actions
                           Strategy   Available Strategies
-------------------------------------------------------------------------------------
lib_missing_physical_reference
                library    error      {repair error accept }
                           null       {create_placeholder_physical_lib_cell }
lib_missing_logical_port
                library    error      {repair error accept }
                           null       {create_placeholder_logic_lib_port }
lib_missing_physical_port
                library    error      {repair error accept }
                           null       {create_placeholder_physical_lib_port }
lib_port_type   library    accept     {accept }
                           change_lib_port_type
                                      {change_lib_port_type }
lib_port_direction
                library    accept     {accept }
                           change_lib_port_direction
                                      {change_lib_port_direction }
lib_port_name_synonym
                library    ignore     {repair accept ignore }
                           null       {record_lib_port_name_synonym }
...
lib_cell_name_case
                library    ignore     {repair accept ignore }
                           null       {record_lib_cell_name_case_insensitivity }
lib_port_name_case
                library    ignore     {repair accept ignore }
                           null       {record_lib_port_name_case_insensitivity }
lib_missing_rail_pg
                library    ignore     {repair accept ignore }
                           null       {create_pg_for_rail }
...
1
```

See Also

- Creating a Mismatch Configuration
- Setting a Mismatch Configuration

Verifying Libraries

The Library Manager tool provides a verification flow to validate the logical information or frame views of a library. You can use this flow to

- Compare the logical information in a logic library source file and an existing cell library, as described in Verifying Logical Information in a Cell Library.
- Compare the frame views in two libraries. The libraries can be cell libraries or physical libraries, as described in Verifying the Frame Views Between Libraries.

Verifying Logical Information in a Cell Library

The library manager can verify that the following objects are the same in a cell library and a logic library source file:

- The library-level attributes
- The cells
- The pin information, timing arcs, and power data for each cell

To verify the logical information in a cell library,

1. Open the cell library in a verification workspace by using the `create_workspace -flow verification` command.

   ```
   lm_shell> create_workspace -flow verification mylib.ndm
   ```

   The cell library opened in the verification workspace is referred to as the *base library*.

2. Compare the logical information in a logic library source file to the cell library in the library workspace by using the `compare_db` command.

   ```
   lm_shell> compare_db -process_label myprocess mylib.db
   ```

   The logic library source file is referred to as the *comparison library*.

   The `compare_db` command uses the following tests to identify the pane (logic library) in the base library to compare to the comparison library:

   - An exact match between the name of the comparison library and the source file used to generate a pane in the base library
   - A matching pane based on process label, process number, temperature, and voltage

     To see the panes in the base library, use the `report_workspace -panes` command.

   If the `compare_db` command cannot find a matching pane, the comparison fails. If it finds a matching pane, it performs the comparison and reports any differences.

3. When you have finished verifying the cell library, remove the library workspace by using the `remove_workspace` command.

Verifying the Frame Views Between Libraries

The library manager can verify that the frame views are the same in two libraries by comparing the following information:

- The schema version
- The default site
- The number of frame views
- For each frame view, the design type, boundary, port information, number of terminals, terminal colors, existence of access edges, and via region contact codes

To verify the frame views,

1. Open a physical library or cell library in a verification workspace by using the `create_workspace -flow verification` command.

   ```
   lm_shell> create_workspace -flow verification mylib1.ndm
   ```

   The library opened in the verification workspace is referred to as the *base library*.

2. Compare the frame views in another library to the base library by using the `compare_ndm` command.

   ```
   lm_shell> compare_ndm mylib2.ndm
   ```

   The library opened by the `compare_ndm` command is referred to as the *comparison library*. It can be either a physical library or a cell library.

3. When you have finished verifying the cell library, remove the library workspace by using the `remove_workspace` command.

Modifying a Cell Library

You can modify the technology data and physical cell data, but not functional cell data, for an existing cell library. In general, modifications to a cell library must be made within an edit-mode library workspace.

To modify a cell library,

1. Define the search path by setting the `search_path` variable, as described in Defining the Search Path.

2. Create a library workspace for the edit flow by using the `create_workspace` command, as described in Creating a Library Workspace.

   For example, to modify the cell library stored in the my_ref.ndm directory, use the following command:

   ```
   lm_shell> create_workspace -flow edit my_ref.ndm
   ```

3. Modify the cell library by performing one or more of the following tasks:

   - Load new technology data, as described in Loading the Technology Data.

     To determine the consequences of loading new technology data into an existing cell library, use the `report_tech_diff` command.

   - Modify the technology data, as described in Completing the Technology Data.

   - Load physical library source files into the library workspace, as described in Loading the Physical Data.

   - Load a physical library or cell library into the library workspace by using the `read_ndm` command, as described in Loading Physical Data from an Existing Library.

     Note:

     When you load a cell library into an edit workspace, only the physical information is loaded; the timing information is not used.

   - Modify the physical attributes for one or more cells.

     In general, you can modify physical cell attributes by using the `set_attribute` command. You can also set certain attributes by loading a physical rules file, as described in Loading the Physical Rules File.

   - Update the frame views for one or more cells by using the `update_clib_frame` command.

> **Note:**
>
> You can update frame views directly in the on-disk library, without creating an edit-flow library workspace. For details, see the man page.

- Rename one or more cells by using the `rename_clib_cell` command.

  Note:

  > You can rename cells directly in the on-disk library, without creating an edit-flow library workspace. For details, see the man page.

- Remove one or more library cell by using the `remove_clib_cell` command.

  Note:

  > You can remove cells directly from the on-disk library, without creating an edit-flow library workspace. For details, see the man page.

- Update the via regions, as described in Updating Via Regions.
- Define antenna properties, as described in Defining Antenna Properties on Standard Cells and Defining Antenna Properties on Hard Macro Cells.
- Edit the layout for one or more cells, as described in Editing the Physical Layout.

4. Validate the contents of the library workspace by using the `check_workspace` command, as described in Validating the Workspace.

   When you validate the library workspace, the tool checks only the modified physical data. By default, the tool generates updated frame views only for cells with modified design views. To generate a frame view for a cell whose design view is unchanged, set the `frame_update` attribute on the cell to `true` before running the `check_workspace` command.

   If you want to use the same frame generation options that were used when the cell library was created, use the `write_frame_options` command to generate a Tcl script with the application option settings, and source this script before running the `check_workspace` command.

5. Commit the library workspace to disk by using the `commit_workspace` command, as described in Committing the Workspace.

For example, the following script updates the technology file for an existing cell library named existing.ndm and regenerates the frame views for all of the design views using the same application option settings that were used to generate the cell library:

```
write_frame_options -library existing.ndm -output frame_options.tcl
create_workspace –flow edit existing.ndm
read_tech_file new_tech.tf
set_attribute [get_lib_cells */*/design] frame_update true
```

```
source frame_options.tcl
check_workspace
commit_workspace -force -output existing.ndm
```

Updating Via Regions

A via region provides guidance to the router about where to drop a via to make a connection to prevent DRC violations. If your technology data changes or you need finer control of the via regions, use the following methods to update the via regions in the existing frame views:

- Remove all existing via regions and create new via regions by using the `derive_via_regions` command

  For details about via region creation and the application options used to modify the default behavior, see Creating Via Regions.

- Create new via regions by using the `create_via_region` command
- Remove existing via regions by using the `remove_via_regions` command

To update the via regions in the existing frame views, you must use the edit flow, as shown in the following example:

```
lm_shell> create_workspace -flow edit mylib.ndm
lm_shell> read_tech_file new.tf
lm_shell> derive_via_regions
lm_shell> check_workspace
lm_shell> commit_workspace
```

Defining Antenna Properties on Standard Cells

The implementation tools use antenna properties annotated on the frame view of the standard cells to perform antenna checking and fixing.

If the LEF file defines these antenna properties for the standard cells, they are automatically annotated on the frame view of the cells when you generate the cell library. Otherwise, you must use the edit flow to annotate these properties on the standard cells after generating the cell library.

To annotate antenna properties on a standard cell,

1. Open the cell library in edit mode as described in Modifying a Cell Library.
2. Use one of the following methods to set the antenna property attributes:
   - Use the `set_attribute` command to set the attributes directly.
   - Use the `read_clf_antenna_properties` command to set the attributes by reading a Cell Library Format (CLF) file.

   For information about the antenna property attributes for standard cells, see Antenna Properties for Standard Cells.
3. Validate the contents of the library workspace by using the `check_workspace` command, as described in Validating the Workspace.
4. Commit the library workspace to disk by using the `commit_workspace` command, as described in Committing the Workspace.

Antenna Properties for Standard Cells

A standard cell has antenna properties that define the gate and antenna areas of its input pins, diode protection value of its output pins, and whether a pin can be used as a diode. These settings are represented by attributes on the cell pins. You can set these attributes directly with the `set_attribute` command or set them by reading a CLF file with the `read_clf_antenna_properties` command.

Table 19 Standard Cell Antenna Property Attributes

Attribute	Description

`gate_area`	This attribute defines the gate area of an input pin.
	• If the library supports a single gate oxide mode, the attribute value uses the following syntax:
	`{ {`*`layer_name area_value`*`} ... }`
	• If the library supports multiple gate oxide modes, the attribute value uses the following syntax:
	`{ {`*`oxide_mode layer_name area_value`*`} ... }`
	Note:
	If this attribute is not specified for a pin, the IC Compiler II tool uses the setting of the `route.detail.default_gate_size` application option.

`antenna_area` `antenna_side_area`	These attributes define the antenna area of an input pin. The `antenna_area` attribute defines the antenna polygon area. The `antenna_side_area` attribute defines the antenna sidewall area. The attribute values use the following syntax: `{`*`layer_name area_value`*`}` **Note:** If these attributes are not specified for a pin, the implementation tool uses the pin shapes to compute the antenna area.
`diff_area`	This attribute defines the diode protection value of an output pin. The attribute value uses the following syntax: `{`*`layer_name area_value`*`}` **Note:** If this attribute is not specified for a pin, the implementation tool uses the setting of the `route.detail.default_diode_protection` application option.
`is_diode`	This attribute specifies whether a pin is a diode that can be used by the implementation tool to fix antenna violations.

Defining Antenna Properties on Hard Macro Cells

The implementation tools use antenna properties annotated on the frame view of the hard macro cells to perform antenna checking.

If you enable IC Validator antenna extraction during library validation, as described in Automatic Antenna Property Extraction Using the IC Validator Tool, these properties are automatically annotated on the frame view of the cells when you generate the cell library. Otherwise, you must use the edit flow to annotate these properties on the hard macro cells after generating the cell library.

To annotate antenna properties on a hard macro,

1. Open the cell library in edit mode as described in Modifying a Cell Library.
2. Use one of the following methods to set the antenna property attributes:
    - Use the `set_attribute` and `set_port_antenna_property` commands to set the attributes directly.
    - Use the `read_clf_antenna_properties` command to set the attributes by reading a Cell Library Format (CLF) file.

    For information about the antenna property attributes for standard cells, see Antenna Properties for Hard Macro Cells.
3. Validate the contents of the library workspace by using the `check_workspace` command, as described in Validating the Workspace.
4. Commit the library workspace to disk by using the `commit_workspace` command, as described in Committing the Workspace.

Antenna Properties for Hard Macro Cells

A hard macro cell has antenna properties that define the gate area, antenna area, and antenna ratio of its input ports and the diode protection value of its output ports. You can set these attributes directly with the `set_attribute` and `set_port_antenna_property` commands or set them by reading a CLF file with the `read_clf_antenna_properties` command.

Setting Hard Macro Antenna Properties Directly

To set the hard macro antenna properties directly,

1. Use the `set_attribute` command to set the diode protection value (`diff_area` attribute) on the hard macro output ports.

   Use the following syntax to specify this attribute:

   ```
   { {layer_name area_value} ... }
   ```

   For example,

   ```
   lm_shell> set_attribute mylib/hm1/z diff_area \
      {M1 0 M2 0 M3 0 M4 0.0595}
   ```

2. Use the `set_port_antenna_property` command to set the gate area, antenna area, and antenna ratio properties on the hard macro input ports. This command has the following syntax:

   ```
   set_port_antenna_property -port port -data property_settings_list
   ```

   In the *property_settings_list* argument, specify the properties for one or more layers using the following syntax:

   ```
   { {metalLayer layer_name gate_size area_value mode1_area mode1_data
      mode2_ratio mode2_data mode3_area mode3_data mode4_area mode4_data
      mode5_ratio mode5_data mode6_area mode6_data} ... }
   ```

   where

   - *layer_name* is the name of the metal layer in the technology file
   - *area_value* is the gate area of the port on the specified layer
   - *mode1_data* is the area of the specified metal layer when the connectivity is traced up to that metal layer
   - *mode2_data* is the maximum internal ratio of the area of the metal layer below the specified metal layer when the connectivity is traced up to the specified metal layer
   - *mode3_data* is the total metal area
   - *mode4_data* is the sidewall area of the specified metal layer when the connectivity is traced up to the specified metal layer
   - *mode5_data* is the maximum internal ratio of the sidewall area of the metal layer below the specified metal layer when the connectivity is traced up to the specified metal layer
   - *mode6_data* is the total metal sidewall area

The library manager also supports a simplified syntax that excludes the property names:

```
{ {layer_name area_value mode1_data mode2_data mode3_data
   mode4_data mode5_data mode6_data} ... }
```

For example, you can use either of the following commands to set the antenna properties on the a pin of the hm1 hard macro cell:

