Copyright Notice for the Command-Line Editing Feature
© 1992, 1993 The Regents of the University of California. All rights reserved. This code is derived from software contributed to Berkeley by Christos Zoulas of Cornell University.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software must display the following acknowledgement:

This product includes software developed by the University of California, Berkeley and its contributors.

4. Neither the name of the University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Copyright Notice for the Line-Editing Library
© 1992 Simmule Turner and Rich Salz. All rights reserved.

This software is not subject to any license of the American Telephone and Telegraph Company or of the Regents of the University of California.

Permission is granted to anyone to use this software for any purpose on any computer system, and to alter it and redistribute it freely, subject to the following restrictions:
1. The authors are not responsible for the consequences of use of this software, no matter how awful, even if they arise from flaws in it.
2. The origin of this software must not be misrepresented, either by explicit claim or by omission. Since few users ever read sources, credits must appear in the documentation.
3. Altered versions must be plainly marked as such, and must not be misrepresented as being the original software. Since few users ever read sources, credits must appear in the documentation.
4. This notice may not be removed or altered.
# Contents

About This Guide ........................................................................................................... xii
Customer Support ........................................................................................................... xiv

1. Working With the IC Compiler II Library Manager Tool

Library Preparation Terminology .................................................................................. 1-2
  Reference Library ........................................................................................................ 1-2
  Technology Library ..................................................................................................... 1-4
  Frame-Only Library ..................................................................................................... 1-5
  Aggregate Library ....................................................................................................... 1-5
  Library Workspace ....................................................................................................... 1-5
  Pane ............................................................................................................................. 1-6
  Breakpoints ................................................................................................................ 1-6
  Scaling Group ............................................................................................................. 1-6
  Antenna Properties ..................................................................................................... 1-7

Library Preparation Flow ............................................................................................... 1-8

User Interfaces .............................................................................................................. 1-9
  Starting the Command-Line Interface ...................................................................... 1-10
  Exiting the IC Compiler II Library Manager Tool ..................................................... 1-11

Entering icc2_lm_shell Commands ............................................................................... 1-11
  Interrupting or Terminating Command Processing .................................................. 1-12
  Getting Information About Commands ..................................................................... 1-12
    Displaying Command Help ....................................................................................... 1-12

Using Application Options ............................................................................................ 1-13
Using Variables .......................................................... 1-13
Viewing Man Pages ......................................................... 1-14
Using Tcl Scripts .......................................................... 1-15
Setup Files ................................................................. 1-16
Using the Command Log File ............................................. 1-16

2. Preparing Reference Libraries

Library Preparation Flows ................................................ 2-2
Analyzing the Library Source Files ..................................... 2-5
Building Reference Libraries ............................................. 2-7
  Defining the Search Path ............................................. 2-8
  Creating a Library Workspace ..................................... 2-8
Specifying the Library Preparation Flow ............................... 2-10
Specifying the Length Precision ...................................... 2-12
Loading the Technology Data .......................................... 2-12
Completing the Technology Data ...................................... 2-13
  Preparing Site Definitions ....................................... 2-14
  Preparing Routing Layers ........................................ 2-15
Loading the Logic Libraries ............................................. 2-15
  Identifying the Process Associated With a Logic Library .... 2-17
  Filtering Logic Libraries Based on Operating Corners ........ 2-17
Loading Extracted Timing Models ...................................... 2-18
Loading the Physical Libraries ........................................ 2-19
  Reading LEF Files ................................................ 2-19
  Reading GDSII or OASIS Files ..................................... 2-23
  Loading Physical Data from a Reference Library ............... 2-31
  Loading Physical Data from a Design Library .................. 2-32
  Importing Milkyway FRAM Views From the IC Compiler Tool .. 2-33
Specifying Which Blocks to Process .................................. 2-34
Handling Duplicate Blocks in the Physical Libraries .............. 2-34
Block Mapping File Syntax ............................................ 2-35
Creating Scaling Groups ............................................... 2-37
Loading Parasitic Parameters .......................................... 2-38
Loading Signal Electromigration Constraints ........................ 2-39
Completing the Physical Cell Information ............................ 2-39
  Preparing I/O Cells ............................................... 2-40
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preparing Standard Cells</td>
<td>2-40</td>
</tr>
<tr>
<td>Preparing Macros</td>
<td>2-41</td>
</tr>
<tr>
<td>Querying the Workspace</td>
<td>2-42</td>
</tr>
<tr>
<td>Validating the Workspace</td>
<td>2-43</td>
</tr>
<tr>
<td>Library Checks</td>
<td>2-44</td>
</tr>
<tr>
<td>PG Rail Checks</td>
<td>2-44</td>
</tr>
<tr>
<td>Cell Checks</td>
<td>2-46</td>
</tr>
<tr>
<td>Pin Checks</td>
<td>2-48</td>
</tr>
<tr>
<td>Timing Arc Checks</td>
<td>2-52</td>
</tr>
<tr>
<td>Scaling Group Checks</td>
<td>2-54</td>
</tr>
<tr>
<td>Leakage-Power Checks</td>
<td>2-54</td>
</tr>
<tr>
<td>Allowing Incomplete or Inconsistent Library Data</td>
<td>2-55</td>
</tr>
<tr>
<td>Creating a Mismatch Configuration</td>
<td>2-58</td>
</tr>
<tr>
<td>Setting a Mismatch Configuration</td>
<td>2-58</td>
</tr>
<tr>
<td>Reporting Mismatch Configurations</td>
<td>2-59</td>
</tr>
<tr>
<td>Generating Frame Views</td>
<td>2-59</td>
</tr>
<tr>
<td>Identifying Secondary PG Pins</td>
<td>2-60</td>
</tr>
<tr>
<td>Committing the Workspace</td>
<td>2-61</td>
</tr>
<tr>
<td>Using the Exploration Flow</td>
<td>2-62</td>
</tr>
<tr>
<td>Enabling Scaling Group Creation</td>
<td>2-63</td>
</tr>
<tr>
<td>Performing Automated Library Analysis</td>
<td>2-64</td>
</tr>
<tr>
<td>Using a Library Preparation Script</td>
<td>2-65</td>
</tr>
<tr>
<td>Verifying Reference Libraries</td>
<td>2-67</td>
</tr>
<tr>
<td>Verifying Logical Information in a Reference Library</td>
<td>2-67</td>
</tr>
<tr>
<td>Verifying the Frame Views in a Reference Library</td>
<td>2-68</td>
</tr>
<tr>
<td>Modifying a Reference Library</td>
<td>2-69</td>
</tr>
<tr>
<td>Defining Antenna Properties on Standard Cells</td>
<td>2-70</td>
</tr>
<tr>
<td>Antenna Properties for Standard Cells</td>
<td>2-71</td>
</tr>
<tr>
<td>Defining Antenna Properties on Hard Macro Cells</td>
<td>2-72</td>
</tr>
<tr>
<td>Antenna Properties for Hard Macro Cells</td>
<td>2-73</td>
</tr>
<tr>
<td>Creating an Aggregate Library</td>
<td>2-75</td>
</tr>
<tr>
<td>Setting the Search Order</td>
<td>2-75</td>
</tr>
<tr>
<td>Modifying an Aggregate Library</td>
<td>2-76</td>
</tr>
<tr>
<td>Opening a Reference Library</td>
<td>2-77</td>
</tr>
<tr>
<td>Getting Information About Reference Libraries</td>
<td>2-77</td>
</tr>
<tr>
<td>Querying Libraries</td>
<td>2-78</td>
</tr>
</tbody>
</table>
Reporting Libraries ................................................. 2-78
Reporting the Frame Properties ................................. 2-81
Querying Library Objects ..................................... 2-82
Reporting Library Objects ................................... 2-83

Exporting Reference Library Content ...................... 2-86
Closing a Reference Library ................................ 2-86

Appendix A. Library Manager Application Options

Library Manager Application Options ...................... A-2
LEF Input Application Options .............................. A-5
GDSII and OASIS Input Application Options ............. A-6
Timing View Generation Application Options .............. A-8
Frame View Generation Application Options ............. A-9
Antenna Extraction Application Options .................. A-14
Command-Line Interface Application Options ............ A-17
Graphical User Interface Application Options .......... A-18

Appendix B. Working With the Library Manager GUI

Starting the Tool in the GUI .................................. B-2
Opening the GUI ................................................. B-3
Closing the GUI ................................................ B-4
Exiting the Tool From the GUI ................................. B-4

Using the Library Preparation Wizard ...................... B-5
Creating a Reference Library ................................ B-6
Modifying a Reference Library ............................. B-10
Viewing a Reference Library ............................... B-11

Working in the Main Window ................................. B-12
Using the Library Browser .................................. B-13
Using the Message Browser Window ..................... B-15
Appendix C. Using Non-PG Logic Libraries

- PG Rail Checks ................................................................. C-2
- PG Pin Connections ...................................................... C-3
Preface

This preface includes the following topics:

• About This Guide
• Customer Support
About This Guide

The IC Compiler II Library Manager tool merges input from a variety of sources to create a single, unified reference library that contains all of the physical and logical information for a technology and its library cells. Reference libraries created by the IC Compiler II Library Manager tool are used by the IC Compiler II tool to implement designs.

Audience

This user guide is for engineers who use the IC Compiler II Library Manager tool to prepare libraries for use with the IC Compiler II tool.

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

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

Related Publications

For additional information about the IC Compiler II Library Manager tool, see the documentation on the Synopsys SolvNet® online support site at the following address:

https://solvnet.synopsys.com/DocsOnWeb

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

- Library Compiler™
- IC Compiler™ II

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://solvnet.synopsys.com/DownloadCenter
2. Select IC Compiler II, and then select a release in the list that appears.
## Conventions

The following conventions are used in Synopsys documentation.

<table>
<thead>
<tr>
<th>Convention</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Courier</td>
<td>Indicates syntax, such as <code>write_file</code>.</td>
</tr>
<tr>
<td>Courier italic</td>
<td>Indicates a user-defined value in syntax, such as <code>write_file design_list</code>.</td>
</tr>
<tr>
<td>Courier bold</td>
<td>Indicates user input—text you type verbatim—in examples, such as <code>prompt&gt; write_file top</code></td>
</tr>
</tbody>
</table>
| []                  | 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` |
| Control-c           | Indicates a keyboard combination, such as holding down the Control key and pressing c. |
| \                  | Indicates a continuation of a command line.                                  |
| /                   | Indicates levels of directory structure.                                     |
| Edit > Copy         | Indicates a path to a menu command, such as opening the Edit menu and choosing Copy. |
Customer Support

Customer support is available through SolvNet online customer support and through contacting the Synopsys Technical Support Center.

Accessing SolvNet

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

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

https://solvnet.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 SolvNet site, click HELP in the top-right menu bar.

Contacting the Synopsys Technical Support Center

If you have problems, questions, or suggestions, you can contact the Synopsys Technical Support Center in the following ways:

- Open a support case to your local support center online by signing in to the SolvNet site at https://solvnet.synopsys.com, clicking Support, and then clicking “Open A Support Case.”

- Send an e-mail message to your local support center.
  - E-mail support_center@synopsys.com from within North America.
  - Find other local support center e-mail addresses at http://www.synopsys.com/Support/GlobalSupportCenters/Pages

- Telephone your local support center.
  - Call (800) 245-8005 from within North America.
  - Find other local support center telephone numbers at http://www.synopsys.com/Support/GlobalSupportCenters/Pages
You use the IC Compiler II Library Manager tool to build a reference library for use with the IC Compiler II implementation tool. To learn how to use the library manager, see the following topics:

- Library Preparation Terminology
- Library Preparation Flow
- User Interfaces
- Entering icc2_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:

- Reference Library
- Technology Library
- Frame-Only Library
- Aggregate Library
- Library Workspace
- Pane
- Breakpoints
- Scaling Group
- Antenna Properties

Reference Library

Reference libraries for the IC Compiler II tool use the new data model (NDM) format. These reference libraries are unified libraries that contain 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. A library cell can have the following views:

- Timing view
  The timing view contains the timing, power, and functional information for a cell. It can contain timing and power information for multiple characterization points (voltage and temperature) for a given process setting. The data for each characterization point is stored in a pane.
  The IC Compiler II tool uses the timing view to perform static timing analysis, power analysis, and optimization.