```
lm_shell> set_port_antenna_property -port mylib/hm1/a -data \
   { {metalLayer M1 gate_size 0.3436 mode1_area 6.664
      mode2_ratio 2.05346 mode3_area 12.708 mode4_area 17.5392
      mode5_ratio 7.04044 mode6_area 12.682}
     {metalLayer V1 gate_size 0.3436 mode1_area 6.664
      mode2_ratio 2.05346 mode3_area 12.708 mode4_area 17.5392
      mode5_ratio 7.04044 mode6_area 12.682} }
```

or

```
lm_shell> set_port_antenna_property -port mylib/hm1/a -data \
   { {M1 0.3436 6.664 2.05346 12.708 17.5392 7.04044 12.682}
     {V1 0.3436 6.664 2.05346 12.708 17.5392 7.04044 12.682} }
```

Note:

For the layer below the pin layer, set the hierarchical antenna properties to 0. Also, if the `gate_size` value for a metal layer is zero, such as for an output pin, the `mode2_ratio` and `mode5_ratio` values for that layer must be 0.

Creating an Aggregate Library

An aggregate library is grouping of multiple cell libraries into a single cell library for easier maintenance and distribution.

To create an aggregate library,

1. Define the search path, as described in Defining the Search Path.
2. Create a library workspace for the aggregate flow, as described in Creating a Library Workspace.

   When you create the library workspace, specify the aggregate library name as the workspace name. For example, to create an aggregate library named my_agg, use the following command:

```
lm_shell> create_workspace my_agg -flow aggregate
```

3. Load the cell libraries into the library workspace, as described in Opening a Cell Library.

   All of the cell libraries in the library workspace become part of the aggregate library. To remove libraries from the workspace, use the `remove_lib` command.

4. Validate the contents of the library workspace, as described in Validating the Workspace.

5. Set the search order, as described in Setting the Search Order.

6. Commit the library workspace to disk, as described in Committing the Workspace.

See Also

- Aggregate Library
- Modifying an Aggregate Library

Setting the Search Order

The search order specifies the order in which to search the cell libraries comprising the aggregate library when resolving a cell reference.

To specify the search order, use the `set_lib_order` command. When the implementation tool resolves a cell reference, it starts searching in the leftmost cell library in the list and uses the first matching cell it finds.

Modifying an Aggregate Library

To modify an aggregate library,

1. Define the search path, as described in Defining the Search Path.

2. Create a library workspace for the edit flow, as described in Creating a Library Workspace.

   When you create the library workspace, specify the library's root directory as the workspace name. For example, to modify the aggregate library stored in the my_agg.ndm directory, use the following command:

   ```
   lm_shell> create_workspace -flow edit my_agg.ndm
   ```

3. Adjust the cell libraries in the library workspace.

   When you create a library workspace for an existing aggregate library, its cell libraries are automatically opened in the library workspace. To see the cell libraries in the workspace, use the `get_libs` or `report_workspace` command.

To change the search order of the cell libraries, set the search order, as described in Setting the Search Order.

To add cell libraries to the library workspace, open the libraries, as described in Opening a Cell Library.

To remove cell libraries from the library workspace, close the libraries, as described in Closing a Cell Library.

4. Validate the contents of the library workspace, as described in Validating the Workspace.
5. Set the search order, as described in Setting the Search Order.
6. Commit the library workspace to disk, as described in Committing the Workspace.

See Also

- Creating an Aggregate Library

Opening a Cell Library

To query or view the contents of the cell library, you must open it outside of a library workspace by using the `open_lib` command.

Note:

To load a cell library into a library workspace, use the `read_ndm` command, as described in Loading Physical Data from an Existing Library.

For example, to open the mylib cell library, use the following command:

```
lm_shell> open_lib mylib.ndm
```

Note:

The `open_lib` command takes the path to the cell library rather than the library name. For libraries created with version N-2017.09 and later versions, this is the path to the library's root directory. For libraries created with earlier versions, this is the path to the library file.

Each time you run the command, you can specify only a single cell library to open; to open multiple cell libraries, you must run the command multiple times.

You can specify the cell library with an absolute path, a relative path, or no path. If you use a relative path or no path, the tool uses the search path to find the library.

See Also

- Defining the Search Path

Getting Information About Cell Libraries

You can report information about the cell libraries and their contents. To learn about these reporting capabilities, see

- Querying Libraries
- Reporting Libraries
- Reporting the Frame Properties
- Querying Library Objects
- Reporting Library Objects
- Querying Cell Area and Geometry of Cell Libraries

Querying Libraries

To list the libraries loaded into memory, use the `get_libs` command.

```
lm_shell> get_libs
{mylib mylib_lvt mylib_hvt}
```

If you have opened an aggregate library, the `get_libs` command returns the cell libraries within the aggregate library. The cell library names are prefixed with the aggregate library name.

```
lm_shell> get_libs
{AGG1|reflib1 AGG1|reflib2 }
```

You can specify a pattern to restrict the set of libraries. For example, to include only the low threshold voltage libraries, enter the following command:

```
lm_shell> get_libs *lvt
{mylib_lvt}
```

Reporting Libraries

To report information about a cell library, use the `report_lib` command. You must specify the name of an open cell library as an argument. You can run this command both before and after committing the library workspace.

By default, this command provides information about the library, such as the number of cells, the power rails, the characterization points (panes), and the operating condition definitions. Use the following options to report additional information about the cell library:

- `-antenna`

  Reports the library cells that are missing antenna properties and the affected pins.

- `-cell_summary`

  Reports the cell summary information of library cells in tabular form.

  The cell summary information table includes the cell name, cell panes, information about whether the cell has a frame view, cell-site name, width and height of the bounding box, single-height and multiple-height information, and cell type.

  When used with the `-cell_summary` option, the `-user_defined_cell_attributes` and `-user_defined_pin_attributes` options append the user-specified cell attribute and the user-specified pin attribute information to the cell summary information table.

- `-char_model`

  Reports data model information in addition to the default data model information.

  The default data model (such as nonlinear power model and CCS timing model) information lists models found in at least one cell in one pane of the library. The additional data model information can have one or more of the following blocks depending on model existence in the library:

  - The common data models present in all cells across all panes of the library.
  - The other common data models present in all cells in each pane of the library.
  - The other uncommon data models for the cells in each pane in tabular form.

- `-parasitic_tech`

  Reports the TLUPlus files loaded into the library, if any.

- `-physical`

  Reports the physical characteristics of the cell library.

  The reported information includes the site definitions, the layer definitions, the source file names, the number of each type of cell view (timing, layout, design, and frame) in the library, the number of each design type in the library, details about each library cell, the number of each type of library cell pin (ground, power, and signal) in the library, and details about the pins for each library cell.

- `-placement_constraints`

  Reports missing or improper placement constraints.

  Reported issues include improper site definitions and library cells that do not have any shapes on the placement layers defined in the technology file, such as implant and diffusion layers. By default, the report includes a maximum of 10 cells with violations. To report all cells with violations, use the `-verbose` option.

- `-routability`

  Reports library issues that could impact the routability.

  Reported issues include library cell pins that do not have either a via region or access edges, irregular cut shapes or routing blockages, frame views that are converted from IC Compiler FRAM views, improper frame generation settings, frame views with many obstructions, shorted pins, routing ports without geometry information, ports whose `is_secondary_pg` or `is_diode` attribute differs between the timing view and the frame view, and fully blocked terminals.

  To report details about the cell objects, such as the number of terminals, shapes, and routing blockages, and to report both fully and partially blocked terminals, use the `-verbose` option.

- `-timing_arcs`

  Reports the timing arcs for the library cells.

- `-wire_tracks`

  Reports library issues related to the wire tracks.

  Reported issues include cells with off-track or missing via regions and layers without a preferred direction. By default, the report includes the name and layer for each violating library cell pin, as well as the issue causing the violation. To report additional details for the violating library cell pins such as the contact code and bounding box, use the `-verbose` option.

  By default, the library manager determines the wire tracks from the library being reported. If the wire track information is stored in a technology library, use the `-technology_lib` option to specify the library.

- `-wire_track_colors`

  Reports library issues related to wire track coloring.

Reported issues include tracks without color information, terminals without color information, and terminals on a track with the incorrect color.

- `-min_pin_layer` *`layer`*

  Reports terminals on layers lower than the specified pin layer.

Example 10 shows an example of the default cell library report.

Example 10   Cell Library Report

```
lm_shell> report_lib mylib
****************************************
Report : library
Library: mylib
****************************************

Full name: /usr/libs/mylib.ndm:mylib
File name: /usr/libs/mylib.ndm
Lib type: cell
Design count: 434
Timing data:

Power rails:

Index  Name         Type
  0    <default>    power
  1    V            power
  2    VL           power
  3    VSS          ground

Pane count: 2

Pane 0:
  Process label: (none)
  Process number: 1
  Voltage rail count: 3
  Voltage for rail 0 (<default>): 1.08
  Voltage for rail 1 (V): 1.08
  Voltage for rail 2 (VL): 1.08
  Temperature: 125

  Thresholds:
    r/f InputDelay: 0.5/0.5  r/f OutputDelay: 0.5/0.5
    l/h RiseTrans:  0.1/0.9  h/l FallTrans:   0.9/0.1
    TransDerate:    1

 Source .db file:
    /usr/libs//DB/mylib_125c.db
Pane 1:
  ...
```

```
Source .db libraries:
    mylib_125c.db:mylib_125c
    mylib_40c.db:mylib_40c


Operating Conditions:

   Name                 Process   Temp      Voltage   Original DB Name
------------------------------------------------------------------------
   WORST                1.00      125.00    1.08      mylib_max_hth
   BEST                 1.00      -40.00    1.32      mylib_min

1
```

Reporting the Frame Properties

The library manager extracts frame views of the library cells when you run the `check_workspace` command.

To report the frame view properties for one or more library cells, use the `report_frame_properties` command. You can run this command only after committing the library workspace. You must use the `-library` option to specify the name of an open cell library and the `-output` option to specify the name of the output file.

```
lm_shell> open_lib mylib
lm_shell> report_frame_properties -library [current_lib] \
   -output frame.txt
```

By default, the command reports the frame view properties for all cells in the cell library. To report the frame view properties for a specific cell, use the `-block` option to specify the cell name. Note that when you use this option, you must explicitly specify the frame view of the cell.

Example 11 shows an example frame properties report, which was generated by the following command:

```
lm_shell> report_frame_properties -library [current_lib] \
   -output frame.txt -block mylib/AND2/frame
```

Example 11   Frame Properties Report

```
****************************************
Report : frame properties
Library: mylib

****************************************
**********************
Design : AND2
**********************
Total number of row: 1.
```

```
***** implant_width property *****
      No implant width property on this block.

***** diffusion_width_height property *****
      No diffusion width height property on this block.

***** source_drain annotation *****
      No source/drain annotation on this block.
```

See Also

- Frame View Generation Application Options

Querying Library Objects

You can query the library cells and library cell pins both before and after committing the library workspace.

- To determine the library cells available in a library, use the `get_lib_cells` command.

  If you run the command without any arguments, it returns all the timing views in the current library. If the library does not contain any cells with timing views, the command returns the frame views, if they exist; otherwise, it returns the design views.

  If you specify the library cells, use the following format:

  library_name/*cell_name*[/*view*]

  You must specify both the library name and the cell name. You can specify a pattern, including wildcards for the library name and cell name. If you specify the view, it must be one of the following keywords: `design`, `frame`, `layout`, or `timing`.

  For example, to return all the frame views in the mylib cell library, use the following command:

  ```
  lm_shell> get_lib_cells mylib/*/frame
  {mylib/AND2/frame mylib/AND4/frame mylib/AO21/frame ...}
  ```

- To determine the pins on a library cell, use the `get_lib_pins` command.

  If you run the command without any arguments, it returns all the library cell pins in the current library. If you specify the library cells, use the following format:

  library_name/*cell_name*/*pin_name*

  You must specify the library name, the cell name, and the pin name. You can specify a pattern, including wildcards for the library name, cell name, and pin name.

For example, to return all the pins of the AND2 cell in the mylib cell library, use the following command:

```
lm_shell> get_lib_pins mylib/AND2/*
{mylib/AND2/IN1 mylib/AND2/IN2 mylib/AND2/OUT}
```

Note:

To reduce runtime when you load physical library source files in the physical-only flow, the tool loads only the cell names, and not detailed information for the cells. The tool loads the detailed information during workspace validation. This means that after loading the physical library source files, you can query the library cell names, but you cannot query other information such as pins, attributes, and shapes until after workspace validation.

If you need to examine the cells in detail before workspace validation, load the physical library source files in the normal flow.

See Also

- Validating the Workspace

Reporting Library Objects

You can report information about the library cells, library cell pins, and library timing arcs. You can generate the library cell and library cell pin reports both before and after committing the library workspace. You can generate the library timing arc report only after committing the library workspace.

- To report information about one or more library cells, use the `report_lib_cells` command. You must use the `-objects` option to specify the library cells.

  Use the following format to specify the library cells: *library_name/cell_name*. You must specify both the library name and the cell name. You can specify a pattern, including wildcards for the library name and cell name.

  By default, the report includes the full name, area, pin count, and design type for the specified cells, as well as whether the cell is sequential.

  For example, to report on the AND2 cell in the mylib cell library, use the following command:

  ```
  lm_shell> report_lib_cells -objects mylib/AND2
  ****************************************
  Report : lib_cell
  ****************************************

  full_name            area       pin_count is_sequential design_type
  -------------------- ---------- --------- ------------- ------------
  mylib/AND2           11.0005    3         false         lib_cell
  1
  ```

  You can customize the report by using the `-columns` option to specify the information to report. The following column names are valid for this report: `area`, `design_type`, `dont_touch`, `full_name`, `is_combinational`, `is_isolation`, `is_level_shifter`, `is_memory_cell`, `is_sequential`, `is_three_state`, `name`, `pad_cell`, `pin_count`, `valid_purposes`, and `view_name`. To display all of the available information, specify `-columns all`.

- To report information about one or more library cell pins, use the `report_lib_pins` command. You must use the `-objects` option to specify the library cell pins.

  Use the following format to specify the library cell pins: *library_name/cell_name/pin_name*. You must specify the library name, the cell name, and the pin name. You can specify a pattern, including wildcards for the library name, cell name, and pin name.

  By default, the report includes the full name and direction for the specified pins.

For example, to report on all the pins of the AND2 cell in the mylib cell library, use the following command:

```
lm_shell> report_lib_pins -objects mylib/AND2/*
****************************************
Report : lib_pin
****************************************

full_name                 direction
------------------------- --------
mylib/AND2/IN1            in
mylib/AND2/IN2            in
mylib/AND2/OUT            out
1
```

You can customize the report by using the `-columns` option to specify the information to report. The following column names are valid for this report: `direction`, `disable_timing`, `full_name`, `function`, `is_async_pin`, `is_clock_pin`, `is_data_pin`, `is_pad`, `is_preset_pin`, `is_three_state`, `is_three_state_enable_pin`, `is_three_state_output_pin`, `name`, `pin_number`, `port_type`, `rail_name`, `signal_type`, and `three_state_function`. To display all of the available information, specify `-columns all`.

- To report information about one or more timing arcs, use the `report_lib_timing_arcs` command. You must use the `-objects` option to specify the library timing arcs.

  The easiest way to specify the timing arcs is to use the `get_lib_timing_arcs` command and use the `-of_objects` option to specify the library cell for which you want to report the timing arcs.

  By default, the report includes the from pin, to pin, sense, when condition, and SDF condition for the specified timing arcs.

  For example, to report on all the timing arcs of the AND4 cell in the mylib cell library, use the following command:

```
lm_shell> report_lib_timing_arcs \
   -objects [get_lib_timing_arcs -of_objects mylib/AND2]
****************************************
Report : lib_timing_arc
****************************************

from   to     sense           when       sdf_cond        is_disabled
------ ------ --------------- ---------- --------------- -----------
A1     Y      positive_unate                             false
A2     Y      positive_unate                             false
A1     Y      positive_unate                             false
A2     Y      positive_unate                             false
1
```

You can customize the report by using the `-columns` option to specify the information to report. The following column names are valid for this report: `from`, `from_lib_pin`, `is_disabled`, `is_user_disabled`, `sdf_cond`, `sense`, `to`, `to_lib_pin`, and `when`. To display all of the available information, specify `-columns all`.

Querying Cell Area and Geometry of Cell Libraries

A physical rules file includes information about layer density that can be used to generate and store the layer geometry and area information into the library cell. For information about loading the layer density related syntax into the cell library from the physical rules files, see Loading the Physical Rules File.

To query the layer geometry and area information,

1. Define the following attributes in the given format:

   - *`layername`*`_density_area` of type `float`
   - *`layername`*`_density_shapes` of type `string`

   `layername` is the layer for which you are querying the information , such as `poly`, and must be defined in the `saved_layers` attribute in the physical rules file syntax. For more information, see Table 6.

   For example,

   ```
   define_user_attribute poly_density_area -type float -classes lib_cell
   define_user_attribute poly_density_shapes -type string \
                                             -classes lib_cell
   ```

2. To obtain the value of these attributes, use the `get_attribute` command as shown in the following example:

   ```
   get_attribute [get_lib_cell */AND/frame] poly_density_area
   get_attribute [get_lib_cell */AND/frame] poly_density_shapes
   ```

Exporting Cell Library Content

You can export cell library content to several formats for use with other tools. Table 20 lists the supported formats and the command used to generate the output files.

Table 20 Cell Library Export Formats

Format	Command
Technology file	`write_tech_file`

Table 20 Cell Library Export Formats (Continued)

Format	Command
Antenna properties in Cell Library Format (CLF)	`write_clf_antenna_properties`
LEF	`write_lef`
GDSII	`write_gds`
OASIS	`write_oasis`
Reference library for Design Compiler Graphical or Design Compiler NXT	`write_dc_fram`

Closing a Cell Library

To close a cell library that was opened outside of a library workspace, use the `close_lib` command. This command decrements the open count of the library and removes it from memory when no copies of the library remain open.

Note:

> To remove a cell library from a library workspace, use the `remove_lib` command.

By default, the `close_lib` command closes the current library. To close a specific library, specify the library name. To close all libraries, use the `-all` option.

For example, to close the current library, use the following command:

```
lm_shell> close_lib
```

To close the mylib cell library, use the following command:

```
lm_shell> close_lib mylib
```

To close all cell libraries, use the following command:

```
lm_shell> close_lib -all
```

A

Library Manager Application Options

To learn about the application options you can use to control the behavior of the IC Compiler II library manager, see the following topics:

- Library Manager Application Options
- LEF Input Application Options
- GDSII and OASIS Input Application Options
- Timing View Generation Application Options
- Frame View Generation Application Options
- Library Setting Application Options
- Antenna Extraction Application Options
- Command-Line Interface Application Options
- Graphical User Interface Application Options

Library Manager Application Options

The library manager (`lib.workspace`) application options control various aspects of the cell library creation process. Table 21 describes the `lib.workspace` application options. If the application option has a data type of Boolean or enumerated string, the valid values are listed, with the default listed first and shown in purple; otherwise, the data type is specified along with the default.

Table 21 lib.workspace Application Options

Application option	Valid values or data type	Description
`lib.workspace.allow_commit_workspace_overwrite`	**false** | true	Controls whether the `commit_workspace` command overwrites an existing cell library (`true`) or issues an error message if the library already exists (`false`).
`lib.workspace.allow_missing_related_pg_pins`	**true** | false	Controls whether the `check_workspace` command requires signal pins to have at least one related power or ground pin.
`lib.workspace.allow_loading_layout_view_only_lib`	**false** | true	Loads layout-view-only physical libraries to the workspace.
`lib.workspace.create_workspace_tf_verbose`	**true** | false	When `true`, the `create_workspace -technology` command outputs detailed information when reading the technology file.
`lib.workspace.enable_secondary_pg_marking`	**true** | false	Controls whether secondary power and ground pins are annotated on the frame view.
`lib.workspace.exclude_design_filters`	list (default = ””)	Specifies one or more patterns to match designs to exclude when loading logic or physical libraries. This option works with the `lib.workspace.include_design_filters` application option to determine the set of designs to read. For more information, see Specifying Which Blocks to Process.

Table 21 lib.workspace Application Options (Continued)

Application option	Valid values or data type	Description
lib.workspace.explore_create_aggregate	**false** | true	Controls the behavior of the write_workspace and commit_workspace commands when the exploration flow creates multiple workspaces. To create an aggregate library, set this application option to true.
lib.workspace.fast_exploration	**false** | true	Controls whether the fast exploration flow is enabled. For more information, see Using the Exploration Flow.
lib.workspace.group_libs_fix_cell_shadowing	**true** | false	Controls whether the group_libs command modifies the grouping strategy to prevent cell shadowing issues across subworkspaces. Cell shadowing occurs when a library cell with the same name exists in multiple cell libraries. In this case, the tool uses the data from the first cell library in which it finds the cell, and it ignores that cell’s data in all other cell libraries, which can result in missing data for the cell.
lib.workspace.group_libs_macro_grouping_strategy	**aggregate_single_cell** | single_cell_per_lib	Controls the macro grouping strategy used for the subworkspaces created by the group_libs command. For more information, see Performing Automated Library Analysis.
lib.workspace.group_libs_naming_strategies	list (default = ””)	Controls the naming strategy used for the subworkspaces created by the group_libs command. By default, the tool creates the subworkspace names by using a unique combination of logic library prefix and suffix characters. For more information, see Performing Automated Library Analysis.

Table 21 lib.workspace Application Options (Continued)

Application option	Valid values or data type	Description
lib.workspace.group_libs_physical_only_name	string (default = "")	Specifies the name of the physical-only library group created by the group_libs command. If this application option has a value of "", the command names the physical-only subworkspace <*root_workspace*>_physical_only. For more information, see Performing Automated Library Analysis.
lib.workspace.include_design_filters	string (default = "")	Lists the filters for read commands to include designs.
lib.workspace.keep_all_physical_cells	**false** | true	Keeps physical cells even when there is no logic cell.
lib.workspace.include_design_filters	list (default = "")	Specifies one or more patterns to match designs to include when loading logic or physical libraries. When you set this option, only the specified designs are read. This option works with the lib.workspace.exclude_design_filters application option to determine the set of designs to read. For more information, see Specifying Which Blocks to Process.
lib.workspace.library_developer_mode	**false** | true	Controls whether the technology file checker works in user mode (the default) or developer mode.
lib.workspace.remove_frame_bus_properties	**false** | true	Controls whether the check_workspace command removes bus properties that exist in the frame view but not in the timing view.
lib.workspace.save_design_views	**false** | true	Controls whether the commit_workspace command saves the design views in the cell library.
lib.workspace.save_layout_views	**false** | true	Controls whether the commit_workspace command saves the layout views in the cell library.

Feedback

LEF Input Application Options

The LEF input (file.lef) application options control the behavior of the read_lef command. Table 22 describes the file.lef application options. If the application option has a data type of Boolean or enumerated string, the valid values are listed, with the default shown in purple; otherwise, the data type is specified along with the default.

Table 22 file.lef Application Options

Application option	Valid values or data type	Description
file.lef.allow_empty_pin	**false** | true	Controls whether the read_lef command allows a PIN statement that does not have an associated PORT.
file.lef.auto_rename_conflict_sites	**false** | true	Controls whether the read_lef command automatically renames conflicting site names. For more information about this application option, see Reading LEF Files.
file.lef.derive_must_join_for_pg	**false** | must_join_group | pattern_must_join	Controls how the read_lef command derives must-join pin connections for PG pins that have the LEF58_MUSTJOINALLPORTS property in the LEF file.

Table 22 file.lef Application Options (Continued)

Application option	Valid values or data type	Description
file.lef.derive_must_join_for_signal	**pattern_must_join** | false | must_join_group	Controls how the read_lef command derives the must-join port signal pins that have the LEF58_MUSTJOINALLPORTS property in the LEF file and derive it to IC Compiler II library attributes. The write_lef command writes out the LEF58_MUSTJOINALLPORTS property under pin section for signal pins. The list expects two values, the first value is for input signal pins and the second is for output signal pins. Usage: set_app_options -name file.lef.derive_must_join_for_signal -value { inputPinVal outputPinVal } For example: To set input pin as must_join_group and output pin as pattern_must_join, use the set_app_options -name file.lef.derive_must_join_for_signal -value { must_join_group pattern_must_join } command.
file.lef.non_real_cut_obs_mode	**true** | false	Controls whether to use zero minimum spacing for non-real cut obstructions.

GDSII and OASIS Input Application Options

The `read_gds` and `read_oasis` commands support similar application options to control their behavior. Table 23 describes these application options; replace *format* in the application option names with `gds` for the `read_gds` command and `oasis` for the `read_oasis` command. These application options also affect the `trace_connectivity` commands.

In the following table, if the application option has a data type of Boolean or enumerated string, the valid values are listed, with the default shown in purple; otherwise, the data type is specified along with the default.

Table 23 file.gds Application Options

Application option	Valid values or data type	Description
`file.`*`format`*`.create_custom_via`	**`false`** | `true`	Controls whether the commands create custom vias from the cut shapes in the library source file.
`file.`*`format`*`.exclude_layers`	list (default = “”)	Specifies the rules for mapping between metal layers and the layers that are used to exclude geometries from the specified layers. For more information about this statement, see Mapping Exclude Layers.
`file.`*`format`*`.port_type_map`	list (default = “”)	Identifies the power and ground pins by mapping the pin name to its type. The mapping applies to all cells in the library source file. For more information about this application option, see Identifying Power and Ground Pins.
`file.`*`format`*`.text_layer_map`	list (default = “”)	Maps the text layers to the metal layers so that the tool can associate metal shapes with a net or pin when the text is not on the same layer as the metal shapes. For more information about this application option, see Mapping Text Layers.
`file.`*`format`*`.trace_copy_overlap_shape_from_sub_cell`	**`false`** | `true`	Controls whether the commands duplicate a shape in a subblock to the top-level block when the shape is overlapped by terminals.

Table 23 file.gds Application Options (Continued)

Application option	Valid values or data type	Description
file.*format*.trace_terminal_length	integer (default = 0)	Controls whether the commands split long pins on the block boundary into a terminal and a net shape. By default (0), the commands do not split long pins. When set to a positive integer, the commands split pins that are longer than the specified value. After splitting, the new terminal is the specified length; the rest of the pin is converted to a net shape.
file.*format*.trace_terminal_type	**default** | pg | signal	Specifies the types of long pins that are considered for splitting. By default, both PG and signal pins are considered.
file.*format*.trace_unmapped_text	**false** | true	Controls whether the commands trace all routing layers for text that is not mapped to any routing layer.
file.*format*.use_only_mapped_text	**false** | true	Controls whether the commands use only the text specified by the file.*format*.text_layer_map application option when tracing ports.

Timing View Generation Application Options

The timing view generation (lib.logic_model) application options control the timing view generation, which is performed by the check_workspace command. Table 24 describes the lib.logic_model application options. If the application option has a data type of Boolean or enumerated string, the valid values are listed, with the default shown in purple; otherwise, the data type is specified along with the default.

Table 24 lib.logic_model Application Options

Application option	Valid values or data type	Description
lib.logic_model.auto_remove_incompatible_timing_designs	**false** | true	Controls whether the check_workspace command automatically removes cells from the cell library that have a mismatch between the base library and additional logic libraries.
lib.logic_model.auto_remove_timing_only_designs	**false** | true	Controls whether the check_workspace command automatically removes cells from the cell library that do not exist in the physical library.
lib.logic_model.preserve_nldm_pin_caps	**false** | true	Controls whether the composite current source (CCS) reciever_capacitance data overwrites the native nonlinear delay model (NLDM) pin capacitance when reading the logic libraries.
lib.logic_model.relax_pin_equal_opposite_attr_check	**false** | true	Controls whether the check_workspace command requires that the pin_equal and pin_opposite attributes on a cell must match between the base library and additional logic libraries.
lib.logic_model.require_same_opt_attrs	**false** | true	Controls whether the check_workspace command requires that the optimization attributes on a cell must match between the base library and additional logic libraries.
lib.logic_model.resolve_voltage_range_differences	**use_base_library** | require_same	Specifies how to handle voltage range differences across the logic libraries.
lib.logic_model.use_db_rail_names	**true** | false	When true, the commit_workspace command uses the voltage rail names from the logic libraries; otherwise, it generates the names.

Frame View Generation Application Options

The frame view generation (`lib.physical_model`) application options control the frame view generation, which is performed by the `check_workspace` command. Table 25 describes the `lib.physical_model` application options. If the application option has a data type of Boolean or enumerated string, the valid values are listed, with the default shown in purple; otherwise, the data type is specified along with the default.

Table 25 lib.physical_model Application Options

Application option	Valid values or data type	Description
`lib.physical_model.block_all`	**`auto`** | `false` | `true` | `used_layers` | *`top_layer`*	Controls whether the entire cell, except for the area around the pins, is treated as a blockage in the frame view. If you specify the *top_layer* argument, the tool creates a zero-spacing blockage on the specified layer and all layers below it; otherwise, the setting applies to all layers.
`lib.physical_model.block_core_margin`	list of layer-float pairs (default = “”)	Specifies the margin between the core blockage and the macro cell boundary for each layer in the frame view. If you specify this option for a layer, the tool creates a core blockage with the specified margin regardless of the setting of the `lib.physical_model.block_all` application option.
`lib.physical_model.block_hole_and_diagonal_of_pin`	list of layer names (default = "")	Specifies the layers to block holes and diagonal edges of pins.
`lib.physical_model.block_lower_layers_of_pins`	**`none`** | `via_blockage_upper` | `via_blockage_lower` | `via_blockage_both` | `metal_blockage`	Controls whether to block the lower layer pins of the ports from adjacent via layer or current metal layer.

Table 25 lib.physical_model Application Options (Continued)

Application option	Valid values or data type	Description
`lib.physical_model.block_via_layers_by_pins`	list of string (default = "") The format is `{{portName {metalLayerName mode1 mode2} …} …}` where, mode1 can be `upper | lower | both | none` and mode2 can be `inside_only | all`	Controls whether to block adjacent via layers of the pins by layer and modes. (mode1 controls which via layer to be blocked and mode2 controls the blocking region of the pin by the design boundary).
`lib.physical_model.color_based_dpt_flow`	**`auto`** `| true | false`	Controls whether frame generation uses the color-based flow. By default (`auto`), frame generation uses the color-based flow when the technology file contains double-patterning rules.
`lib.physical_model.connect_within_pin`	**`true`** `| false`	Controls whether the via enclosure must be within the pin shape. when dropping a via on the via region.
`lib.physical_model.convert_metal_blockage_to_zero_spacing`	list of layer-distance-mode triplets (default = “”)	Converts metal blockages in the design views to zero-spacing routing blockages in the for the specified layers. When creating the zero-spacing routing blockage, the tool increases the size of the metal blockage by the distance specified for that layer. By default, the specified distance applies to all metal blockages. To specify different distances based on whether the blockage touches a pin, specify the optional *mode* argument as either `touch_pin` or `no_touch_pin`.
`lib.physical_model.create_frame_for_subblocks`	**`false`** `| true`	Controls whether frame view generation creates a frame view only for the top-level block (the default) or for the top-level block and the subblocks.

Table 25 lib.physical_model Application Options (Continued)

Application option	Valid values or data type	Description
`lib.physical_model.create_zero_spacing_blockages_around_pins`	list of layer-width pairs (default = “”)	Specifies the width of the zero-spacing routing blockages created around the pins for each layer in the frame view. By default, frame view generation does not create zero-spacing routing blockages around the pins.