- Layout view
  The layout view contains the physical shape information for a cell, not including connectivity and pins.
  The library manager uses the layout view to create the design view from GDSII or Open Artwork System Interchange Standard (OASIS) source libraries, and the IC Compiler II tool uses it for mask generation.
- Design view
  The design view contains 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
  The frame view contains the information needed to perform placement and routing, such as the block boundary, pins, via regions, and routing blockages.
  The IC Compiler II tool uses the frame view to perform placement and routing.

Figure 1-1 shows a conceptual view of a reference library. The views shown in green are the views used by the IC Compiler II tool.

Figure 1-1 Conceptual View of a Reference Library

You create a reference library by loading the logical and physical models from the following sources:

- The technology file
  The technology file contains information about the routing layers and routing rules for a specific technology.
  The IC Compiler II tool uses the same technology file as the IC Compiler tool. 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 reference libraries in NDM format, 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 reference library or in the design library. For information about storing TLUPlus information in a design library, see the *IC Compiler II Implementation User Guide*.

A reference library contains all of the required information, both logical and physical, for a set of cells for a technology. The IC Compiler II tool does not directly use the source files used to generate a reference library.

See Also
• Preparing Reference Libraries

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

See Also
• Reference Library
Frame-Only Library
A frame-only library is an intermediate library that contains only physical information (frame, design, and layout views) for the library cells. This type of reference library is used only as an input to generate a reference library; it cannot be used outside of the library manager.

See Also
• Reference Library

Aggregate Library
An aggregate library combines multiple reference libraries into a single reference library with a defined search order for the included reference 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 reference libraries for easier maintenance and distribution.

See Also
• Creating an Aggregate Library

Library Workspace
A library workspace is the in-memory representation of a reference library while it is being built. You use a library workspace to create a new reference library, modify an existing reference library, or create an aggregate reference library. You load the source files into the library workspace, validate the contents, and then commit the workspace to save it as a reference 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 reference library, you work only with a root workspace.
• Subworkspaces
  Subworkspaces are used in the exploration flow to partition source data into multiple reference libraries. Each subworkspace generates a single reference 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 IC Compiler II tool. In addition, they enable interpolation between the voltage and temperature values of the characterization points, which enables efficient multivoltage analysis and optimization.

The characterization points form a complete grid when data exists for each temperature and voltage combination. Figure 1-2 shows an example of a complete grid and an incomplete grid. The figure on the left represents a complete grid that consists of four logic libraries for a process. The figure on the right represents an incomplete grid that consists of three logic
libraries for a process. The grid is incomplete because the characterization point at 1.0V and 0°C is missing.

Figure 1-2  Grid Representation of Characterization Points

See Also
• Creating 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
Library Preparation Flow

Figure 1-3 shows the library preparation flow used to build a reference library.

For detailed information about the library preparation flow, see “Preparing Reference Libraries.”
User Interfaces

The IC Compiler II Library Manager tool operates in the X windows environment on UNIX or 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, icc2_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 icc2_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 icc2_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 IC Compiler II Library Manager Tool

See Also

- Working With the Library Manager GUI
Starting the Command-Line Interface

To start the icc2_lm_shell command-line interface,

1. Include the path to the bin directory in your $PATH variable.
2. Enter the `icc2_lm_shell` command in a UNIX or Linux shell.
   ```shell
   % icc2_lm_shell
   ```

   You can include other options when you start icc2_lm_shell. For example,
   - `-file script_file_name` to execute a script
   - `-x command` to execute an icc2_lm_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 icc2_lm_shell)

At startup, icc2_lm_shell performs the following tasks:

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 icc2_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 IC Compiler II Library Manager Tool

To exit the IC Compiler II 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 icc2_lm_shell Commands

You interact with the library manager by using icc2_lm_shell commands, which are based on the tool command language (Tcl) and include certain command extensions needed to implement specific IC Compiler II Library Manager functionality. The library manager command language provides capabilities similar to UNIX command shells, including variables, conditional execution of commands, and control flow commands. You can

• Enter individual commands interactively at the icc2_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 icc2_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 icc2_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 icc2_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 icc2_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, icc2_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 icc2_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 IC Compiler II library manager:

• Command help

• Command man pages

Displaying Command Help

Command help consists of either a brief description of an icc2_lm_shell command or a list of the options and arguments supported by an icc2_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:

  icc2_lm_shell> help read_db

• To display the options supported by an icc2_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:

  icc2_lm_shell> read_db -help
Using Application Options

The IC Compiler II 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 the IC Compiler II Data Model User Guide.

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 icc2_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:

```
icc2_lm_shell> man read_db
```

To display the man page for an icc2_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 that 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:

  ```
  icc2_lm_shell> man lm_options
  ```

- **Subcategory summaries**

  To view a man page that 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:

  ```
  icc2_lm_shell> man signoff.antenna_options
  ```

- **Command summaries**

  To view a man page that 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:

  ```
  icc2_lm_shell> man read_db_options
  ```

If you enter the `man` command on the icc2_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.

See Also

- *IC Compiler II Graphical User Interface User Guide*
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 icc2_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 `icc2_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.

See Also

- Starting the Command-Line Interface
- Using Tcl With Synopsys Tools
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_icc2_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_icc2_lm.setup file in your home directory
   Typically the settings in this file define your working environment.

2. The .synopsys_icc2_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 icc2_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 icc2_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_icc2_lm.setup file. If your user-defined or project-specific setup file does not contain this variable, the tool automatically creates the icc2_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
Preparing Reference Libraries

To learn how to prepare reference libraries for the IC Compiler II tool, see the following topics:

- Library Preparation Flows
- Analyzing the Library Source Files
- Building Reference Libraries
- Using the Exploration Flow
- Verifying Reference Libraries
- Modifying a Reference Library
- Creating an Aggregate Library
- Modifying an Aggregate Library
- Opening a Reference Library
- Getting Information About Reference Libraries
- Exporting Reference Library Content
- Closing a Reference Library
Library Preparation Flows

The library manager supports the following library preparation flows:

- **Normal flow**
  
  Use this flow to create a reference 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 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 reference library contains the cells that exist in both the logic and physical library 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 reference library that contains only the information from the technology file and TLUPlus files. This type of reference library is referred to as a technology library.

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

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

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

- **Organize-by-PVT flow**
  
  Use this flow to create a reference library if you have multiple logic library files, each of which contains a single cell, and one or more physical library files. Each cell might exist in multiple logic library files, where each logic library file contains data for a different characterization point (process, voltage, and temperature). This flow results in fewer generated references libraries. Library vendors typically use this library organization for macro cells.

  By default, the generated reference library contains the cells that exist in both the logic and physical library files.

  **Note:**

  This flow is deprecated and will be removed in a future release. Instead of using the organize-by-PVT flow, use the exploration flow with one of the following application
option settings: `lib.workspace.group_libs_macro_grouping_strategy` set to `aggregate_single_cell` or `lib.workspace.explore_create_aggregate set to true`.

- **Extracted timing model (ETM) flow**
  Use this flow to create a reference 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 file that contains the design.

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

- **Physical-only flow**
  Use this flow to create a separate physical-only reference library for the cells that exist only in a physical library 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 files; however, the generated reference library includes only those cells that exist only in the physical library 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 reference library. To build a library of cells using only physical libraries, use the frame-only flow.

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

- **Edit flow**
  Use this flow to modify the physical data for cells in an existing reference library.

- **Aggregate flow**
  Use this flow to combine multiple separate reference libraries into a single reference library.

  Figure 2-1 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 2-1  Normal and Physical-Only Example

Figure 2-2 shows the results of using the organize-by-PVT flow on a library workspace that contains four .db files, each of which contains a single cell, and one LEF file.

Figure 2-2  Organize-by-PVT Example
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 reference 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 reference libraries using various library preparation flows. When you build multiple reference libraries, you need to determine which source files to use for each reference 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.

- 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 reference library. Ideally, each cell is included in exactly one reference library; otherwise, users must be very careful when specifying the reference 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 reference library for these physical-only cells.
The following are some examples of groups of input files that might be used to create a reference library:

- Use the normal flow to create a reference library from six .db files that each have the same set of library cells but represent different process corners spanning two voltages and three temperatures and a LEF file that includes the library cells in the .db files
- Use the organize-by-PVT flow to create a reference library from two .db files for a specific RAM at two different voltage-temperature points and a LEF file that includes the RAM

See Also

- Reference Library
Building Reference Libraries

The basic process for building a reference 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. For detailed information about a task, click on the task in the figure.

Figure 2-3  Overview of the Library Preparation Process

- **Define the search path**
  
  set_app_var_search_path

- **Create a workspace**
  
  create_workspace

- **Load the technology data**
  
  read_tech_file, read_ndm

- **Load the logic libraries**
  
  read_db

- **Load the physical libraries**
  
  read_lef, read_gds, read_oasis, read_ndm

- **Create scaling groups**
  
  create_scaling_group

- **Load the parasitic parameters**
  
  read_parasitic_tech

- **Load the signal electromigration constraints**
  
  read_signal_em_constraints

- **Validate the workspace**
  
  check_workspace

- **Commit the workspace**
  
  commit_workspace
To build a technology library, you follow the same flow, but omit steps four through seven, which involve the source files for the library cells.

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

See Also

• Using the Exploration Flow
• Modifying a Reference 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 reference 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 name of an existing reference library, including its `.ndm` file extension.

• 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 reference 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 reference library named myreflib, use the following command to create the library workspace:

```
icc2_lm_shell> create_workspace myreflib
```

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 length precision for the library**
  
  By default, one database unit represents 1 Angstrom, which corresponds to a scale factor of 10000 database units per micron. The grid size (pitch) for a technology must divide evenly into the scale factor. If the grid size for your technology does not divide into 10000, specify a different scale factor by using the `-scale_factor` option.

  Note:
  
  The tool does not use the length precision specified in the technology file.

  For more information, see Specifying the Length Precision.

**See Also**

- Library Workspace
- Analyzing the Library Source Files
- Using the Exploration Flow
- Modifying a Reference 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 2-1 shows the options to use for the supported library preparation flows.

<table>
<thead>
<tr>
<th>Library preparation flow</th>
<th>create_workspace -flow keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference library creation flows</td>
<td></td>
</tr>
<tr>
<td>Normal</td>
<td><code>normal</code></td>
</tr>
<tr>
<td>Technology-only</td>
<td><code>normal</code></td>
</tr>
<tr>
<td>Frame-only</td>
<td><code>frame</code></td>
</tr>
<tr>
<td>Organize-by-PVT</td>
<td><code>organize_by_pvt</code></td>
</tr>
<tr>
<td>Extracted timing model</td>
<td><code>etm_moded</code></td>
</tr>
<tr>
<td>Physical-only</td>
<td><code>physical_only</code></td>
</tr>
<tr>
<td>Exploration</td>
<td><code>exploration</code></td>
</tr>
<tr>
<td>Reference library verification flow</td>
<td></td>
</tr>
<tr>
<td>Verification</td>
<td><code>verification</code></td>
</tr>
<tr>
<td>Reference library modification flow</td>
<td></td>
</tr>
<tr>
<td>Edit</td>
<td><code>edit</code></td>
</tr>
<tr>
<td>Aggregate library creation flow</td>
<td></td>
</tr>
<tr>
<td>Aggregate</td>
<td><code>aggregate</code></td>
</tr>
</tbody>
</table>
For example, to use the normal flow to create a reference library named myreflib, use the following command:

```
icc2_lm_shell> create_workspace myreflib
```

To use the organize-by-PVT flow to create a reference library named myreflib_pvt, use the following command:

```
icc2_lm_shell> create_workspace -flow organize_by_pvt myreflib_pvt
```

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

```
icc2_lm_shell> create_workspace -flow physical_only myreflib_po
```