`lib.physical_model.derive_pattern_must_join`	**`false`** | `true` | `exclude_pg`	Controls whether frame view generation sets the `pattern_must_join` attribute on ports that have more than one terminal on a metal layer.
`lib.physical_model.design_rule_via_blockage_layers`	list of via layers (default = “”)	Converts zero-spacing routing blockages on the via layers of the frame view to design rule routing blockages.
`lib.physical_model.drc_distances`	list of layer-distance pairs (default = “”)	Specifies the distance from the boundary of blockages within with to preserve detailed shapes. By default, the tool uses information in the Layer section of the technology file to determine this value for each layer. The value is computed by adding the maximum spacing value to the maximum width threshold from the fat spacing table.
`lib.physical_model.enable_via_regions_for_all_design_types`	**`false`** | `true`	Controls whether frame view generation creates via regions for all design types. By default, frame view generation creates via regions only for standard cells.
`lib.physical_model.hierarchical`	**`true`** | `false`	Controls whether the tool extracts pins, vias, and blockages hierarchically when creating the frame view.
`lib.physical_model.include_nondefault_via`	list (default = “”)	Specifies the nondefault vias to include for via region extraction.
`lib.physical_model.include_routing_pg_ports`	list (default = “”)	Specifies the PG ports that require via regions and access edges in the frame view.

Table 25 lib.physical_model Application Options (Continued)

Application option	**Valid values or data type**	**Description**
`lib.physical_model.keepout_spacing_for_non_pin_shapes`	list of layer-spacings pairs (default = “”)	Defines how to create the zero-spacing routing blockages for the non-pin shapes on each affected layer.
`lib.physical_model.merge_metal_blockage`	**`false`** | `true`	Controls whether metal blockages are merged if they are too close, based on the minimum width and minimum spacing for the layer.
`lib.physical_model.pin_channel_distances`	list of layer-distance pairs (default = “”)	Controls the channel distance for each pin layer. If a pin is within the channel distance of the block boundary, the tool cuts a channel for the pin when generating the frame view.
`lib.physical_model.pin_must_connect_area_layers`	list of layer-spare layer pairs (default = “”)	Specifies the spare layer associated with each pin layer. The spare layer defines the must-area within the pin geometry. Use this option to restrict the via landing area on complex-shaped pins.
`lib.physical_model.pin_must_connect_area_thresholds`	list of layer-threshold pairs (default = “”)	Specifies the width threshold used to determine the must-connect area within the pin geometry for each connection layer. Use this option to restrict the via landing area on finger-shaped pins.
`lib.physical_model.port_contact_selections`	string (default = “”)	Specifies custom contact codes for specific ports for via region generation.
`lib.physical_model.preserve_metal_blockage`	**`auto`** | `true` | `false`	Controls whether metal blockages are trimmed when creating the frame view. By default, metal blockages are preserved (not trimmed) for the following types of cells: library cells, diode cells, end cap cells, fill cells, filler cells, physical-only cells, and well-tap cells. For all other types of cells, the tool trims metal blockages that touch a pin.
`lib.physical_model.read_fill_cells`	**`false`** | `true`	Controls whether to extract the shapes of the fill instance cells to frame view

Table 25 lib.physical_model Application Options (Continued)

Application option	Valid values or data type	Description
lib.physical_model.remove_non_pin_shapes	list of layer-mode pairs (default = “”)	Specifies whether to remove non-pin shapes when generating the frame view. Valid values for the mode are • none (the default) Frame generation does not remove non-pin shapes. • all Frame generation removes all non-pin shapes on the specified layer. • core Frame generation removes all non-pin shapes that are completely within the core on the specified layer. The core is determined by shrinking the block boundary by the DRC distance. Removing non-pin shapes reduces the number of blockages in the frame view. However, to ensure that the removal of the non-pin shapes from the frame view does not cause routing DRC violations, you should create a core blockage by using the lib.physical_model.block_all or lib.physical_model.block_core_margin application option.
lib.physical_model.source_drain_annotation	string (default = “”)	Specifies the name of the input file containing the source-drain annotation information for the cells in the library workspace.
lib.physical_model.trim_metal_blockage_around_pin	list of layer-method pairs (default = “”)	Specifies how to trim metal blockages around pins for each layer. Valid values for the method are touch, all, and none. If you specify this option for a layer, the tool uses the specified method to trim the metal blockages for that layer regardless of the setting of the lib.physical_model.preserve_metal_blockage application option.

Library Setting Application Options

The library setting (lib.setting) application options helps to set the library, before running the create_workspace command. Table 26 describes the lib.setting application options. If an application option has a data type of Boolean or enumerated string, the valid values are listed, with the default shown in purple; otherwise, the data type is specified along with the default.

Table 26 lib.setting Application Options

Application option	Valid values or data type	Description
lib.setting.enable_multithreaded_check_database	**false** | true	Enables multi threading on the check_database command.
lib.setting.handle_noncolored_shapes	**none** | loose | strict	Enables checking of non-colored shapes.
lib.setting.max_wait_time	**3600(sec)**, Type integer	Waits at-most the specified number of seconds before lock request timeout.
lib.setting.on_disk_operation	**false** | true	Saves the block while executing the copy_block or create_lib command.
lib.setting.show_internal_pins	**true** | false	Shows internal pin objects in the user interface.
lib.setting.use_tech_scale_factor	**true** | false	Specifies the technology length precision as the scale factor.

Antenna Extraction Application Options

The antenna extraction (signoff.antenna) application options control the IC Validator antenna extraction that occurs during frame view generation by the check_workspace command. Table 27 describes the signoff.antenna application options. If the application option has a data type of Boolean or enumerated string, the valid values are listed, with the default shown in purple; otherwise, the data type is specified along with the default.

Table 27 signoff.antenna Application Options

Application option	Valid values or data type	Description
`signoff.antenna.contact_layer`	list (default = “”)	Specifies the contact layers (contact layers are the layers that connect polysilicon to metal1). Specify the layers by using the layer names from the technology file.
`signoff.antenna.contact_layers_between_m0_diffusion`	list (default = “”)	Specifies the contact layers (contact layers are the layers that connect metal0 to diffusion). Specify the layers by using the layer names from the technology file.
`signoff.antenna.custom_runset_file`	string (default = “”)	Specifies the custom runset file for advanced gate layers.
`signoff.antenna.diffusion_layer`	list (default = “”)	Specifies the diffusion layers. Specify the layers by using the layer names from the technology file.
`signoff.antenna.enabled`	**`false`** | `true`	Controls whether the `check_workspace` command performs antenna extraction during frame view generation.
`signoff.antenna.exclude_non_mask_layers`	**`false`** | `true`	Controls whether nonmask layers are excluded from the antenna extraction flow.
`signoff.antenna.extract_via_antenna_property`	**`false`** | `true`	Controls whether antenna properties are extracted for via layers.
`signoff.antenna.gate_class1_layers`	list (default = “”)	Specifies the marking layers for gate thickness class 1. Specify the layers by using the layer names from the technology file.
`signoff.antenna.gate_class2_layers`	list (default = “”)	Specifies the marking layers for gate thickness class 2. Specify the layers by using the layer names from the technology file.
`signoff.antenna.gate_class3_layers`	list (default = “”)	Specifies the marking layers for gate thickness class 3. Specify the layers by using the layer names from the technology file.
`signoff.antenna.hierarchical_gate_class_include_file`	string (default = “”)	Specifies the gate class include file for advanced gate class marker layers.

Table 27 signoff.antenna Application Options (Continued)

Application option	Valid values or data type	Description
signoff.antenna.m0_layers_for_diffusion_connection	string (default = “”)	Specifies the m0 layers for diffusion connection. Specify the layers by using the layer names from the technology file.
signoff.antenna.m0_layers_for_poly_connection	string (default = “”)	Specifies the m0 layers for poly connection. Specify the layers by using the layer names from the technology file.
signoff.antenna.max_gate_class_name	string (default = “”)	Specifies the maximum gate class name when the gate class layers are passed through a custom runset.
signoff.antenna.poly_layer	list (default = “”)	Specifies the polysilicon layers. Specify the layers by using the layer names from the technology file.
signoff.antenna.report_diodes	**false** | true	Controls whether the tool reports the location of generated protection diodes.
signoff.antenna.top_cell_pin_only	**false** | true	Controls whether the tool extracts hierarchical antenna properties only for nets that connect to pins in the top-level of the hard macro.
signoff.antenna.treat_source_drain_as_diodes	**false** | true	Controls whether the tool treats the MOS source and drain as protection diodes and considers any MOS source and drain regions that are connected to a net to be a part of the diode protection area available to that net.
signoff.antenna.user_defined_ options	list (default = “”)	Specifies additional options for the IC Validator command line. The string that you specify in this option is added to the command line used to invoke antenna extraction in the IC Validator tool. The library manager does not perform any checking on the specified string.
signoff.antenna.v0_layers_between_m1_m0	list (default = “”)	Specifies the v0 layers (v0 layers are the layers that connect metal1 to metal0). Specify the layers by using the layer names from the technology file.

Command-Line Interface Application Options

The shell application options control various aspects of the library manager command-line interface. Table 28 describes the shell application options. If the application option has a data type of Boolean or enumerated string, the valid values are listed, with the default shown in purple; otherwise, the data type is specified along with the default.

Table 28 shell Application Options

Application option	Valid values or data type	Description
shell.common.collection_result_display_limit	integer (default = 100)	Specifies the maximum number of results displayed when the return value is a collection.
shell.common.enable_line_editing	**true** | false	Controls whether line editing mode is enabled.
shell.common.line_editing_mode	**emacs** | vi	Specifies the line editing mode.
shell.common.man_path	string	Specifies the search path for the man pages.
shell.common.report_default_significant_digits	integer default = 2)	Specifies the default number of significant digits used to display values in reports.
shell.common.single_line_messages	**false** | true	When set to true, error, warning, and information messages that contain newlines are output without newlines.
shell.common.tmp_dir_path	string (default = /tmp)	Specifies the directory that the tool uses for temporary storage.

Graphical User Interface Application Options

The `gui` application options control various aspects of the library manager GUI. Table 29 describes the `gui` application options. If the application option has a data type of Boolean or enumerated string, the valid values are listed, with the default shown in purple; otherwise, the data type is specified along with the default.

Table 29 gui Application Options

Application option	Valid values or data type	Description
`gui.auto_open_layout_windows`	**`true`** | `false`	Controls whether the GUI automatically creates a block window when the current design changes.
`gui.command_form_close_dialog_after_run_or_script`	**`false`** | `true`	Controls whether a shell command form invoked from a command search closes itself after the command is successfully executed or scripted.
`gui.command_form_log_options_with_default_values`	**`false`** | `true`	Controls whether a shell command form invoked from command search appends optional command options with default values to the command string for execution and logging.
`gui.command_form_show_result_dialog_after_run`	**`false`** | `true`	Controls whether a shell command form invoked from a command search displays the results from the command execution in a dialog box.
`gui.command_form_use_object_names_for_proc_options`	**`false`** | `true`	Controls the value format for Tcl procedure options declared as design objects.
`gui.custom_setup_files`	string (default = “”)	Specifies the files to source when invoking the GUI. These files are sourced in addition to the standard setup files.
`gui.enable_custom_setup_files`	**`false`** | `true`	Controls whether the tool sources the custom setup files specified in the `gui.custom_setup_files` application option.
`gui.file_editor_command`	string (default = “”)	Controls the command used to edit a file.
`gui.graphics_system`	string (default = `auto`)	Controls the graphics system used by the GUI.

Table 29 gui Application Options (Continued)

Application option	Valid values or data type	Description
`gui.layer_solid_fill_pattern`	string (default = `Dense4Pattern`)	Controls default display of solid filled layers. To see the valid values for this application option, use the `get_app_option_value -details -name gui.layer_solid_fill_pattern` command.
`gui.max_cores`	integer (default = 0)	Specifies the maximum number of CPU cores used by the GUI.
`gui.name_based_hier_delimiter`	string (default = /)	Specifies the hierarchy delimiter used by the name-based hierarchy browser.
`gui.url_browser_command`	string (default = “”)	Specifies the command used to display a URL in a web browser.

B

Working With the Library Manager GUI

The library manager GUI provides a variety of tools for viewing and editing the cells in your library.

The following topics provide an overview of these tools:

- Starting the Tool in the GUI
- Opening the GUI
- Closing the GUI
- Exiting the Tool From the GUI
- Using the Library Preparation Wizard
- Working in the Main Window
- Using the Library Browser
- Using the Message Browser Window

Starting the Tool in the GUI

To start the lm_shell tool in the GUI,

1. Set the DISPLAY environment variable to the name of your Linux system display.
2. Include the path to the bin directory in your $PATH variable.
3. Enter the `lm_shell` command with the `-gui` option in a Linux shell.

   ```
   % lm_shell -gui
   ```

   You can include other options when you start the GUI. For example,

   - `-f` *`script_file_name`* to execute a script
   - `-x` *`command`* to execute an lm_shell command

When you start the library manager in the GUI, it

1. Runs the commands in the .synopsys_lm.setup and .synopsys_lm_gui/setup.tcl setup files

   The tool first runs the setup files in your home directory followed by the files in the project directory (the current working directory in which you start the tool).

2. Runs the command specified by the `-x` option or the script specified by the `-f` option.
3. Opens a main window that displays the library preparation wizard

   By default, the console appears attached (docked) to the main window above the status bar.

See Also

- Starting the Command-Line Interface

Opening the GUI

If you start lm_shell without the GUI, you can open the GUI at any time by entering the `gui_start` command at the lm_shell> prompt.

```
lm_shell> gui_start
```

If you want to execute a script before opening the GUI, use the `-file` option. For example,

```
lm_shell> gui_start -file my_gui_setup.tcl
```

Note:

The GUI uses the DISPLAY environment variable to determine the display location. Before opening the GUI, ensure that this variable is set to the name of your Linux system display.

When you open the library manager GUI, it

1. Runs the commands in the .synopsys_lm_gui/setup.tcl setup files

   The tool first runs the setup files in your home directory followed by the files in the project directory (the current working directory in which you start the tool).

2. Runs the script specified by the `-file` option.

3. Opens a main window

   - If a library workspace or cell library is open, it opens the library browser in the main window.

     The library browser is a hierarchical representation of the library workspace or cell library.

   - If a library workspace or cell library is not open, it opens the library preparation wizard in the main window.

Closing the GUI

You can open or close the GUI at any time during the session. For example, if you need to conserve system resources, you can close the GUI and continue the session inlm_shell.

To close the GUI, do one of the following:

- Choose File > Close GUI in the block window.
- Choose Window > Close All Windows (Close GUI) in any GUI window.
- Enter the `gui_stop` command.

```
lm_shell> gui_stop
```

Exiting the Tool From the GUI

To exit the library manager from the GUI, enter the `exit` or `quit` command or choose File > Exit.

Note:

The library manager does not save the view settings when you exit. For information about saving changed view settings, see the Setting and Saving View Properties.

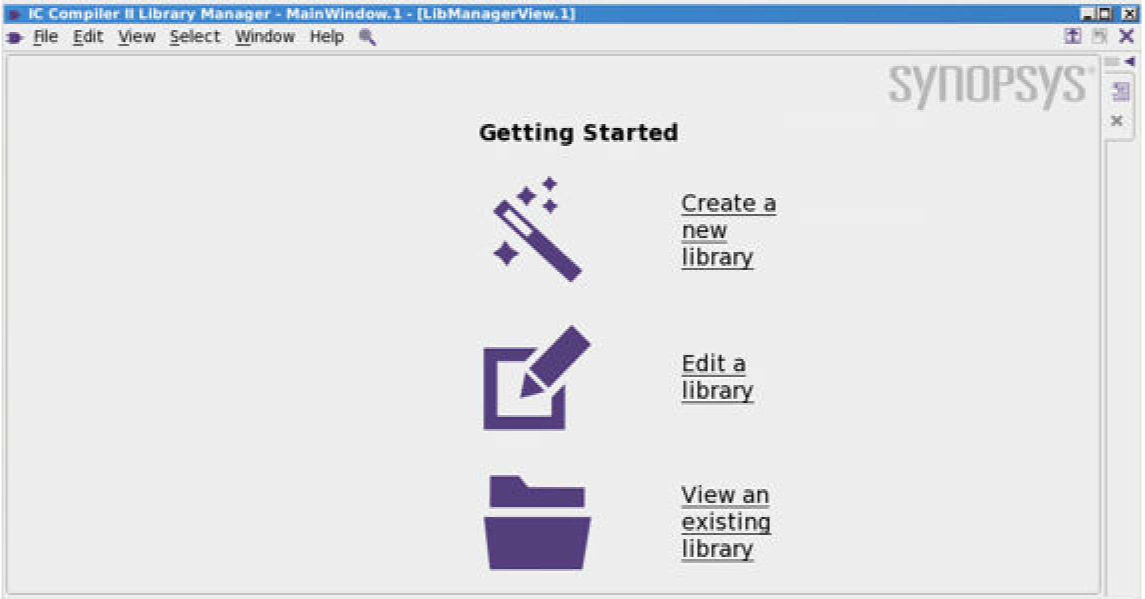
See Also

- Exiting the Library Manager Tool

Using the Library Preparation Wizard

The library preparation wizard is an easy-to-use tool for navigating the library preparation flow and executing the individual tasks to create and modify your cell libraries. Figure 26 shows the start page of the library preparation wizard.

Figure 26 Library Preparation Wizard



IC Compiler II Library Manager - MainWindow.1 - [LibManagerView.1]
File Edit View Select Window Help
SYNOPSYS
Getting Started
Create a new library
Edit a library
View an existing library

The following topics describe how to use the wizard to perform these tasks:

- Creating a Cell Library
- Modifying a Cell Library
- Viewing a Cell Library

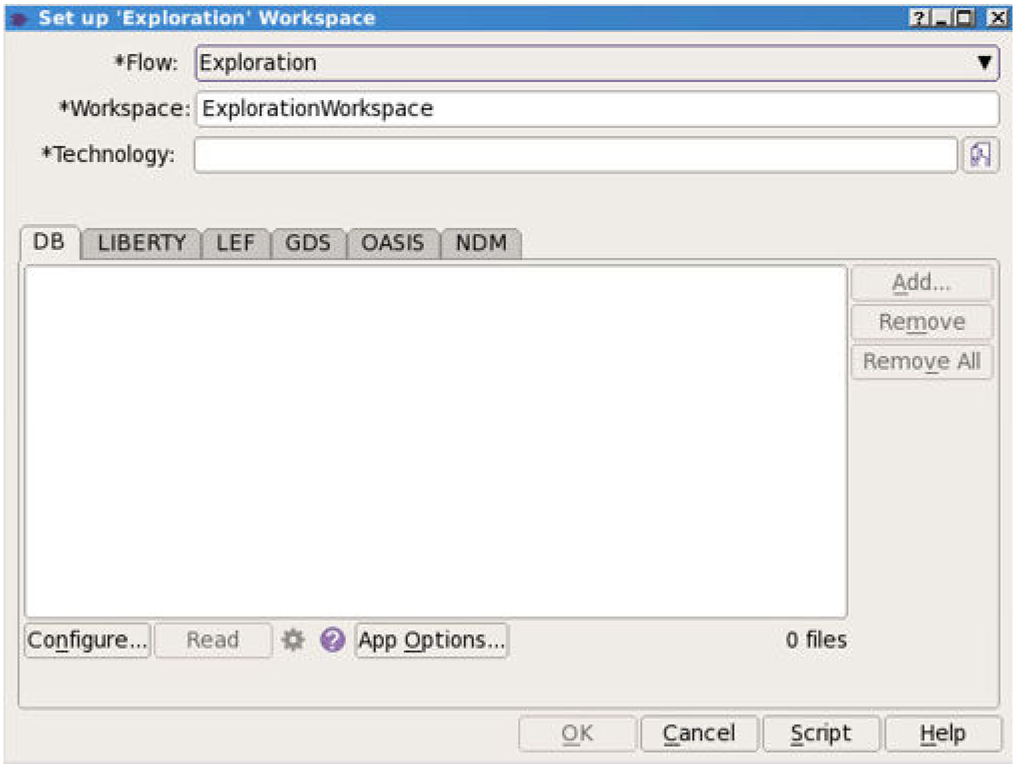
Creating a Cell Library

To use the library preparation wizard to create a new cell library,

1. Click the create icon ( ) or the “Create a new library” link.

   The “Set up ‘Exploration’ Workspace” dialog box (Figure 27) appears.

Figure 27 Set up “Exploration” Workspace Dialog Box



*Flow: Exploration

*Workspace: ExplorationWorkspace

*Technology:

0 files

2. Select the library preparation flow in the Flow text box.

   By default, the exploration flow is selected. To use another flow, select the flow from the drop-down list.

3. Specify the workspace name in the Workspace text box.

4. Specify the technology data source.

   You can specify either a technology file, which must have a file extension of .tf, or a technology library, which must have a file extension of .ndm. You can either enter the file path or click to browse for it.

5. Read the library source files.

   To read the source files,

   a. Click the tab associated with the file type (DB for the logic libraries; LEF, GDS, or NDM for the physical libraries) and click Add, which opens the Open dialog box.

   b. Browse to the directory that contains the library source files, select the files, and click Open. To select multiple files, hold the Ctrl key.

      When you open library source files, they appear in the “Set up ‘Exploration’ Workspace” dialog box.

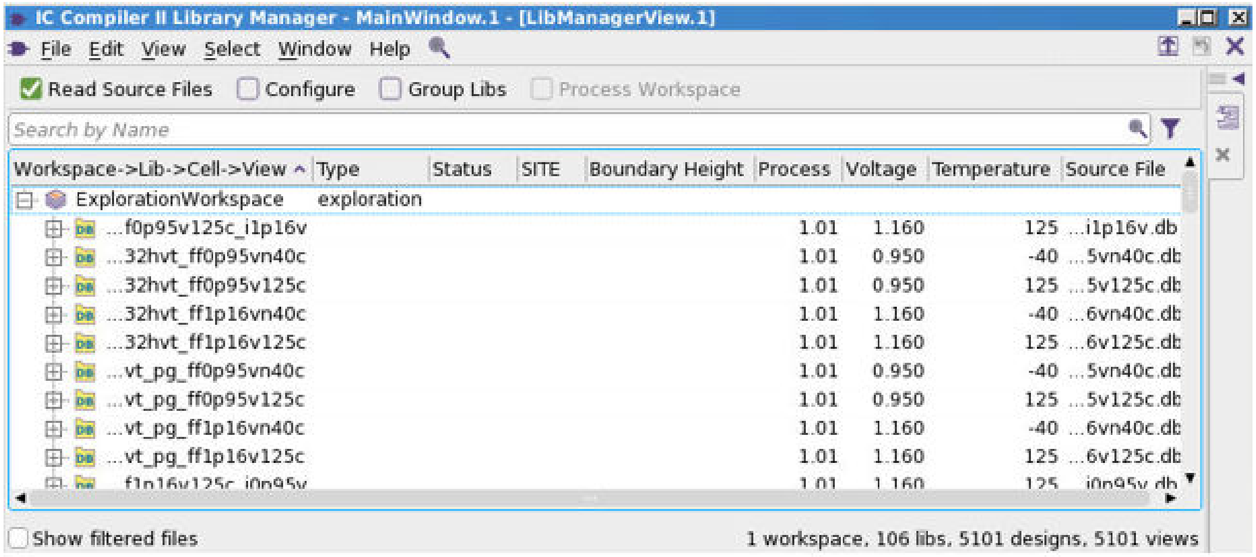
   c. Click Read to read the source files listed in the dialog box.

   For details about reading library source files, see Loading the Logic Data and Loading the Physical Data.

6. Click OK to finish the workspace setup.

   The library browser is updated to display the libraries loaded into the library workspace, as shown in Figure 28.

Figure 28 Library Browser After Loading Source Libraries


IC Compiler II Library Manager - MainWindow.1 - [LibManagerView.1]
File
Edit
View
Select
Window
Help
Search by Name

☑ Read Source Files □ Configure □ Group Libs □ Process Workspace

Workspace->Lib->Cell->View	Type	Status	SITE	Boundary Height	Process	Voltage	Temperature	Source File
ExplorationWorkspace	exploration							
...f0p95v125c_i1p16v					1.01	1.160	125	...i1p16v.db
...32hvt_ff0p95vn40c					1.01	0.950	-40	...5vn40c.db
...32hvt_ff0p95v125c					1.01	0.950	125	...5v125c.db
...32hvt_ff1p16vn40c					1.01	1.160	-40	...6vn40c.db
...32hvt_ff1p16v125c					1.01	1.160	125	...6v125c.db
...vt_pg_ff0p95vn40c					1.01	0.950	-40	...5vn40c.db
...vt_pg_ff0p95v125c					1.01	0.950	125	...5v125c.db
...vt_pg_ff1p16vn40c					1.01	1.160	-40	...6vn40c.db
...vt_pg_ff1p16v125c					1.01	1.160	125	...6v125c.db
...f1p16v125c_i0p95v					1.01	1.160	125	...i0p95v.db

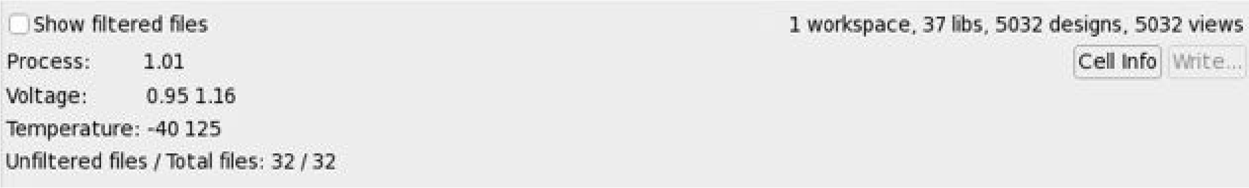
□ Show filtered files

1 workspace, 106 libs, 5101 designs, 5101 views

7. Configure the PVT filtering.

   The library manager displays the current PVT configuration, as well as the filtering statistics, as shown in Figure 29.

Figure 29 PVT Configuration



☐ Show filtered files

1 workspace, 37 libs, 5032 designs, 5032 views

Process: 1.01

Voltage: 0.95 1.16

Temperature: -40 125

Unfiltered files / Total files: 32 / 32

   To modify the PVT configuration,

   a. Select Configure.

      The Filter Process/Voltage/Temperature dialog box appears. The dialog box displays the settings available in the logic libraries loaded in step Step 5.

   b. Select the process, voltage, and temperature settings and click OK.

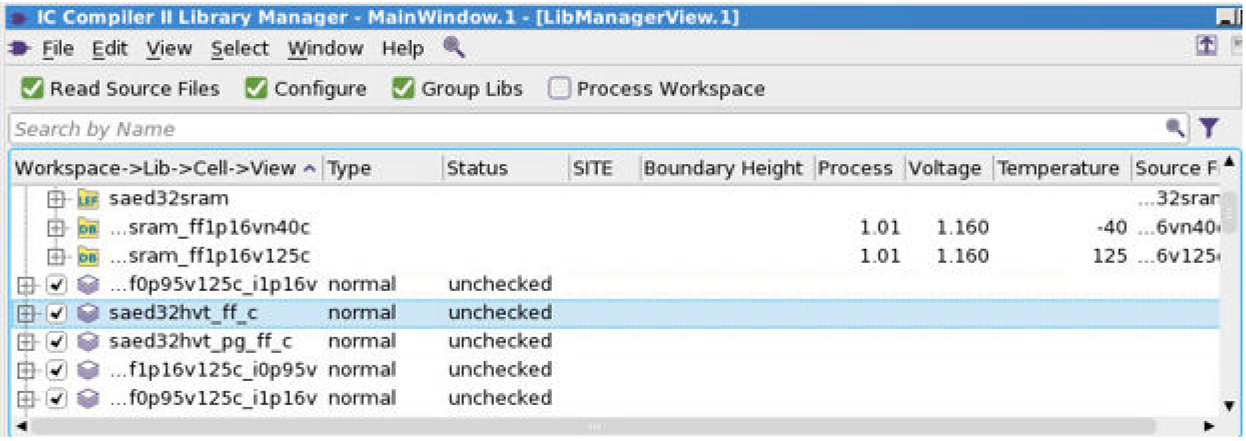
      By default, the filtered logic libraries are not shown in the library browser. To display the filtered logic libraries, select “Show filtered files.” When you select this option, the library browser displays the filtered files in gray italic text.

   For details about configuring PVT filtering, see Filtering Logic Libraries Based on Operating Corners.

8. Create the subworkspaces by selecting Group Libs and then clicking OK in the Group Libraries dialog box.

   The library browser is updated to display the library groups created by the library preparation wizard, as shown in Figure 30.

Figure 30 Library Browser After Grouping Libraries



IC Compiler II Library Manager - MainWindow.1 - [LibManagerView.1]

File Edit View Select Window Help

☑ Read Source Files ☑ Configure ☑ Group Libs ☐ Process Workspace

Search by Name

Workspace->Lib->Cell->View	Type	Status	SITE	Boundary Height	Process	Voltage	Temperature	Source F
saed32sram								...32sran
...sram_ff1p16vn40c					1.01	1.160	-40	...6vn40
...sram_ff1p16v125c					1.01	1.160	125	...6v125
☑ ...f0p95v125c_i1p16v	normal	unchecked						
☑ saed32hvt_ff_c	normal	unchecked						
☑ saed32hvt_pg_ff_c	normal	unchecked						
☑ ...f1p16v125c_i0p95v	normal	unchecked						
☑ ...f0p95v125c_i1p16v	normal	unchecked						

9. Validate the workspace by selecting Process Workspace and then clicking OK in the Process Workspace dialog box.

   The workspace must pass validation before you can create a cell library. For details about workspace validation, see Validating the Workspace.

10. Create the cell library by selecting Commit Workspace and then clicking OK in the Commit Workspace dialog box.

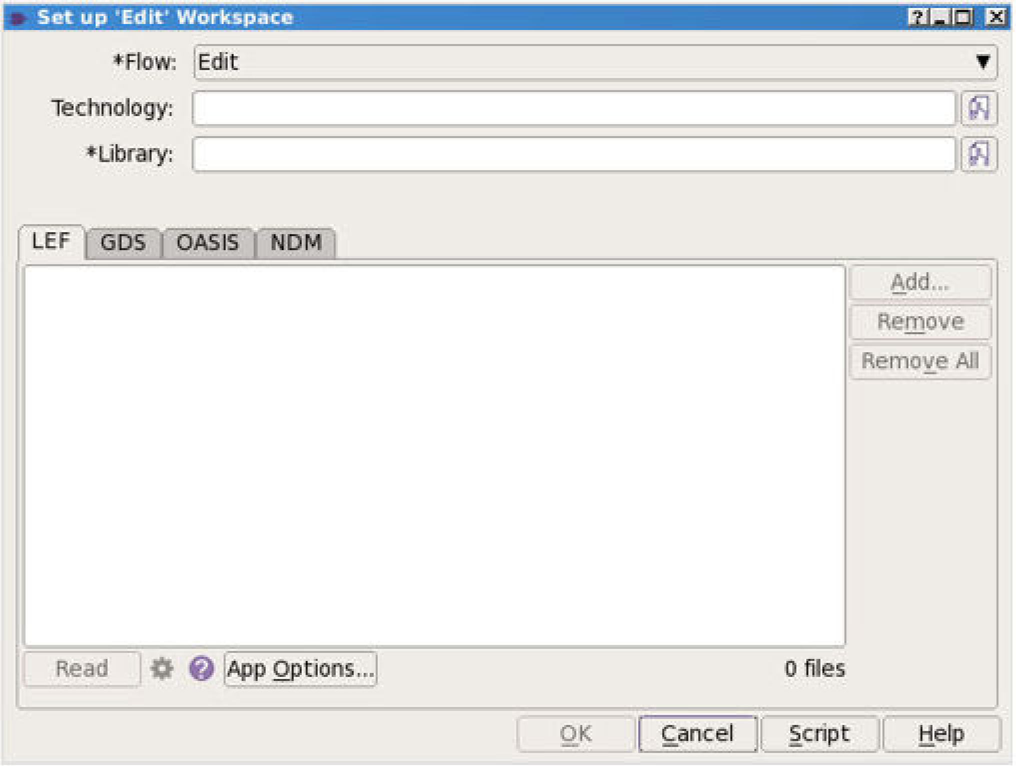
Modifying a Cell Library

To use the library preparation wizard to modify an existing cell library,

1. Click the edit icon ( ) or the “Edit a library” link.

   The “Set up ‘Edit’ Workspace” dialog box (Figure 31) appears.

Figure 31 Set up “Edit” Workspace Dialog Box



Set up 'Edit' Workspace

*Flow: Edit

2. Specify the library in the Library text box.

   You can either enter the file path or click to browse for it.

3. If you want to update the technology data for the cell library, specify the technology data source in the Technology text box.

   You can specify either a technology file, which must have a file extension of .tf, or a technology library, which must have a file extension of .ndm. You can either enter the file path or click to browse for it.

4. Read the physical library source files.

   To read the physical library source files,

   a. Click the tab associated with the file type you want to read (LEF, GDS, or NDM) and click Add, which opens the Open dialog box.

   b. Browse to the directory that contains the library source files, select the files, and click Open. To select multiple files, hold the Ctrl key.

      When you open library source files, they appear in the “Set up ‘Edit’ Workspace” dialog box.

   c. Click Read to read the source files listed in the dialog box.

   For details about reading physical library source files into an edit workspace, see Modifying a Cell Library.

5. Validate the workspace by selecting Check Workspace in the library browser and then clicking OK in the Check Workspace dialog box.

   The workspace must pass validation before you can create a cell library. For details about workspace validation, see Validating the Workspace.

6. Save the updated cell library by selecting Commit Workspace and then clicking OK in the Commit Workspace dialog box.

Viewing a Cell Library

You can use the GUI to list the cells in a cell library, view information about the cells, and view the layout of the cells.

To use the library preparation wizard to view an existing cell library,

1. Click the folder icon (📁) or the “View an existing library” link.

   The Open Library dialog box appears.

2. Select the cell library.

   You can either enter the path to the cell library or click 📂 to browse for it.

3. If you want to edit the cell library, select “Open reference library for edit.”

4. Click OK.

   The library browser opens with the selected cell library.

Feedback

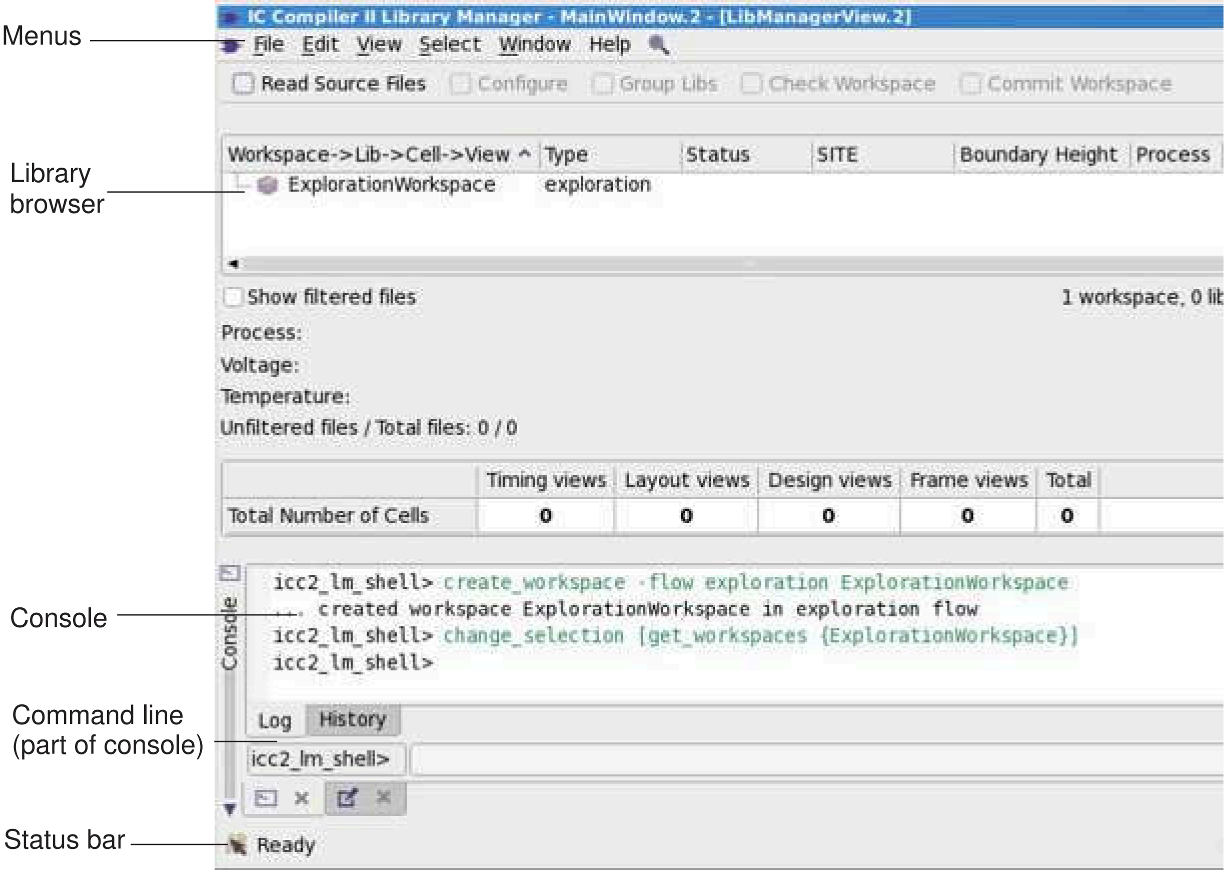
See Also

- Using the Library Browser

Working in the Main Window

When you start the library manager tool and open the GUI, the main window appears. Figure 32 identifies the major features of the main window.

Figure 32 Main Window Features


Menus
Library browser
Console
Command line (part of console)
Status bar
IC Compiler II Library Manager - MainWindow.2 - [LibManagerView.2]
File Edit View Select Window Help
Read Source Files
Configure
Group Libs
Check Workspace
Commit Workspace
Console
Log
History
Ready

Workspace->Lib->Cell->View	Type	Status	SITE	Boundary Height	Process
ExplorationWorkspace	exploration				

□ Show filtered files 1 workspace, 0 lib

Process:
Voltage:
Temperature:
Unfiltered files / Total files: 0 / 0

	Timing views	Layout views	Design views	Frame views	Total
Total Number of Cells	0	0	0	0	0