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

```
icc2_lm_shell> create_workspace -flow exploration myreflib_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 reference library, use the `remove_workspace` command.

**See Also**
- Library Workspace
- Analyzing the Library Source Files
- Using the Exploration Flow
- Verifying Reference Libraries
- Modifying a Reference Library
- Creating an Aggregate Library
Specifying the Length Precision

IC Compiler II reference libraries use a scale factor to determine the length precision. The scale factor specifies the number of database units per micron. The length precision for a library workspace must be a multiple of both the length precision of the technology associated with the workspace and the length precision of the physical libraries loaded into the workspace.

By default, IC Compiler II reference libraries use a scale factor of 10000 (one database unit represents 1 Angstrom). To specify a different scale factor, use the -scale_factor option with the create_workspace command.

Loading the Technology Data

You can load the technology data for a reference 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 reference library named myreflib that gets its technology data from the my.tf technology file, use the following command:

  icc2_lm_shell> create_workspace -technology my.tf myreflib

- 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:

  icc2_lm_shell> read_tech_file my.tf

- 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 reference library named myreflib that gets its technology data from the mytech.ndm technology library, use the following command:

  icc2_lm_shell> create_workspace -use_technology_lib mytech.ndm \ myreflib
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:

```
icc2_lm_shell> read_ndm mytech.ndm
```

When you load technology data into an existing library workspace,

- The new technology data overwrites any existing technology data
- The library manager verifies that the frame views in the library workspace are consistent with the new technology data

Important:
- The technology file does not contain all of the technology data required to perform placement and routing in the IC Compiler II tool. For information about completing the technology data, see Completing the Technology Data.

See Also

- Creating a Library Workspace
- “Technology Data Access” in the IC Compiler II Data Model User Guide

---

**Completing the Technology Data**

The technology file does not contain all of the data required by the IC Compiler II tool for placement and routing. The following topics describe how to add the missing technology data to a reference 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 information 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:
  ```shell
  icc2_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:
  ```shell
  icc2_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:
  ```shell
  icc2_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 information 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 reference library contains only single-height site definitions. The IC Compiler II tool uses 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 and vertical. For example, to set the preferred routing direction for the M2 layer as horizontal, use the following command:

  
  ```
  icc2_lm_shell> set_attribute [get_layers M2] \n  routing_direction horizontal
  ```

• **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:

  ```
  icc2_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 information 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 the Logic Libraries

The logic libraries contain the timing, power, and functionality information for the standard cells and macro cells in the reference library. Logic libraries have the following characteristics:

• Each library represents a specific characterization point (process, voltage, and temperature).

  Note: For information about the process setting for a logic library, see Identifying the Process Associated With a Logic Library.

• The timing model format is generic CMOS, nonlinear delay model (NLDM), or Composite Current Source (CCS).

Important: 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:

```
icc2_lm_shell> read_db db1_pvt1.db
```

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

```
icc2_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 reference 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

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 \texttt{-process\_label} option specified with the \texttt{read\_db} command
    - The \texttt{process\_label} attribute defined in the \texttt{operating\_conditions} group for the default operating conditions in the logic library
  
- The process number
  - The library manager determines the process number from the \texttt{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 \texttt{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:

\begin{verbatim}
icc2_lm_shell> read_db -process\_label PVT1 db1.db
\end{verbatim}

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

\begin{verbatim}
icc2_lm_shell> set_process -libraries {db1} -label PVT2
\end{verbatim}

Filtering Logic Libraries Based on Operating Corners

By default, the generated reference 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 reference library.

To define the PVT configuration for a reference library, use the \texttt{set\_pvt\_configuration} command. If you run the \texttt{set\_pvt\_configuration} command multiple times, the settings are additive. To ensure that you are starting without any predefined filters, use the \texttt{-clear\_filter all} option when you define a new PVT configuration.
For example, to set a PVT configuration that allows any process setting, voltages of 0.65 or 1.32 volts, and a temperature of 25 degrees, use the following command:

```
icc2_lm_shell> set_pvt_configuration -clear_filter all \
  -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 reference 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:

```
icc2_lm_shell> read_db -mode_label functional modell.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 Libraries

The physical libraries contain 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 libraries can be in LEF, GDSII, or OASIS format. In addition, you can load the physical libraries from a previously saved workspace or reference 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 a Reference Library
- Importing Milkyway FRAM Views From the IC Compiler Tool

In some cases, the technology file and physical libraries do not provide all of the information required by the IC Compiler II implementation tool. For information about the required information and how to specify it, if it is not loaded automatically, see the following topics:

- Completing the Technology Data
- Completing the Physical Cell Information

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:

```
icc2_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 Blocks in the Physical Libraries.

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.

• Fails if the LEF file contains layer or site information that conflicts with the technology file

For information about resolving site definition conflicts, see Resolving Site Definition Conflicts When Reading LEF Files.

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
• Preparing Standard Cells

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.

icc2_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 2-2 to convert the LEF CLASS properties to `design_type` attributes for the processed blocks.

**Table 2-2  LEF CLASS Property to IC Compiler II `design_type` Attribute Mapping**

<table>
<thead>
<tr>
<th>LEF syntax</th>
<th>CLASS property</th>
<th>Subtype</th>
<th>IC Compiler II <code>design_type</code> attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLOCK</td>
<td>N/A</td>
<td>macro</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BLACKBOX</td>
<td>black_box</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SOFT</td>
<td>module</td>
<td></td>
</tr>
<tr>
<td>CORE</td>
<td>N/A</td>
<td>lib_cell</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ANTENNACELL</td>
<td>diode</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FEEDTHRU</td>
<td>feedthrough</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SPACER</td>
<td>filler</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TIEHIGH</td>
<td>lib_cell</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TIELOW</td>
<td>lib_cell</td>
<td></td>
</tr>
<tr>
<td></td>
<td>WELLTAP</td>
<td>well_tap</td>
<td></td>
</tr>
<tr>
<td>COVER</td>
<td>N/A</td>
<td>cover</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BUMP</td>
<td>flip_chip_pad</td>
<td></td>
</tr>
<tr>
<td>ENDCAP</td>
<td>BOTTOMLEFT</td>
<td>corner</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BOTTOMRIGHT</td>
<td>corner</td>
<td></td>
</tr>
<tr>
<td></td>
<td>POST</td>
<td>end_cap</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PRE</td>
<td>end_cap</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TOPLEFT</td>
<td>corner</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TOPRIGHT</td>
<td>corner</td>
<td></td>
</tr>
<tr>
<td>PAD</td>
<td>N/A</td>
<td>pad</td>
<td></td>
</tr>
<tr>
<td></td>
<td>AREAIO</td>
<td>flip_chip_driver</td>
<td></td>
</tr>
<tr>
<td></td>
<td>INOUT</td>
<td>pad</td>
<td></td>
</tr>
<tr>
<td></td>
<td>INPUT</td>
<td>pad</td>
<td></td>
</tr>
<tr>
<td></td>
<td>OUTPUT</td>
<td>pad</td>
<td></td>
</tr>
<tr>
<td></td>
<td>POWER</td>
<td>pad</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SPACER</td>
<td>pad_spacer</td>
<td></td>
</tr>
<tr>
<td>RING</td>
<td>N/A</td>
<td>macro</td>
<td></td>
</tr>
</tbody>
</table>
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:

  `icc2_lm_shell> read_lef -convert_sites {unit new_unit} myphys.lef`

  **Note:** You can use this method only for reference libraries that contain only single-height cells. If the reference library contains both single-height cells and multiple-height cells, associate site definitions with the standard cells by using the `set_attribute` command to set the `site_name` attribute, as described in Preparing Standard Cells.

- 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 `file.lef.auto_rename_conflictSites` 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 file 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:

```
icc2_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.

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 Blocks in the Physical Libraries
- Specifying the Site Definitions When Reading GDSII or OASIS Files
- Adjusting Standard Cell Boundaries
- Identifying Power and Ground Pins
- 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

  ```sh
  icc2_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

  ```sh
  icc2_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. To specify the block type for the blocks in a GDSII or OASIS file, create a block mapping file. For information about the syntax of the block mapping file, see Block Mapping File Syntax. When you run the `read_gds` or `read_oasis` command, use the `-block_map` option to specify the block mapping file.

**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.

To specify the site definitions for the blocks in a GDSII or OASIS file, create a block mapping file. For information about the syntax of the block mapping file, see Block Mapping File Syntax. When you run the `read_gds` or `read_oasis` command, use the `-block_map` option to specify the block mapping file.

You can also associate site definitions with the standard cells by using the `set_attribute` command after loading the library source file, as described in Preparing Standard Cells.

**Adjusting Standard Cell Boundaries**

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 2-4 shows the default and adjusted boundaries for a standard cell with shared power and ground pins.
Figure 2-4  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 2-1 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 2-1  Script to Resize Cell Boundaries**

```bash
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 -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 all pins in the library source file are signal pins.

To identify power and ground pins, use the `file.format.port_type_map` application option to map the pin name to its type, where `format` is either `gds` or `oasis`. The mapping applies to all cells in the GDSII or OASIS files.

For example, to set all pins named VDD as power pins and all pins named VSS as ground pins when reading GDSII files, use the following command:

```
icc2_lm_shell> set_app_options -name file.gds.port_type_map
-value {{power VDD} {ground VSS}}
```

**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 the *IC Compiler II Graphical User Interface User Guide*.

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*.

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 2-3
describes the supported methods, the keywords used to enable them, and the default method for each block type.

**Table 2-3 Pin Tracing Methods**

<table>
<thead>
<tr>
<th>Tracing method</th>
<th>Cell types for which this is the default tracing method</th>
<th>-trace_option Keyword</th>
<th>-layer Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create terminal shapes only for shapes that overlap pin text.</td>
<td>Modules, macro cells¹, analog cells, and black box cells</td>
<td>pins_only</td>
<td>pins_only</td>
</tr>
<tr>
<td>Trace the connectivity only on the same layer as the pin text.²</td>
<td>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</td>
<td>same_layer</td>
<td>same</td>
</tr>
<tr>
<td>Trace the connectivity across all layers.²</td>
<td>N/A</td>
<td>all_layers</td>
<td>all</td>
</tr>
</tbody>
</table>

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.
2. To reduce runtime, limit the number of geometries traced by setting the -trace_connectivity_limit option.

**Mapping Text Layers**

If the text is not on the same layer as the metal shape, you must use the file.format.text_layer_map application option to map the text layers to the metal layers, where format is either gds or oasis. Use the following format to specify the mapping:

```
{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

**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
  Valid values are `power`, `ground`, `signal`, `clock`, `boundary`, `hard_placement_blockage`, `soft_placement_blockage`, `routing_blockage`, and `area_fill`.
- `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 2-4 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 2-4  Additional Layer Mapping File Statements

<table>
<thead>
<tr>
<th>Mapping file statement</th>
<th>Command-line option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>read_always true</td>
<td>-read_always true</td>
<td>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.</td>
</tr>
<tr>
<td>read_always false</td>
<td>-read_always false</td>
<td>The command issues an error message and fails if the library source file contains layers that are not defined in the technology file.</td>
</tr>
<tr>
<td>read_always false -mapped_only</td>
<td>-read_always false</td>
<td>The command loads only those layers specified in the layer mapping file and ignores all other layers defined in the library source file.</td>
</tr>
<tr>
<td>read_always false -ignore_missing_layers</td>
<td>-read_always false</td>
<td>The command loads only those layers defined in the technology file and ignores all other layers defined in the library source file.</td>
</tr>
<tr>
<td>blockage_as_zero_spacing false</td>
<td>N/A</td>
<td>The command does not mark blockages read from the library source file as zero-spacing blockages. This is the default behavior.</td>
</tr>
<tr>
<td>blockage_as_zero_spacing true</td>
<td>N/A</td>
<td>The command marks blockages read from the library source file as zero-spacing blockages.</td>
</tr>
<tr>
<td>cell_prop_attribute attribute_value</td>
<td>-cell_property</td>
<td>The command sets the cell instance names by using the name associated with the specified attribute value.</td>
</tr>
</tbody>
</table>
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 2-2  Layer Mapping File

```plaintext
; 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 a Reference Library

To load the physical data from a complete or frame-only reference library into the current library workspace, use the `read_ndm` command. If the reference 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.ndm frame-only library into the current library workspace, use the following command:

```
icc2_lm_shell> read_ndm myphys.ndm
```

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

---

**Table 2-4  Additional Layer Mapping File Statements (Continued)**

<table>
<thead>
<tr>
<th>Mapping file statement</th>
<th>Command-line option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>net_prop_attribute attribute_value</td>
<td>-net_property attribute_value</td>
<td>The command associates geometries with nets by using the specified attribute value.</td>
</tr>
<tr>
<td>pin_prop_attribute attribute_value</td>
<td>-pin_property attribute_value</td>
<td>The command associates geometries with pins by using the specified attribute value.</td>
</tr>
</tbody>
</table>
Each time you run the command, you can specify only a single library file to load; to load multiple libraries, you must run the command multiple times. In addition, this command loads only reference libraries; you cannot use this command to load aggregate libraries.

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

**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.ndm design library into the current library workspace, use the following command:

```shell
icc2_lm_shell> read_ndm myblock.ndm
```

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

```shell
icc2_lm_shell> read_ndm -label mylabel myblock.ndm
```

*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 file to load; to load multiple libraries, you must run the command multiple times.

You can specify the library file with an absolute path, a relative path, or no path. If you specify the file with a relative path or with no path, the library manager uses the search path to find the file.
Importing Milkyway FRAM Views From the IC Compiler Tool

If you do not have source files for your physical libraries, but do have Milkyway FRAM views in an IC Compiler reference library, you can use these to create an IC Compiler II frame-only reference library. You can then use this frame-only reference library to create a complete IC Compiler II reference library.

To create a frame-only reference 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 IC Compiler II Library Manager tool, create a frame-only 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:
   
   ```
   icc2_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:
   
   ```
   icc2_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 Reference Libraries
- Loading Physical Data from a Reference Library

---

**Specifying Which Blocks to Process**

By default, when you load logic libraries or physical libraries, the tool processes all of the cells defined in the library source 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 Blocks in the Physical Libraries**

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 **lib_cell**, **macro**, **pad**, **corner**, **pad_spacer**, **cover**, **flip_chip_pad**, **flip_chip_driver**, **analog**, **black_box**, **feedthrough**, **well_tap**, **abstract**, **end_cap**, **diode**, **filler**, and **fill**
  If you do not specify this argument, the **read_gds** and **read_oasis** commands use a type of **lib_cell**.

- **-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. You can also specify the site definition for a library cell by using the **set_attribute** command to set its **site_name** attribute.
  For more information about setting the site definitions, see Preparing Standard Cells.

- **-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 Adjusting 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 2-3  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
```

---

**Creating Scaling Groups**

A scaling group enables the IC Compiler II tool to perform timing analysis and optimization at intermediate voltage and temperature operating conditions. You can create a scaling group when the voltage and temperature combinations of the logic libraries for a given process form a complete grid.

When deciding whether to create scaling groups, you need to consider the effect of the index values (breakpoints) used for the timing arc lookup tables in each library.

- If the breakpoints for all the lookup table indexes for a given arc are the same for all the characterization points, you should create scaling groups because you gain efficiency in delay calculation with a minimal impact on memory usage.
- If the breakpoints for at least one index in the lookup table are the same for all the characterization points, you can create scaling groups, but there is a larger impact on memory usage.
- If the breakpoints for all indexes in the lookup table for a given arc differ among the characterization points, you cannot create scaling groups because the memory cost is too high.

**Note:**

- If the logic libraries contain on-chip variation (OCV) data, you cannot create scaling groups.

To create a scaling group, use the `create_scaling_group` command to specify the logic libraries in the scaling group. The `create_scaling_group` command verifies that the specified libraries meet the requirements; if they do not meet the requirements, the command fails and issues an error message. When you run the `check_workspace`
command, the tool performs further validation of the scaling groups (for details, see Scaling Group Checks).

For example, assume that you load four logic libraries with a process label of TYP: one at 1.0V and 0°C (typ_pvt1.db), one at 1.0V and 125°C (typ_pvt2.db), one at 0.8V and 0°C (typ_pvt3.db), and one at 0.8V and 125°C (typ_pvt4.db). To create a scaling group using these logic libraries, use the following command:

```
icc2_lm_shell> create_scaling_group \[get_libs {typ_pvt1 typ_pvt2 typ_pvt3 typ_pvt4}]```

In the reference library, the scaling group is represented as a single pane that covers the whole range of voltages and temperatures.

You can define any number of scaling groups, but a logic library can appear in only one scaling group. A reference library can contain some logic libraries in scaling groups, and others standalone.

See Also
- Scaling Group
- Breakpoints
- Loading the Logic Libraries

---

**Loading Parasitic Parameters**

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 coefficients.

To read a TLUPlus file into the library workspace, use the `read_parasitic_tech` command. You must specify the TLUPlus files by using the `-tlup` option. If you specify the TLUPlus 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 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
  ...
```
To include comments in the layer mapping file, start the line with an asterisk (*) or pound sign (#).

After you read in a TLUPlus file, it is identified by its parasitic technology model name. By default, the parasitic technology model name is the base name of the specified TLUPlus 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:

```
icc2_lm_shell> read_parasitic_tech -tlup my.tlup \
      -layermap my.layermap -name para1
```

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

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

```
icc2_lm_shell> read_signal_em_constraints em.itf
```

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

```
icc2_lm_shell> read_signal_em_constraints -format ALF em.alf
```

If the library workspace already contains signal electromigration constraints, it is replaced by the data in the new file.

Completing the Physical Cell Information

The physical libraries might not contain all of the information required by the IC Compiler II tool for placement and routing. The following topics describe how to annotate missing physical information for a reference library:

- Preparing I/O Cells
- Preparing Standard Cells
- Preparing Macros
Preparing I/O Cells

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`.

- **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`.

In general, the LEF file defines these attributes, and the library manager automatically sets them when you read the LEF file. If you load the I/O cells 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.

Preparing Standard Cells

Standard cells must have the following attributes to enable placement and legalization in the IC Compiler II tool:

- **site_name** library cell attribute
  
  This attribute identifies the site definition associated with the standard cell. If a standard cell does not have a site definition or its site definition does not match a site definition in the technology file, the IC Compiler II tool cannot place the cell.

  **Note:**
  
  Use the single-height site definition, which is often named `unit`, for both single-height and multiple-height cells. The IC Compiler II tool automatically identifies the polarity of the power and ground pins and places the single-height and the multiple-height cells accordingly.

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

  ```
  icc2_lm_shell> set_attribute -objects [get_lib_cells */*/design] \
  -name site_name -value unit
  ```

  To report the site definitions in the technology file, use the `report_site_defs` command.
boundary library cell attribute

This attribute defines the boundary of the standard cell. 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.

For information about changing the boundary attribute, see Adjusting Standard Cell Boundaries.

In general, the LEF file defines these attributes, and the library manager automatically sets them when you read the LEF file. If you load the standard cells 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.

See Also

• Resolving Site Definition Conflicts When Reading LEF Files
• Specifying the Site Definitions When Reading GDSII or OASIS Files
• Defining Antenna Properties on Standard Cells

Preparing Macros

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:

```bash
% 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 IC Compiler II 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 libraries loaded into the workspace
- The TLUPlus files loaded into the workspace
- The validation status of the workspace

For example,

```bash
icc2_lm_shell> report_workspace
Workspace name: myreflib  
...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.

```shell
icc2_lm_shell> check_workspace
```

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

- Generates the frame views of 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)
- **Scaling Group Checks** (includes checks for unique libraries for each characterization point, and differences in threshold value settings and timing arc breakpoints)
- **Leakage-Power Checks** (includes checks for missing leakage-power information and differences in power conditions)

**Note:**

The checks performed by the `check_workspace` command validate the consistency of the library source files loaded into the workspace; however, the workspace might not
contain all of the information required by the IC Compiler II tool for placement and routing. For information about completing the information, see the following topics:

- Completing the Technology Data
- Completing the Physical Cell Information

To generate a reference library from a library workspace, the workspace must pass these checks. However, in some cases you can generate a reference library using incomplete or inconsistent source libraries by defining a mismatch configuration, as described in Allowing Incomplete or Inconsistent Library Data. Reference 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`, `scaling_group_info`, 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:

- The library workspace contains at least one physical library.
- 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 2-4 shows the use of these attributes to define the power connections in a .lib logic library file.

Example 2-4  Logic Library Example With voltage_map Attributes

```verbatim
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*/

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:

```plaintext
icc2_lm_shell> rename_rail -library A -from VDD1.0 -to VDD
icc2_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 reference 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 `-ignore_missing` option to ignore the errors for cells that are defined in the base library but not in another logic library
  The generated reference library will contain the set of cells from the base library; if a cell does not exist in another logic library, it will have missing data in the pane associated with that logic library.
- Rerun the `check_workspace` command with the `-allow_missing` option to enable the tool to add the missing cells to the generated reference library
  The generated reference 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 not in any physical library
  If a cell exists in the logic libraries, but not in any physical library, the tool issues an error message and you cannot commit the workspace.

  To create a reference 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 source libraries

  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 cell that exists in a physical library, but not in the logic libraries
  If a cell exists in a physical library, but not in any logic library, the tool behavior depends on the flow you selected.
  - For the normal and organize-by-PVT flows, the cell is not included in the generated reference library.
  - For the physical-only flow, the cell is included in the generated reference 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 reference 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 against the pins defined for the cell in the base library. If a cell in any library (logic or physical) 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 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, the tool issues a warning, redefines the pin as a PG pin, and removes all timing information from the pin.

To create a reference 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 in any library (logic or physical) 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 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, 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 reference 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, 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, the tool issues a warning, redefines the pin as a PG pin, and removes all timing information from the pin.
To generate a reference 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) 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.

```
icc2 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,

```
icc2 lm_shell> set_attribute [get_lib_pins lib/cell/pin] \
        related_power_pin_name PGpin_name
```

Note:
You can ignore missing power and ground relationships during the workspace validation by setting the lib.workspace.allow_missing_related_pg_pins application option to true before running the check_workspace command. However, the missing information might affect analysis performed by the IC Compiler II tool.

• Pin order

For each cell, the tool determines the pin order in each logic and physical library 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, 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 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:

```plaintext
cell from_pin to_pin [related_pin] sense [object_name] [{w:when_cond}] [MIN] [{m:modes}] [DUP\(n\)]
```

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\(n\)` 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. You must decide how you want to handle these cells, and then rerun the `check_workspace` command with either the `-ignore_missing` or `-allow_missing` option (unless you correct the libraries).

- To ignore arcs that are defined in the base library but not in another logic library, use the `-ignore_missing` option when you rerun the `check_workspace` command. The generated reference library will contain only the timing arcs from the base library.
- To add arcs to the generated reference library that are not defined in the base library, but are defined in another logic library, use the `-allow_missing` option when you rerun the `check_workspace` command. Note that 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` or `-ignore_missing` option, the reference 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.

Scaling Group Checks
For each scaling group, the check_workspace command verifies that

• Each point on the grid matches exactly one logic library
• The threshold values are the same for all logic libraries in a scaling group

The tool checks the threshold values at both the pin level and the library level. The threshold values are defined by the following attributes: input_threshold_pct_fall, input_threshold_pct_rise, output_threshold_pct_fall, output_threshold_pct_rise, slew_lower_threshold_pct_fall, slew_lower_threshold_pct_rise, slew_upper_threshold_pct_fall, slew_upper_threshold_pct_rise, and slew_derate_from_library.

If a threshold value is not the same among all the logic libraries in a scaling group, the tool issues an error message and you cannot commit the workspace.

For detailed messages during the scaling group checks, specify the scaling_group_info keyword with the -details option.

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}]

You must decide how you want to handle these cells, and then rerun the check_workspace command with either the -ignore_missing or -allow_missing option (unless you correct the libraries).

• To ignore leakage-power information that is defined in the base library but not in another logic library, use the -ignore_missing option when you rerun the check_workspace
command. The generated reference library will contain only the leakage-power information from the base library.

- To add leakage-power information to the generated reference library that is not defined in the base library, but is defined in another logic library, use the -allow_missing option when you rerun the check_workspace command.

Note:
If you choose to override this error with the -allow_missing or -ignore_missing option, the reference 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.

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 reference 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 reference 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. Reference 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:

- **tapeout** *(the default)*
  
  This is the default behavior of the workspace validation and must be used to generate reference libraries that can be used throughout the implementation flow.

- **prototype**
  
  This mismatch configuration enables the `check_workspace` command to repair the mismatches for all of the supported error types, which are shown in Table 2-5.

### Table 2-5 Mismatch Configuration Error Types

<table>
<thead>
<tr>
<th>Error type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>lib_cell_name_case</code></td>
<td>Cell name mismatch due to case-sensitivity.</td>
</tr>
<tr>
<td><strong>Supported repairs:</strong></td>
<td></td>
</tr>
<tr>
<td>• <code>record_lib_cell_name_case_insensitive</code></td>
<td>Enables case-insensitivity when comparing cell names. Uses the cell name from the base library.</td>
</tr>
<tr>
<td><code>lib_missing_logical_port</code></td>
<td>Logic library cell is missing a signal port that exists in the base logic library or physical library.</td>
</tr>
<tr>
<td><strong>Supported repairs:</strong></td>
<td></td>
</tr>
<tr>
<td>• <code>create_placeholder_logic_lib_port</code></td>
<td>Creates a placeholder port on the logic library cell.</td>
</tr>
<tr>
<td><code>lib_missing_physical_port</code></td>
<td>Physical library cell is missing a port that exists in the base logic library.</td>
</tr>
<tr>
<td><strong>Supported repairs:</strong></td>
<td></td>
</tr>
<tr>
<td>• <code>create_placeholder_physical_lib_port</code></td>
<td>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.</td>
</tr>
<tr>
<td><code>lib_missing_physical_reference</code></td>
<td>A cell exists in the logic libraries, but not in any physical libraries.</td>
</tr>
<tr>
<td><strong>Supported repairs:</strong></td>
<td></td>
</tr>
<tr>
<td>• <code>create_placeholder_physical_lib_cell</code></td>
<td>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.</td>
</tr>
</tbody>
</table>
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.

### Table 2-5  Mismatch Configuration Error Types (Continued)

<table>
<thead>
<tr>
<th>Error type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lib_port_name_case</td>
<td>Pin name mismatch due to case-sensitivity.</td>
</tr>
<tr>
<td>Supported repairs:</td>
<td></td>
</tr>
<tr>
<td>• record_lib_port_name_case</td>
<td>Enables case-insensitivity when comparing pin names.</td>
</tr>
<tr>
<td>insensitive</td>
<td>Uses the pin name from the base library.</td>
</tr>
<tr>
<td>lib_port_direction</td>
<td>A library cell pin has a different direction than in the base library.</td>
</tr>
<tr>
<td>Supported repairs:</td>
<td>Uses the pin direction from the base library in the case of signal pin mismatches. Uses the pin direction from the physical library in the case of PG pin mismatches.</td>
</tr>
<tr>
<td>• change_lib_port_direction</td>
<td>Note: The library manager fixes many types of pin direction mismatches by default. For details, see Pin Definition Checks.</td>
</tr>
<tr>
<td>lib_port_type</td>
<td>A library cell pin has a different type than in the base library.</td>
</tr>
<tr>
<td>Supported repairs:</td>
<td>Uses the pin type from the base library in the case of mismatches.</td>
</tr>
<tr>
<td>• change_lib_port_type</td>
<td>Note: The library manager fixes many types of pin type mismatches by default. For details, see Pin Definition Checks.</td>
</tr>
</tbody>
</table>
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 tapeout 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 this attribute: `error`, `ignore`, or `repair`. The initial setting of this attribute comes from the reference configuration used to create the user-defined configuration.

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 the repair keyword specified in Table 2-5 for the error type.

For example, the following commands create a user-defined mismatch configuration named `user_def` that repairs missing physical library cells.

```bash
icc2_lm_shell> create_mismatch_config user_def
icc2_lm_shell> set_attribute \
   [get_mismatch_type lib_missing_physical_reference] \
   action(user_def) repair
icc2 lm_shell> set_attribute \
   [get_mismatch_type lib_missing_physical_reference] \
   current_repair(user_def) create_placeholder_physical_lib_cell
```

Setting a Mismatch Configuration

By default, the library manager uses the tapeout mismatch configuration. To set a different mismatch configuration, use the `set_current_mismatch_config` command. You can specify one of the predefined mismatch configurations (`tapeout` or `prototype`) or a user-defined mismatch configuration.

For example, to set the mismatch configuration to the predefined prototype configuration, use the following command:

```bash
icc2 lm_shell> set_current_mismatch_config prototype
```

To see the available mismatch configurations, use the `report_mismatch_configs` command.
Reporting Mismatch Configurations

To report the current mismatch configuration, use the `get_current_mismatch_config` command.

```plaintext
icc2_lm_shell> get_current_mismatch_config tapeout
```

To restrict the report to specific mismatch configurations, use the `--config_list` option. Note that the report includes mismatch error types that apply to both the library manager and the implementation tool; only the error types listed in Table 2-5 apply to the library preparation process.

For example, to report the details about the tapeout configuration, use the following command:

```plaintext
icc2_lm_shell> report_mismatch_configs --config_list tapeout
```

```
****************************************
Report : Reporting Mismatch Configs
Version: J-2014.12
Date   : Tue Nov 18 21:41:23 2014
****************************************
Config : tapeout
Mismatch-Type Action Repair-Strategy Available Strategies
-----------------------------------------------
lib_missing_physical_reference error null {create_placeholder_physical_lib_cell }
lib_missing_logical_port error null {create_placeholder_logic_lib_port }
lib_missing_physical_port error null {create_placeholder_physical_lib_port }
lib_port_type error null {change_lib_port_type }
lib_port_direction error null {change_lib_port_direction }
lib_port_name_synonym error null {null}
lib_cell_name_case error null {record_lib_cell_name_case_insensitivity }
lib_port_name_case error null {record_lib_port_name_case_insensitivity }
lib_no_pg_port error null {null}
bus_bit_naming error null {bus_bit_blast_naming }
```

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 IC Compiler II Library Manager tool uses the `lib.physical_model` application options to control the frame view generation. You must set these application options before running

---

Chapter 2: Preparing Reference Libraries

Building Reference Libraries
the `check_workspace` command. When you commit the workspace, the library manager saves these application option settings with the reference library.

To update frame views for an existing reference library, such as when the technology file or tool version is updated, use the process described in Modifying a Reference Library.

**See Also**

- Building Reference Libraries
- Frame View Generation Application Options

## 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 2-6 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 2-7 shows the `is_secondary_pg` settings for the various combinations of these attributes.

### Table 2-6  Determining the `is_secondary_pg` Setting for PG Pins

<table>
<thead>
<tr>
<th>Logic library PG pin attribute</th>
<th>Reference library cell attribute</th>
<th>Reference library pin attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pg_type</code></td>
<td><code>std_cell_main_rail</code></td>
<td><code>design_type</code></td>
</tr>
<tr>
<td>primary_power or primary_ground</td>
<td>true</td>
<td>all</td>
</tr>
<tr>
<td></td>
<td>false or unset</td>
<td>all</td>
</tr>
<tr>
<td></td>
<td>any value</td>
<td>all</td>
</tr>
</tbody>
</table>
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 reference library. To commit the workspace, use the `commit_workspace` command. By default, the tool saves the reference library in the current directory using the name `workspace_name.ndm` using the current schema revision.

```
icc2_lm_shell> commit_workspace
```

To override the default behavior, use the following options:

- `--output file_name`
  This option saves the reference library with the specified file name. For example, to save the reference library in a file named `myreflib.ndm`, use the following command:

```
icc2_lm_shell> commit_workspace --output myreflib.ndm
```

Note that this option also changes the name of the reference library.

- `--version version_string`
  This option saves the reference 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 reference library already exists. To overwrite an existing reference library, use the `-force` option. For example,

```
icc2_lm_shell> commit_workspace --output myreflib.ndm
Error: File 'myreflib.ndm' already exists (FILE-009)
icc2_lm_shell> commit_workspace --force --output myreflib.ndm
```

To enable all `commit_workspace` commands to overwrite an existing reference library without using the `-force` option, set the `lib.workspace.allow_commit_workspace_overwrite` application option to true.

---

**Table 2-7  Determining the is_secondary_pg Setting for Bias Pins**

<table>
<thead>
<tr>
<th>Logic library PG pin attribute</th>
<th>Reference library pin attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pg_type</code></td>
<td><code>is_secondary_pg</code></td>
</tr>
<tr>
<td>physical_connection</td>
<td>routing_pin</td>
</tr>
<tr>
<td>deepwell</td>
<td>true</td>
</tr>
<tr>
<td>device_layer</td>
<td>false</td>
</tr>
</tbody>
</table>

Table 2-7 Determining the `is_secondary_pg` Setting for Bias Pins
icc2 lm_shell> set_app_options \ 
    -name lib.workspace.allow_commit_workspace_overwrite -value true

The views saved in the generated reference library depend on the library preparation flow you enabled and the settings of the related application options.

- For the normal, organize-by-PVT, 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 reference library.

**See Also**

- Library Preparation Flows

---

**Using the Exploration Flow**

The exploration flow automatically analyzes your library source files and generates a script that you can use to create a reference library.

To use the exploration flow to build a reference 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 Libraries.
4. Load the physical library source files into the root workspace, as described in Loading the Physical Libraries.
5. (Optional) Enable scaling group creation, as described in Enabling Scaling Group Creation.
6. Analyze the library source files to automatically create the subworkspaces, as described in Performing Automated Library Analysis.
7. (Optional) Edit the subworkspaces.

8. (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.

9. Validate the library workspace and commit it to disk.

   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 run this command, the tool validates each subworkspace in the root workspace
     and, if the workspace passes validation, commits it to a reference 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 reference library, as
     described in Using a Library Preparation Script.

Enabling Scaling Group Creation

By default, the exploration flow does not create scaling groups for the logic libraries;
however, the `group_libs` command issues the following message if it determines that
scaling groups could be created:

Information: Identified a potential valid scaling group. (LM-027)

To enable scaling group creation, set the `lib.workspace.group_libs_create_slg`
application option to `true` before running the `group_libs` command.

```
icc2_lm_shell> set_app_options \ 
    -name lib.workspace.group_libs_create_slg -value true
```

See Also

- Creating Scaling Groups
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.

```shell
icc2_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 single-cell logic libraries that fit the organize-by-PVT flow and multiple-cell libraries that fit the normal flow. 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 libraries that include at least the same set of cells. If the physical libraries 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 reference libraries. In this case, the tool uses the data from the first reference library in which it finds the cell, and it ignores that cell's data in all other reference libraries.

  By default, the `group_libs` command adjusts the grouping such that each cell exists in only one generated reference 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.

- 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 reference 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. Creates scaling groups for the logic libraries, if enabled.
4. Validates the contents of the library workspace.
5. Commits the library workspace to disk using the name of the first logic library as the reference 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 reference libraries generated for the workspaces.

Note:
The generated script file also includes any set_app_options and reset_app_options commands issued in the icc2_lm_shell session before running the write_workspace command.
To use this method to create the reference 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:
  
  ```
  icc2_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:
  
  ```
  icc2_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:

```
icc2_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.

```
icc2_lm_shell> remove_workspace
```

3. Create the reference libraries by sourcing the generated script.

```
icc2_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.
Verifying Reference Libraries

The IC Compiler II library manager provide a verification flow to validate the logical information or frame views of a reference library. You can use this flow to

- Compare the logical information in a logic library source file and an existing reference library, as described in Verifying Logical Information in a Reference Library.
- Compare the frame views in two reference libraries. The reference libraries can be complete libraries or frame-only libraries, as described in Verifying the Frame Views in a Reference Library.

Verifying Logical Information in a Reference Library

The library manager can verify that the following objects are the same in a reference 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 reference library,

1. Open the reference library in a verification workspace by using the `create_workspace -flow verification` command.
   
   icc2_lm_shell> create_workspace -flow verification mylib.ndm
   
   The reference 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 reference library in the library workspace by using the `compare_db` command.
   
   icc2_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 reference library, remove the library workspace by using the remove_workspace command.

---

**Verifying the Frame Views in a Reference Library**

The library manager can verify that the frame views are the same in two reference libraries by comparing the following information:

- The schema version
- The default site
- Number of frame views
- The design type, boundary, port information, number of terminals, existence of access edges, and via region contact codes for each frame view

To verify the frame views in a reference library,

1. Open the reference library in a verification workspace by using the create_workspace -flow verification command.

   icc2_lm_shell> create_workspace -flow verification mylib1.ndm

   The reference library opened in the verification workspace is referred to as the base library.

2. Compare the frame views in another reference library to the reference library in the library workspace by using the compare_ndm command.

   icc2_lm_shell> compare_ndm mylib2.ndm

   The reference library opened by the compare_ndm command is referred to as the comparison library.

3. When you have finished verifying the reference library, remove the library workspace by using the remove_workspace command.
Modifying a Reference Library

You can modify the technology data and physical cell data, but not functional cell data, for an existing reference library.

To modify a reference 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 reference library stored in the my_ref.ndm file, use the following command:

   ```
   icc2_lm_shell> create_workspace my_ref.ndm -flow edit
   ```

3. Modify the reference library by performing one or more of the following tasks:
   - Load new technology data, as described in Loading the Technology Data.
   - 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 Libraries.
   - Load a reference library into the library workspace by using the `read_ndm` command, as described in Loading Physical Data from a Reference Library.
     
     Note:
     When you load a reference library into an edit workspace, only the physical information is loaded; the timing information is not used.
   - Modify physical cell attributes.
     
     In general, you can modify physical cell attributes by using the `set_attribute` command. For examples of this technique, see Completing the Physical Cell Information.
   - 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 the IC Compiler II Graphical User Interface User Guide.

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 reference 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 reference 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 reference 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
```

---

**Defining Antenna Properties on Standard Cells**

The IC Compiler II tool uses 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 reference library. Otherwise, you must use the edit flow to annotate these properties on the standard cells after generating the reference library.

To annotate antenna properties on a standard cell,

1. Open the reference library in edit mode as described in Modifying a Reference 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 2-8 Standard Cell Antenna Property Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>gate_area</code></td>
<td>This attribute defines the gate area of an input pin.</td>
</tr>
<tr>
<td></td>
<td>• If the library supports a single gate oxide mode, the attribute value</td>
</tr>
<tr>
<td></td>
<td>uses the following syntax:</td>
</tr>
<tr>
<td></td>
<td>{ {layer_name area_value} ... }</td>
</tr>
<tr>
<td></td>
<td>• If the library supports multiple gate oxide modes, the attribute value</td>
</tr>
<tr>
<td></td>
<td>uses the following syntax:</td>
</tr>
<tr>
<td></td>
<td>{ {oxide_mode layer_name area_value} ... }</td>
</tr>
<tr>
<td></td>
<td>Note: If this attribute is not specified for a pin, the IC Compiler II tool</td>
</tr>
<tr>
<td></td>
<td>uses the setting of the <code>route.detail.default_gate_size</code> application</td>
</tr>
<tr>
<td><code>antenna_area</code></td>
<td>These attributes define the antenna area of an input pin. The</td>
</tr>
<tr>
<td><code>antenna_side_area</code></td>
<td>The <code>antenna_area</code> attribute defines the antenna polygon area. The</td>
</tr>
<tr>
<td></td>
<td>The <code>antenna_side_area</code> attribute defines the antenna sidewall area.</td>
</tr>
<tr>
<td></td>
<td>The attribute values use the following syntax:</td>
</tr>
<tr>
<td></td>
<td>{layer_name area_value}</td>
</tr>
<tr>
<td></td>
<td>Note: If these attributes are not specified for a pin, the IC Compiler II</td>
</tr>
<tr>
<td></td>
<td>tool uses the pin shapes to compute the antenna area.</td>
</tr>
</tbody>
</table>
Defining Antenna Properties on Hard Macro Cells

The IC Compiler II tool uses 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 reference library. Otherwise, you must use the edit flow to annotate these properties on the hard macro cells after generating the reference library.

To annotate antenna properties on a hard macro,

1. Open the reference library in edit mode as described in Modifying a Reference 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.

Table 2-8  Standard Cell Antenna Property Attributes (Continued)

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>diff_area</td>
<td>This attribute defines the diode protection value of an output pin.</td>
</tr>
<tr>
<td></td>
<td>The attribute value uses the following syntax:</td>
</tr>
<tr>
<td></td>
<td><code>{layer_name area_value}</code></td>
</tr>
<tr>
<td></td>
<td>Note:</td>
</tr>
<tr>
<td></td>
<td>If this attribute is not specified for a pin, the IC Compiler II tool uses</td>
</tr>
<tr>
<td></td>
<td>the setting of the <code>route.detail.default_diode_protection</code> application option.</td>
</tr>
<tr>
<td>is_diode</td>
<td>This attribute specifies whether a pin is a diode that can be used by</td>
</tr>
<tr>
<td></td>
<td>the IC Compiler II tool to fix antenna violations.</td>
</tr>
</tbody>
</table>
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:

   \[
   \{ \{ \text{layer\_name} \ \text{area\_value} \} \ldots \} 
   \]

   For example,

   \[
   \text{icc2 lm shell} > \text{set\_attribute mylib/hm1/z diff\_area} \ \{ M1 \ 0 \ M2 \ 0 \ M3 \ 0 \ M4 \ 0.0595 \}
   \]

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:

   \[
   \text{set\_port\_antenna\_property -port port -data property\_settings\_list} 
   \]

   In the property\_settings\_list argument, specify the properties for one or more layers using the following syntax:

   \[
   \{ \{ \text{metalLayer} \ \text{layer\_name} \ \text{gate\_size} \ \text{area\_value} \ \text{model1\_area} \ \text{mode1\_data} \ \text{mode2\_ratio} \ \text{mode2\_data} \ \text{mode3\_area} \ \text{mode3\_data} \ \text{mode4\_area} \ \text{mode4\_data} \ \text{mode5\_ratio} \ \text{mode5\_data} \ \text{mode6\_area} \ \text{mode6\_data} \} \ldots \} 
   \]

   where

   \[
   \begin{align*}
   \text{layer\_name} & \text{ is the name of the metal layer in the technology file} \\
   \text{area\_value} & \text{ is the gate area of the port on the specified layer} \\
   \text{model1\_data} & \text{ is the area of the specified metal layer when the connectivity is traced up to that metal layer} \\
   \text{mode2\_data} & \text{ is the maximum internal ratio of the area of the metal layer below the specified metal layer when the connectivity is traced up to the specified metal layer} \\
   \text{mode3\_data} & \text{ is the total metal area}
   \end{align*} 
   \]
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:

```
icc2_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

```
icc2_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 reference libraries into a single reference 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:

   \[
   \text{icc2\_lm\_shell}\gt \text{create\_workspace my\_agg \ -flow aggregate}
   \]

3. Load the reference libraries into the library workspace, as described in Opening a Reference Library.
   
   All of the reference libraries in the library workspace become part of the aggregate library. To remove libraries from the workspace, use the \text{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 reference libraries comprising the aggregate library when resolving a cell reference.

To specify the search order, use the \text{set\_lib\_order} command to list the reference libraries in the order in which to look for files. When the IC Compiler II tool resolves a cell reference, it starts searching in the leftmost reference 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 existing library file name as the workspace name. For example, to modify the aggregate library stored in the my_agg.ndm file, use the following command:

   ```
   icc2_lm_shell> create_workspace my_agg.ndm -flow edit
   ```

3. Adjust the reference libraries in the library workspace.

   When you create a library workspace for an existing aggregate library, its reference libraries are automatically opened in the library workspace. To see the reference libraries in the workspace, use the `get_libs` or `report_workspace` command.

   To change the search order of the reference libraries, set the search order, as described in Setting the Search Order.

   To add reference libraries to the library workspace, open the libraries, as described in Opening a Reference Library.

   To remove reference libraries from the library workspace, close the libraries, as described in Closing a Reference 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 Reference Library

You can open a reference library outside of a library workspace to query or view the contents of the reference library. To open a reference library outside of a library workspace, use the `open_lib` command.

Note:
To load a reference library into a library workspace, use the `read_ndm` command, as described in Loading Physical Data from a Reference Library.

For example, to open the `myreflib` reference library, use the following command:

```
icc2_lm_shell> open_lib myreflib.ndm
```

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

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

See Also

• Defining the Search Path

Getting Information About Reference Libraries

You can report information about the reference 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 Libraries

To list the libraries loaded into memory, use the `get_libs` command.

```
icc2_lm_shell> get_libs
{myreflib myreflib_lvt myreflib_hvt}
```

If you have opened an aggregate library, the `get_libs` command returns the reference libraries within the aggregate library. The reference library names are prefixed with the aggregate library name.

```
icc2_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:

```
icc2_lm_shell> get_libs *lvt
{myreflib_lvt}
```

Reporting Libraries

To report information about a reference library, use the `report_lib` command. You must specify the name of an open reference 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 reference library:

- `-antenna`
  Reports the library cells that are missing antenna properties and the affected pins.

- `-parasitic_tech`
  Reports the TLUPlus files loaded into the library, if any.

- `-physical`
  Reports the physical characteristics of the reference 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 blocked terminals.
  To report details about the cell objects such as the number of terminals, shapes, and
  routing blockages, 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.

Example 2-5 shows an example of the default reference library report.
Example 2-5  Reference Library Report

icc2_lm_shell> report_lib myreflib
****************************************
Report : library
Library: myreflib
Version: K-2015.06
Date   : Thu Apr  4 12:50:33 2013
****************************************
Full name: /usr/libs/myreflib.ndm:myreflib
File name: /usr/libs/myreflib.ndm
Lib type: cell
Design count: 434
Timing data:

Power rails:

<table>
<thead>
<tr>
<th>Index</th>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>&lt;default&gt;</td>
<td>power</td>
</tr>
<tr>
<td>1</td>
<td>V</td>
<td>power</td>
</tr>
<tr>
<td>2</td>
<td>VL</td>
<td>power</td>
</tr>
<tr>
<td>3</td>
<td>VSS</td>
<td>ground</td>
</tr>
</tbody>
</table>

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/myreflib_125c.db
Pane 1:
...