```
icc2_lm_shell> create_workspace -flow exploration ExplorationWorkspace
... created workspace ExplorationWorkspace in exploration flow
icc2_lm_shell> change_selection [get_workspaces {ExplorationWorkspace}]
icc2_lm_shell>
```

```
icc2_lm_shell>
```

Typically, you use the main window to

- Run Tcl scripts
- Enter lm_shelll commands, monitor command processing, and view messages in the console
- Open and close libraries and blocks (library cells or designs)

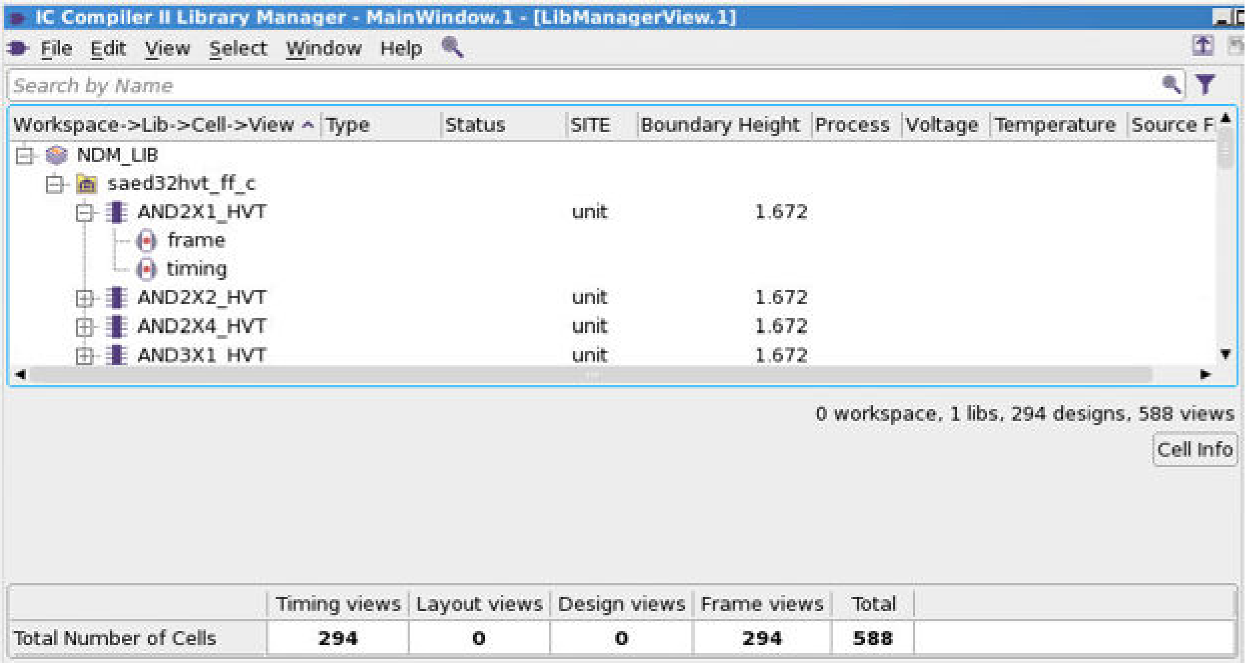
- Explore cell libraries in the library browser
- Set application options
- Open other GUI windows
- Close the GUI or exit the tool

Using the Library Browser

You can use the library browser to view the cells in the source logic and physical libraries in a library workspace or the cells in a cell library.

The library browser shows a hierarchical view of the libraries loaded in memory. If a library workspace is open, it shows its source libraries; otherwise, it shows the open cell libraries. You can expand each library to show the blocks (library cells or designs) in the library. You can expand each block to show the views available for the block. The library browser also reports the number and types of cell views in the library. Figure 33 shows a library browser that displays an expanded cell library with one of its cells expanded to show its views.

Figure 33 Library Browser


IC Compiler II Library Manager - MainWindow.1 - [LibManagerView.1]
File
Edit
View
Select
Window
Help
Search by Name
Cell Info

Workspace->Lib->Cell->View ^	Type	Status	SITE	Boundary Height	Process	Voltage	Temperature	Source F
NDM_LIB								
saed32hvt_ff_c								
AND2X1_HVT			unit	1.672				
frame								
timing								
AND2X2_HVT			unit	1.672				
AND2X4_HVT			unit	1.672				
AND3X1_HVT			unit	1.672				

0 workspace, 1 libs, 294 designs, 588 views

	Timing views	Layout views	Design views	Frame views	Total
Total Number of Cells	294	0	0	294	588

To sort the blocks displayed in the library browser, click the caret (^) in the browser header.

To filter the blocks, enter a string in the search box. As you type, the library browser updates to show only blocks that contain the text you entered. Click the X to the right side of the text box to clear the filter.

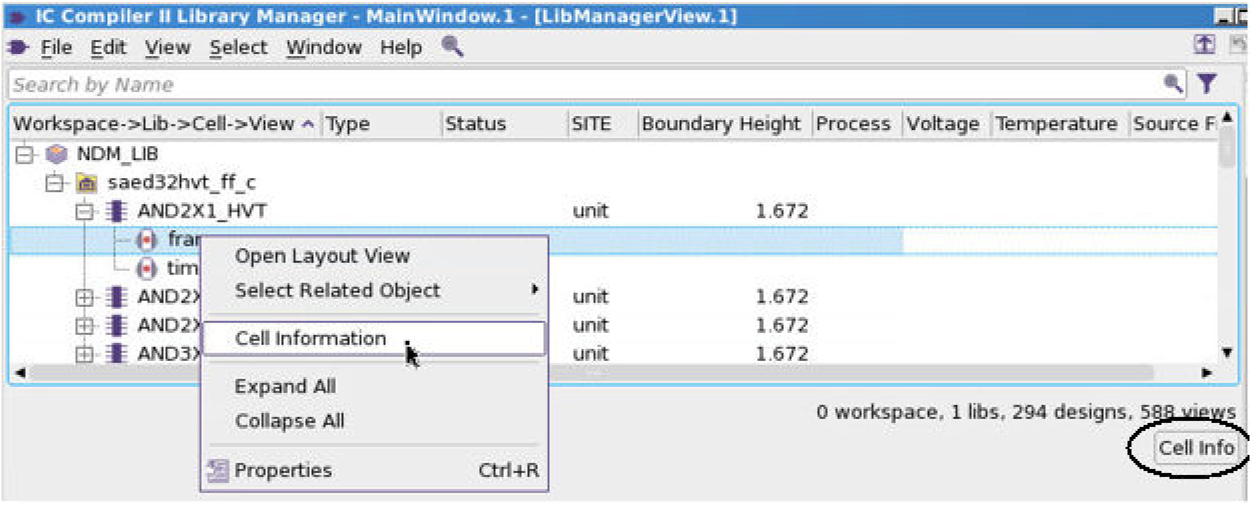
To open a block in a block window to view or modify its layout, select a physical view (frame, design, or layout) of the block, right-click, and choose Open Layout View. For information about using the block window, see Working in the Block Window.

To see detailed information about the library cells, display the library browser cell information.

- To display information for a specific cell, select one of its views and choose Cell Information from the context-sensitive menu.
- To display information for all cells in the library, click the “Cell info” button, which is located at the bottom right of the library browser.

Figure 34 shows these features in the library browser. Figure 35 shows an example cell information report.

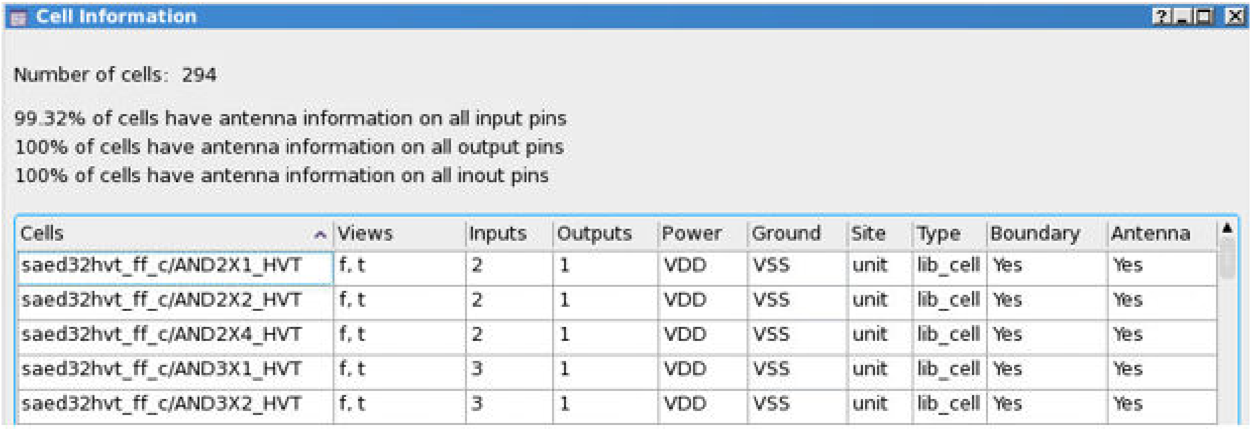
Figure 34 Displaying Cell Information in the Library Browser



Workspace->Lib->Cell->View	Type	Status	SITE	Boundary Height	Process	Voltage	Temperature	Source F
NDM_LIB								
saed32hvt_ff_c								
AND2X1_HVT			unit	1.672				
fra								
tim								
AND2X			unit	1.672				
AND2X			unit	1.672				
AND3X			unit	1.672				

0 workspace, 1 libs, 294 designs, 588 views

Figure 35 Cell Information Report



Number of cells: 294

99.32% of cells have antenna information on all input pins
100% of cells have antenna information on all output pins
100% of cells have antenna information on all inout pins

Cells	Views	Inputs	Outputs	Power	Ground	Site	Type	Boundary	Antenna
saed32hvt_ff_c/AND2X1_HVT	f, t	2	1	VDD	VSS	unit	lib_cell	Yes	Yes
saed32hvt_ff_c/AND2X2_HVT	f, t	2	1	VDD	VSS	unit	lib_cell	Yes	Yes
saed32hvt_ff_c/AND2X4_HVT	f, t	2	1	VDD	VSS	unit	lib_cell	Yes	Yes
saed32hvt_ff_c/AND3X1_HVT	f, t	3	1	VDD	VSS	unit	lib_cell	Yes	Yes
saed32hvt_ff_c/AND3X2_HVT	f, t	3	1	VDD	VSS	unit	lib_cell	Yes	Yes

Using the Message Browser Window

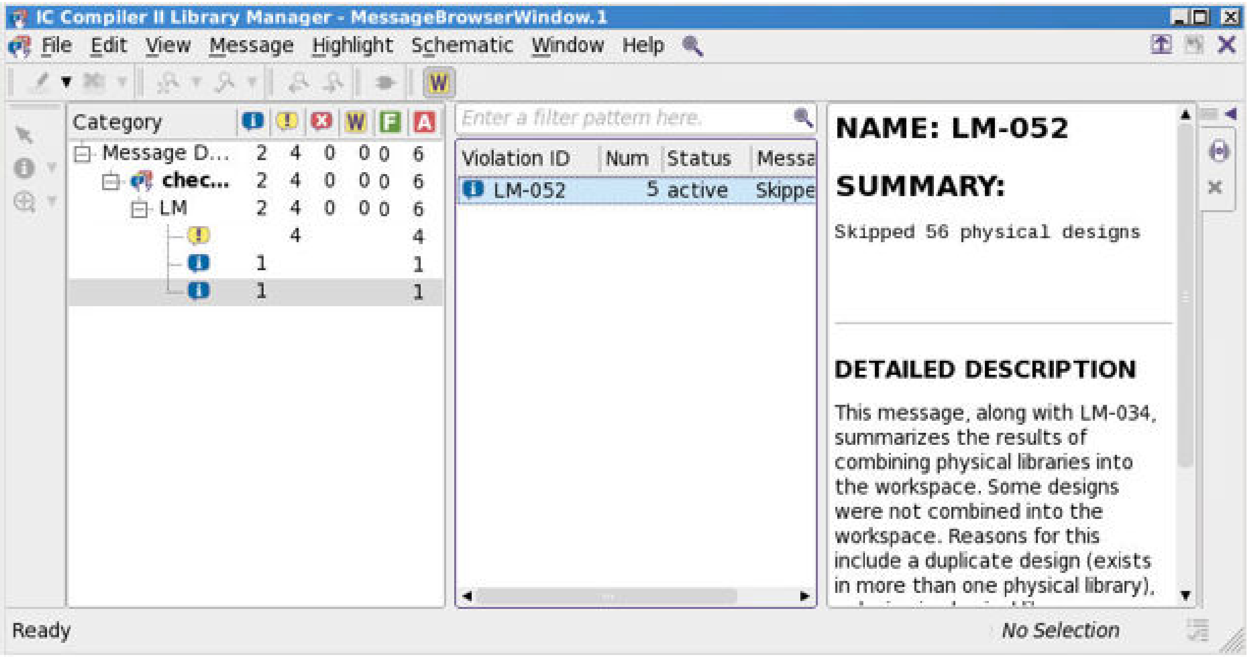
After you run the `check_workspace` or `process_workspaces` command, you can view the messages in the message browser. To view the messages,

To view the messages,

1. Open the message browser window by choosing Window > Message Browser Window.
2. Load the message database by choosing File > Open Message Database, and then selecting the EMS file.

The message browser lists each of the messages (information, warning, and error), as well as the message status (W:waived, F:fixed, and A:active). To display the man page for a message, select the message in the browser view. Figure 36 shows an example of a message browser window.

Figure 36 Message Browser


IC Compiler II Library Manager - MessageBrowserWindow.1
File
Edit
View
Message
Highlight
Schematic
Window
Help
W
Enter a filter pattern here.
Ready
No Selection

Category	i	!	X	W	F	A
Message D...	2	4	0	0	0	6
chec...	2	4	0	0	0	6
LM	2	4	0	0	0	6
!		4				4
i	1					1
i	1					1

Violation ID	Num	Status	Messa
LM-052	5	active	Skippe

NAME: LM-052

SUMMARY:

Skipped 56 physical designs

DETAILED DESCRIPTION

This message, along with LM-034, summarizes the results of combining physical libraries into the workspace. Some designs were not combined into the workspace. Reasons for this include a duplicate design (exists in more than one physical library),

See Also

- Using the Error Browser

C

Using Non-PG Logic Libraries

This topic describes the following workspace checks when using logic libraries that do not use the PG-pin syntax:

- PG Rail Checks
- PG Pin Connections

PG Rail Checks

For logic libraries that do not use the PG-pin syntax, the power rails and pins are derived from the library-level `power_supply` group and cell-level `rail_connection` attributes. The `power_rail` attributes in the `power_supply` group define the power rail names. The `rail_connection` attributes on a cell associate these power rails with the cell's power pins. The signal pins of a cell are associated with the PG rails by using the pin-level `input_signal_level` and `output_signal_level` attributes. Example 12 shows the use of these attributes to define the power connections in a .lib logic library file.

Example 12   Library Example With power_supply Group