Source .db libraries:
myreflib_125c.db:myreflib_125c
myreflib_40c.db:myreflib_40c
Operating Conditions:

<table>
<thead>
<tr>
<th>Name</th>
<th>Process</th>
<th>Temp</th>
<th>Voltage</th>
<th>Original DB Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORST</td>
<td>1.00</td>
<td>125.00</td>
<td>1.08</td>
<td>myreflib_max_hth</td>
</tr>
<tr>
<td>BEST</td>
<td>1.00</td>
<td>-40.00</td>
<td>1.32</td>
<td>myreflib_min</td>
</tr>
</tbody>
</table>

---

**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 reference library and the `-output` option to specify the name of the output file.

```
icc2_lm_shell> open_lib myreflib
icc2_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 reference 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 2-6 shows an example frame properties report, which was generated by the following command:

```
icc2_lm_shell> report_frame_properties -library [current_lib] \ 
               -output frame.txt -block myreflib/AND2/frame
```
Example 2-6  Frame Properties Report
****************************************
Report : frame properties
Library: myreflib
Version: J-2014.06-ICC2
Date   : Sat Aug 2 21:05:01 2014
****************************************
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 myreflib reference library, use the following command:

  icc2_lm_shell> get_lib_cells myreflib/*/frame
  {myreflib/AND2/frame myreflib/AND4/frame myreflib/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 myreflib reference library, use the following command:

```
icc2_lm_shell> get_lib_pins myreflib/AND2/*
{myreflib/AND2/IN1 myreflib/AND2/IN2 myreflib/AND2/OUT}
```

Note:
To reduce runtime when you load physical libraries 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 libraries, 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 in the normal flow.

---

**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, and pin count for the specified cells, as well as whether the cell is sequential or a black box.
For example, to report on the AND2 cell in the myreflib reference library, use the following command:

```
icc2_lm_shell> report_lib_cells -objects myreflib/AND2
```

```
****************************************
Report : lib_cell
Version: MS3.0
Date   : Thu Apr  4 13:17:21 2013
****************************************
full_name                  area    pin_count is_sequential is_black_box
-------------------- ---------- --------- ------------- ------------
myreflib/AND2           11.0005         3 false         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: `area`, `dont_touch`, `full_name`, `is_black_box`, `is_combinational`, `is_isolation`, `is_level_shifter`, `is_memory_cell`, `is_sequential`, `is_three_state`, `lib_name`, `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 myreflib reference library, use the following command:

```
icc2_lm_shell> report_lib_pins -objects myreflib/AND2/*
```

```
****************************************
Report : lib_pin
Version: MS3.0
Date   : Thu Apr  4 13:33:36 2013
****************************************
full_name                 direction
------------------------- --------
myreflib/AND2/IN1         in
myreflib/AND2/IN2         in
myreflib/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_traceable`, `is_user_function`, `name`, and `view_name`. To display all of the available information, specify `-columns all`.
is_three_state_enable_pin, is_three_state_output_pin, name, pin_number, port_type, rail_name, related_ground_pin, related_power_pin, 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 myreflib reference library, use the following command:

```sh
icc2_lm_shell> report_lib_timing_arcs \
    -objects [get_lib_timing_arcs -of_objects myreflib/AND2]
```

```
****************************************
Report : lib_timing_arc
Version: J-2014.06-ICC2
Date   : Sat Aug 2 20:14:31 2014
****************************************

<table>
<thead>
<tr>
<th>from</th>
<th>to</th>
<th>sense</th>
<th>when</th>
<th>sdf_cond</th>
<th>is_disabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>Y</td>
<td>positive_unate</td>
<td></td>
<td></td>
<td>false</td>
</tr>
<tr>
<td>A2</td>
<td>Y</td>
<td>positive_unate</td>
<td></td>
<td></td>
<td>false</td>
</tr>
<tr>
<td>A1</td>
<td>Y</td>
<td>positive_unate</td>
<td></td>
<td></td>
<td>false</td>
</tr>
<tr>
<td>A2</td>
<td>Y</td>
<td>positive_unate</td>
<td></td>
<td></td>
<td>false</td>
</tr>
</tbody>
</table>
```

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.
Exporting Reference Library Content

You can export reference library content to several formats for use with other tools. Table 2-9 lists the supported formats and the command used to generate the output files.

Table 2-9  Reference Library Export Formats

<table>
<thead>
<tr>
<th>Format</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technology file</td>
<td>write_tech_file</td>
</tr>
<tr>
<td>Antenna properties in Cell Library Format (CLF)</td>
<td>write_clf_antenna_properties</td>
</tr>
<tr>
<td>LEF</td>
<td>write_lef</td>
</tr>
<tr>
<td>GDSII</td>
<td>write_gds</td>
</tr>
<tr>
<td>OASIS</td>
<td>write_oasis</td>
</tr>
<tr>
<td>Design Compiler Graphical reference library</td>
<td>export_dc_fram</td>
</tr>
</tbody>
</table>

Closing a Reference Library

To close a reference 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 reference 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:

icc2_lm_shell> close_lib

To close the myreflib.ndm reference library, use the following command:

icc2_lm_shell> close_lib myreflib.ndm

To close all reference libraries, use the following command:

icc2_lm_shell> close_lib -all
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
• 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 reference library creation process. Table A-1 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 A-1  lib.workspace Application Options

<table>
<thead>
<tr>
<th>Application option</th>
<th>Valid values or data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lib.workspace.allow_commit_workspace_overwrite</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>lib.workspace.allow_missing_related_pg_pins</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>lib.workspace.create_workspace_tf_verbose</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>lib.workspace.enable_secondary_pg_marking</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>lib.workspace.exclude_design_filters</td>
<td>list</td>
<td>(default = &quot;&quot;)</td>
</tr>
</tbody>
</table>
### Table A-1  `lib.workspace` Application Options (Continued)

<table>
<thead>
<tr>
<th>Application option</th>
<th>Valid values or data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>lib.workspace.explore_create_aggregate</code></td>
<td>`false</td>
<td>true`</td>
</tr>
<tr>
<td><code>lib.workspace.group_libs_create_slg</code></td>
<td>`false</td>
<td>true`</td>
</tr>
<tr>
<td><code>lib.workspace.group_libs_fix_cell_shadowing</code></td>
<td>`true</td>
<td>false`</td>
</tr>
<tr>
<td><code>lib.workspace.group_libs_macro_grouping_strategy</code></td>
<td>`minimize_libs</td>
<td>single_cell_per_lib`</td>
</tr>
</tbody>
</table>
### lib.workspace Application Options (Continued)

<table>
<thead>
<tr>
<th>Application option</th>
<th>Valid values or data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lib.workspace.group_libs_naming_strategies</td>
<td>list</td>
<td>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.</td>
</tr>
<tr>
<td>lib.workspace.group_libs_physical_only_name</td>
<td>string</td>
<td>Specifies the name of the physical-only library group created by the group_libs command. If this application option has a value of &quot;&quot;, the command names the physical-only subworkspace &lt;root_workspace&gt;_physical_only. For more information, see Performing Automated Library Analysis.</td>
</tr>
<tr>
<td>lib.workspace.include_design_filters</td>
<td>list</td>
<td>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.</td>
</tr>
<tr>
<td>lib.workspace.remove_frame_bus_properties</td>
<td>false</td>
<td>true</td>
</tr>
</tbody>
</table>
### LEF Input Application Options

The LEF input (file.lef) application options control the behavior of the read_lef command. Table A-2 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 A-2  file.lef Application Options

<table>
<thead>
<tr>
<th>Application option</th>
<th>Valid values or data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file.lef.allow_empty_pin</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>file.lef.auto_rename_conflict_sites</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>file.lef.non_real_cut_obs_mode</td>
<td>true</td>
<td>false</td>
</tr>
</tbody>
</table>
The `read_gds` and `read_oasis` commands support similar application options to control their behavior. Table A-3 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 A-3  file.gds Application Options**

<table>
<thead>
<tr>
<th>Application option</th>
<th>Valid values or data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>file.format.create_custom_via</code></td>
<td>`false</td>
<td>true`</td>
</tr>
<tr>
<td><code>file.format.exclude_layers</code></td>
<td><code>list</code> (default = <code>&quot;&quot;</code>)</td>
<td>Specifies the rules for mapping between the input layers and the layers that are used to exclude geometries from the specified input layers.</td>
</tr>
<tr>
<td><code>file.format.port_type_map</code></td>
<td><code>list</code> (default = <code>&quot;&quot;</code>)</td>
<td>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 Reading GDSII or OASIS Files.</td>
</tr>
</tbody>
</table>
**Table A-3  file.gds Application Options (Continued)**

<table>
<thead>
<tr>
<th>Application option</th>
<th>Valid values or data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file.format.text_layer_map</td>
<td>list (default = &quot;&quot;)</td>
<td>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.</td>
</tr>
<tr>
<td>file.format.trace_unmapped_text</td>
<td>false</td>
<td>true</td>
</tr>
</tbody>
</table>
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 A-4 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>
<thead>
<tr>
<th>Application option</th>
<th>Valid values or data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lib.logic_model.auto_remove_incompatible_timing_designs</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>lib.logic_model.auto_remove_timing_only_designs</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>lib.logic_model.relax_pin_equal_opposite_attr_check</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>lib.logic_model.resolve_voltage_range_differences</td>
<td>use_base_library</td>
<td>require_same</td>
</tr>
</tbody>
</table>
The frame view generation (lib.physical_model) application options control the frame view generation, which is performed by the check_workspace command. Table A-5 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 A-5  lib.physical_model Application Options

<table>
<thead>
<tr>
<th>Application option</th>
<th>Valid values or data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lib.physical_model.block_all</td>
<td>auto</td>
<td>false</td>
</tr>
</tbody>
</table>

### 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 A-5 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 A-5  lib.physical_model Application Options (Continued)

<table>
<thead>
<tr>
<th>Application option</th>
<th>Valid values or data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lib.logic_model.require_same_opt_attrs</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>lib.logic_model.use_db_rail_names</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>Application option</td>
<td>Valid values or data type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------------------------</td>
<td>----------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>lib.physical_model.block_core_margin</td>
<td>list of layer-float pairs (default = &quot;&quot;)</td>
<td>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.</td>
</tr>
<tr>
<td>lib.physical_model.connect_within_pin</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>lib.physical_model.convert_metal_blockage_to_zero_spacing</td>
<td>list of layer-distance pairs (default = &quot;&quot;)</td>
<td>Converts metal blockages in the design views to zero-spacing routing blockages in the frame views 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.</td>
</tr>
<tr>
<td>lib.physical_model.create_frame_for_subblocks</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>lib.physical_model.create_zero_spacing_blockages_around_pins</td>
<td>list of layer-width pairs (default = &quot;&quot;)</td>
<td>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.</td>
</tr>
</tbody>
</table>
Table A-5  

<table>
<thead>
<tr>
<th>Application option</th>
<th>Valid values or data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>lib.physical_model.design_rule_via_blockage_layers</code></td>
<td>list of via layers (default = &quot;&quot;)</td>
<td>Converts zero-spacing routing blockages on the via layers of the frame view to design rule routing blockages.</td>
</tr>
<tr>
<td><code>lib.physical_model.drc_distances</code></td>
<td>list of layer-distance pairs (default = &quot;&quot;)</td>
<td>Specifies the distance from the boundary of blockages within with to preserve detail shapes.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td><code>lib.physical_model.hierarchical</code></td>
<td><code>true</code></td>
<td><code>false</code></td>
</tr>
<tr>
<td><code>lib.physical_model.include_nondefault_via</code></td>
<td>list (default = &quot;)&quot;)</td>
<td>Specifies the nondefault vias to include for via region extraction.</td>
</tr>
<tr>
<td><code>lib.physical_model.include_routing_pg_ports</code></td>
<td>list (default = &quot;&quot;)</td>
<td>Specifies the PG ports that requires via regions and access edges in the frame views.</td>
</tr>
<tr>
<td><code>lib.physical_model.keepout_spacing_for_non_pin_shapes</code></td>
<td>list of layer-spacings pairs (default = &quot;&quot;)</td>
<td>Defines how to create the zero-spacing routing blockages for the non-pin shapes on each affected layer.</td>
</tr>
<tr>
<td><code>lib.physical_model.merge_metal_blockage</code></td>
<td><code>false</code></td>
<td><code>true</code></td>
</tr>
</tbody>
</table>
Table A-5  *lib.physical_model* Application Options (Continued)

<table>
<thead>
<tr>
<th>Application option</th>
<th>Valid values or data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lib.physical_model.pin_must_connect_area_layers</td>
<td>list of layer-spare layer pairs (default = &quot;&quot;)</td>
<td>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.</td>
</tr>
<tr>
<td>lib.physical_model.pin_must_connect_area_thresholds</td>
<td>list of layer-threshold pairs (default = &quot;&quot;)</td>
<td>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.</td>
</tr>
<tr>
<td>lib.physical_model.port_contact_selections</td>
<td>string (default = &quot;&quot;)</td>
<td>Specifies custom contact codes for specific ports for via region generation.</td>
</tr>
<tr>
<td>lib.physical_model.preserve_metal_blockage</td>
<td>auto</td>
<td>true</td>
</tr>
</tbody>
</table>
Table A-5  *lib.physical_model* Application Options (Continued)

<table>
<thead>
<tr>
<th>Application option</th>
<th>Valid values or data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>lib.physical_model.remove_non_pin_shapes</code></td>
<td>list of layer-mode pairs</td>
<td>Specifies whether to remove non-pin shapes within the DRC distance of a block boundary when generating the frame view. Valid values for the mode are <code>none</code> or <code>all</code>. If you do not specify a mode for a layer, the shapes are not removed (<code>none</code>). Removing these 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 must create a core blockage by using the <code>lib.physical_model.block_all</code> or <code>lib.physical_model.block_core_margin</code> application option.</td>
</tr>
<tr>
<td><code>lib.physical_model.source_drain_annotation</code></td>
<td>string</td>
<td>Specifies the name of the input file containing the source-drain annotation information for the cells in the library workspace.</td>
</tr>
<tr>
<td><code>lib.physical_model.trim_metal_blockage_around_pin</code></td>
<td>list of layer-method pairs</td>
<td>Specifies how to trim metal blockages around pins for each layer. Valid values for the method are <code>touch</code>, <code>all</code>, and <code>none</code>. 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 <code>lib.physical_model.preserve_metal_blockage</code> application option.</td>
</tr>
</tbody>
</table>
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 A-6 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 A-6  signoff.antenna Application Options

<table>
<thead>
<tr>
<th>Application option</th>
<th>Valid values or data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>signoff.antenna.contact_layer</td>
<td>list (default = &quot;&quot;)</td>
<td>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.</td>
</tr>
<tr>
<td>signoff.antenna.contact_layers_between_m0_diffusion</td>
<td>list (default = &quot;&quot;)</td>
<td>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.</td>
</tr>
<tr>
<td>signoff.antenna.diffusion_layer</td>
<td>list (default = &quot;&quot;)</td>
<td>Specifies the diffusion layers. Specify the layers by using the layer names from the technology file.</td>
</tr>
<tr>
<td>signoff.antenna.enabled</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>signoff.antenna.extract_via_antenna_property</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>signoff.antenna.gate_class1_layers</td>
<td>list (default = &quot;&quot;)</td>
<td>Specifies the marking layers for gate thickness class 1. Specify the layers by using the layer names from the technology file.</td>
</tr>
<tr>
<td>Application option</td>
<td>Valid values or data type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------------------------------------------</td>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>signoff.antenna.gate_class2_layers</td>
<td>list (default = &quot;&quot;)</td>
<td>Specifies the marking layers for gate thickness class 2. Specify the layers by using the layer names from the technology file.</td>
</tr>
<tr>
<td>signoff.antenna.gate_class3_layers</td>
<td>list (default = &quot;&quot;)</td>
<td>Specifies the marking layers for gate thickness class 3. Specify the layers by using the layer names from the technology file.</td>
</tr>
<tr>
<td>signoff.antenna.m0_layers_for_diffusion_connection</td>
<td>string (default = &quot;&quot;)</td>
<td>Specifies the m0 layers for diffusion connection. Specify the layers by using the layer names from the technology file.</td>
</tr>
<tr>
<td>signoff.antenna.m0_layers_for_poly_connection</td>
<td>string (default = &quot;&quot;)</td>
<td>Specifies the m0 layers for poly connection. Specify the layers by using the layer names from the technology file.</td>
</tr>
<tr>
<td>signoff.antenna.poly_layer</td>
<td>list (default = &quot;&quot;)</td>
<td>Specifies the polysilicon layers. Specify the layers by using the layer names from the technology file.</td>
</tr>
<tr>
<td>signoff.antenna.report_diodes</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>signoff.antenna.top_cell_pin_only</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>signoff.antenna.treat_source_drain_as_diodes</td>
<td>false</td>
<td>true</td>
</tr>
</tbody>
</table>
### Table A-6 signoff.antenna Application Options (Continued)

<table>
<thead>
<tr>
<th>Application option</th>
<th>Valid values or data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>signoff.antenna.user_defined_options</td>
<td>list (default = &quot;&quot;)</td>
<td>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.</td>
</tr>
<tr>
<td>signoff.antenna.v0_layers_between_ml_m0</td>
<td>list (default = &quot;&quot;)</td>
<td>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.</td>
</tr>
</tbody>
</table>
The shell application options control various aspects of the library manager command-line interface. Table A-7 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 A-7 shell Application Options**

<table>
<thead>
<tr>
<th>Application option</th>
<th>Valid values or data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>shell.common.collection_result_display_limit</td>
<td>integer (default = 100)</td>
<td>Specifies the maximum number of results displayed when the return value is a collection.</td>
</tr>
<tr>
<td>shell.common.enable_line_editing</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>shell.common.line_editing_mode</td>
<td>emacs</td>
<td>vi</td>
</tr>
<tr>
<td>shell.common.man_path</td>
<td>string</td>
<td>Specifies the search path for the man pages.</td>
</tr>
<tr>
<td>shell.common.report_default_significant_digits</td>
<td>integer (default = 2)</td>
<td>Specifies the default number of significant digits used to display values in reports.</td>
</tr>
<tr>
<td>shell.common.single_line_messages</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>shell.common.tmp_dir_path</td>
<td>string (default = /tmp)</td>
<td>Specifies the directory that the tool uses for temporary storage.</td>
</tr>
</tbody>
</table>
Graphical User Interface Application Options

The gui application options control various aspects of the library manager GUI. Table A-8 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 A-8   gui Application Options

<table>
<thead>
<tr>
<th>Application option</th>
<th>Valid values or data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gui.auto_link_blocks</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>gui.auto_open_layout_windows</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>gui.command_form_close_dialog_after_run_or_script</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>gui.command_form_show_result_dialog_after_run</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>gui.custom_setup_files</td>
<td>string (default = &quot;&quot;)</td>
<td>Specifies the files to source when invoking the GUI. These files are sourced in addition to the standard setup files.</td>
</tr>
<tr>
<td>gui.enable_custom_setup_files</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>gui.layer_solid_fill_pattern</td>
<td>string (default = Dense4Pattern)</td>
<td>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.</td>
</tr>
</tbody>
</table>
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 Message Browser Window
Starting the Tool in the GUI

To start the IC Compiler II Library Manager tool in the GUI,

1. Set the DISPLAY environment variable to the name of your UNIX or Linux system display.

2. Include the path to the bin directory in your $PATH variable.

3. Enter the `icc2_lm_shell` command with the `-gui` option in a UNIX or Linux shell.
   
   ```shell
   % icc2_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 `icc2_lm_shell` command