```
library (my_lib) {
  ...
 power_supply () {
   default_power_rail : VDD085;
   power_rail (VDD085, 0.85);
   power_rail (VDD105, 1.05);
   power_rail (VDD120, 1.20);
  }
  ...
  cell (LS_085_105) {
   is_level_shifter : true;
   rail_connection (VDDH, VDD105) ;
   rail_connection (VDDL, VDD085) ;
   ...
    pin (in) {
      direction : input;
      input_signal_level : VDDL;
      ...
    }
    pin (out) {
      direction : output;
      output_signal_level : VDDH;
      ...
    }
    ...
  } /* end cell group*/
  ...
} /* end library group*/
```

The tool verifies that each logic library has the same number of rails and they have the same names; however, if a rail is defined in a power_supply group but is not used, the tool ignores mismatches involving that rail and does not include the rail definition in the generated cell library.

If the rail order differs, the tool automatically updates the order using the information from the base library.

By default, if a rail name differs between the logic libraries, the tool issues an LM-043 error. If the mismatch is caused only by rail names and positions, and not rail type differences, you can use the rename_rail command to make the rail names match. For example, assume logic library A has a power_rail setting of (VDD1.0, 1.0) and logic library B has a power_rail setting of (VDD1.1, 1.1). You could use the rename_rail command to align these rails, as shown in the following example:

```
lm_shell> rename_rail -library A -from VDD1.0 -to VDD
lm_shell> rename_rail -library B -from VDD1.1 -to VDD
```

You can also use the rename_rail command to correct the situation where the rail_connection attribute for a cell uses the same PG pin name across libraries, but the rail name differs.

PG Pin Connections

If the rail name specified for a rail_connection attribute is not defined in the library-level power_supply group, the tool issues an error message. To fix this error, use the set_attribute command to change the rail_connection attribute to a valid rail name. To see the rail names defined for a library workspace, use the report_workspace -panes command.

D

Running Library Compiler in Fusion Compiler or IC Compiler II Environment

You can run the Library Compiler tool within the Fusion Compiler and IC Compiler II environments. You can start a Library Compiler session in the background, run Library Compiler commands to query, analyze, and edit logic and physical library data, and get the results in the ongoing Fusion Compiler or IC Compiler II session. The Fusion Compiler and IC Compiler II command log files now also include the log files of the Library Compiler sessions. This is useful in co-optimizing design and libraries.

To run the complete set of features available in the Library Compiler tool, you must have a Library Compiler license.

The following topics describe how to run the Library Compiler tool within the Fusion Compiler or IC Compiler II environment:

- Configuring lc_shell Session Options
- Verifying the Status of lc_shell Session
- Executing Library Compiler Commands
- Closing the lc_shell Session

Note:

> Running Library Compiler tool applies to both the Fusion Compiler and the IC Compiler II tools. For simplicity, command usage examples in these topics are shown only with the `fc_shell` prompt.

Configuring lc_shell Session Options

To enable running the Library Compiler tool within the Fusion Compiler environment, use the new `set_lc_options` Tcl command in `fc_shell` or `icc2_shell`.

```
fc_shell> set_lc_options
```

By default, the command uses the current directory as the working directory to store the `lc_shell` session files and logs, and the `SYNOPSYS_LC_ROOT` or `synopsys_root` variable to find the full path to the `lc_shell` executable.

You can also specify the `lc_shell` executable path and the working directory using the `-exec_path` and `-work_dir` options with the following command in the following format:

```
fc_shell> set_lc_options -exec_path lc_shell_path \
                         -work_dir directory_name
```

lc_shell_path is the full path to the `lc_shell` executable and *directory_name* is the working directory where the `lc_shell` session and log files the tool generates.

The `set_lc_options` command also reports the specified `lc_shell` options, as shown in the following example:

```
fc_shell> set_lc_options -exec_path /my_bin/lc_shell \
                         -work_dir ./my_work_dir
Library Compiler options
------------------------------------------------------------
Library Compiler executable  : /my_bin/lc_shell
Work directory               : ./my_work_dir
1
fc_shell>
```

To reset using the `-reset` option,

- For all the existing `lc_shell` options, use the `-reset` option as shown in the following example:

  ```
  fc_shell> set_lc_options -reset
  Library Compiler options
  ------------------------------------------------------------
  Library Compiler executable :
  Work directory :
  1
  fc_shell>
  ```

- For only the executable path, use the `-reset -exec_path` option.
- For only the working directory, use the `-reset -work_dir` option.

If you have an active `lc_shell` session running in the background, you must close it before reconfiguring the `lc_shell` options. Otherwise, the tool issues an error message as shown in the following examples. For more information about closing an active `lc_shell` session, see Closing the lc_shell Session.

```
fc_shell> set_lc_options -exec_path ../new_lcsh_path/lc_shell
Error: please run "lc_sh -exit" to close current lc_shell session
before setting new lc_shell options.
0
fc_shell> set_lc_options -work_dir ./my_new_dir
Error: please run "lc_sh -exit" to close current lc_shell session
before setting new lc_shell options.
0
```

```
fc_shell> set_lc_options -reset
Error: please run "lc_sh -exit" to close current lc_shell session
before setting new lc_shell options.
0
fc_shell>
```

Verifying the Status of lc_shell Session

To verify that the `lc_shell` session is running in the background, use the new `report_lc_status` command as shown in the following examples.

Example 13 Status Reported When lc_shell Runs in Background

```
fc_shell> report_lc_status
lc_shell session is running in the background.

Library Compiler options
------------------------------------------------------------
Library Compiler executable    : /global/apps/lc_2019.03-SP5/bin/lc_shell
Work directory               : ./my_work_dir
1
fc_shell>
```

Example 14 Status Reported When lc_shell is not Running

```
fc_shell> report_lc_status
No lc_shell session is running in the background.
1
fc_shell>
```

Executing Library Compiler Commands

After configuring the `lc_shell` session options, use the new `lc_sh` command to run Library Compiler commands using the `lc_shell` session running in the background, in the following format:

```
fc_shell> lc_sh arguments
```

`arguments` are `lc_shell` commands to be executed in the `lc_shell` session.

The `lc_sh` command starts an `lc_shell` session only if no active `lc_shell` session is running in the background. If no `lc_shell` session is running in the background, the `lc_sh` command creates a session and then executes the `lc_shell` commands:

```
fc_shell> lc_sh read_lib ../test.lib

                             Library Compiler (TM)

 .......
```

```
Initializing...
lc_shell> read_lib ../my_lib.lib
Reading '../my_lib.lib' ...
Technology library 'my_lib' read successfully
1
fc_shell>
```

If the `lc_shell` session is already running in the background, the `lc_sh` command arguments run automatically in the current `lc_shell`.

```
fc_shell> lc_sh write_lib test -o .../test.db
lc_shell> write_lib test -o ../test.db
Wrote the 'test' library to '../test.db' successfully
1
fc_shell>
```

To execute multiple `lc_shell` commands in a single line, enclose the `lc_shell` commands in braces and separate each command by a semicolon (;), as shown in the following example:

```
fc_shell> lc_sh {read_lib ../test.lib; write_lib test -o .../test.db}
lc_shell> read_lib ../test.lib
Reading '../test.lib' ...
Technology library 'test' read successfully
1
lc_shell> write_lib test -o ../test.db
Wrote the 'test' library to '../test.db' successfully
1
fc_shell>
```

To source a Tcl command script for `lc_shell`, use the `lc_shell source` command as shown in the following example:

```
fc_shell> lc_sh source lc.tcl
lc_shell> source lc.tcl
Reading '../test.lib' ...
Technology library 'test' read successfully
1
Wrote the 'test' library to '../test.db' successfully
1
fc_shell>
```

Closing the lc_shell Session

To close the current `lc_shell` session, use the `lc_sh exit` or `lc_sh quit` command, or press Ctrl+C. If you exit from fc_shell or icc2_shell while `lc_shell` is running in the background, you automatically exit `lc_shell`.

```
fc_shell> lc_sh exit
lc_shell> exit
Maximum memory usage for this session: 72.59 MB
Maximum memory usage for this session including child processes: 72.59 MB
...

Thank you...
fc_shell>
```