When you start the library manager in the GUI, it

1. Runs the commands in the `.synopsys_icc2_lm.setup` and `.synopsys_icc2_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 icc2_lm_shell without the GUI, you can open the GUI at any time by entering the gui_start command at the icc2_lm_shell> prompt.

icc2_lm_shell> gui_start

If you want to execute a script before opening the GUI, use the -file option. For example,

icc2_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 UNIX or Linux system display.

When you open the library manager GUI, it

1. Runs the commands in the .synopsys_icc2_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 reference library is open, it opens the library browser in the main window.
     The library browser is a hierarchical representation of the library workspace or reference library.
   - if a library workspace or reference 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 in icc2_lm_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.

icc2_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 IC Compiler II Graphical User Interface User Guide.

See Also

• Exiting the IC Compiler II 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 reference libraries. Figure B-1 shows the start page of the library preparation wizard.

Figure B-1  Library Preparation Wizard

The following topics describe how to use the wizard to perform these tasks:

• Creating a Reference Library
• Modifying a Reference Library
• Viewing a Reference Library
Creating a Reference Library

To use the library preparation wizard to create a new reference library,

1. Click the create icon ( ) or the “Create a new library” link.
   
   The “Set up ‘Exploration’ Workspace dialog box (Figure B-2) appears.

   **Figure B-2  Set up “Exploration” Workspace Dialog Box**

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 Libraries” and “Loading the Physical Libraries.”
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 B-3.

Figure B-3  Library Browser After Loading Source Libraries
7. Configure the PVT filtering.

The library manager displays the current PVT configuration, as well as the filtering statistics, as shown in Figure B-4.

**Figure B-4  PVT Configuration**

![PVT Configuration diagram]

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 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 B-5.
9. Validate the workspace by selecting Check Workspace and then clicking OK in the Check Workspace dialog box.

   The workspace must pass validation before you can create a reference library. For details about workspace validation, see “Validating the Workspace.”

10. Create the reference library by selecting Commit Workspace and then clicking OK in the Commit Workspace dialog box.
Modifying a Reference Library

To use the library preparation wizard to modify an existing reference library,

1. Click the edit icon ( ) or the “Edit a library” link.

The “Set up ‘Edit’ Workspace” dialog box (Figure B-6) appears.

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 reference 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 Reference 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 reference library. For details about workspace validation, see Validating the Workspace.

6. Save the updated reference library by selecting Commit Workspace and then clicking OK in the Commit Workspace dialog box.

---

**Viewing a Reference Library**

You can use the GUI to list the cells in a reference library, view information about the cells, and view the layout of the cells.

To use the library preparation wizard to view an existing reference library,

1. Click the folder icon ( ) or the “View an existing library” link.

   The Open Library dialog box appears.

2. Select the reference library.

   You can either enter the path to the reference library or click to browse for it.

3. If you want to edit the reference library, select “Open reference library for edit.”

4. Click OK.

   The library browser opens with the selected reference library.

**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 B-7 identifies the major features of the main window.

Figure B-7  Main Window Features

Typically, you use the main window to

- Run Tcl scripts
- Enter icc2_lm_shell commands, monitor command processing, and view messages in the console
- Open and close libraries and blocks (library cells or designs)
- Explore reference 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 reference 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 reference 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 B-8 shows a library browser that displays an expanded reference library with one of its cells expanded to show its views.

Figure B-8  Library Browser

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 the IC Compiler II Graphical User Interface User Guide.

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 B-9 shows these features in the library browser. Figure B-10 shows an example cell information report.

**Figure B-9  Displaying Cell Information in the Library Browser**

![Displaying Cell Information in the Library Browser](image)

**Figure B-10  Cell Information Report**

<table>
<thead>
<tr>
<th>Cells</th>
<th>Views</th>
<th>Inputs</th>
<th>Outputs</th>
<th>Power</th>
<th>Ground</th>
<th>Site</th>
<th>Type</th>
<th>Boundary</th>
<th>Antenna</th>
</tr>
</thead>
<tbody>
<tr>
<td>saed2hvt_if_c/XOR3X2_HVT</td>
<td>f, t</td>
<td>3</td>
<td>1</td>
<td>VDD</td>
<td>VSS</td>
<td>unit</td>
<td>lib_cell</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>saed2hvt_if_c/XOR3X1_HVT</td>
<td>f, t</td>
<td>3</td>
<td>1</td>
<td>VDD</td>
<td>VSS</td>
<td>unit</td>
<td>lib_cell</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>saed2hvt_if_c/XOR2X2_HVT</td>
<td>f, t</td>
<td>2</td>
<td>1</td>
<td>VDD</td>
<td>VSS</td>
<td>unit</td>
<td>lib_cell</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>saed2hvt_if_c/XOR2X1_HVT</td>
<td>f, t</td>
<td>2</td>
<td>1</td>
<td>VDD</td>
<td>VSS</td>
<td>unit</td>
<td>lib_cell</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>saed2hvt_if_c/XNOR3X2_HVT</td>
<td>f, t</td>
<td>3</td>
<td>1</td>
<td>VDD</td>
<td>VSS</td>
<td>unit</td>
<td>lib_cell</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>saed2hvt_if_c/XNOR3X1_HVT</td>
<td>f, t</td>
<td>3</td>
<td>1</td>
<td>VDD</td>
<td>VSS</td>
<td>unit</td>
<td>lib_cell</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Using the Message Browser Window

After you run the `check_workspace` command, you can view the messages in the message browser. To view the messages, load the message database by choosing File > Open Message Database, and then select the `check_workspace.ems` file. The message browser lists each of the messages (information, warning, and error) that was issued by the `check_workspace` command. To display the man page for a message, select the message in the browser view. Figure B-11 shows an example of a message browser window.

![Message Browser](image-url)

Figure B-11  Message Browser
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 C-1 shows the use of these attributes to define the power connections in a .lib logic library file.

**Example C-1  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 reference 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:

```shell
icc2_lm_shell> rename_rail -library A -from VDD1.0 -to VDD
icc2_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.