

# **IC Validator User Guide**

---

Version M-2016.12-SP2, March 2017

**SYNOPSYS®**

# Copyright Notice and Proprietary Information

©2017 Synopsys, Inc. All rights reserved. This Synopsys software and all associated documentation are proprietary to Synopsys, Inc. and may only be used pursuant to the terms and conditions of a written license agreement with Synopsys, Inc. All other use, reproduction, modification, or distribution of the Synopsys software or the associated documentation is strictly prohibited.

## Destination Control Statement

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

## Disclaimer

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

## Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at <http://www.synopsys.com/Company/Pages/Trademarks.aspx>. All other product or company names may be trademarks of their respective owners.

## Third-Party Links

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

Synopsys, Inc.  
690 E. Middlefield Road  
Mountain View, CA 94043  
[www.synopsys.com](http://www.synopsys.com)

# Contents

---

About This User Guide . . . . .	xiv
Customer Support. . . . .	xvi
<b>1. IC Validator Basics</b>	
Using the IC Validator Tool in Your Design Flow. . . . .	1-2
Running the IC Validator Tool Within the IC Compiler Tool . . . . .	1-2
Running the IC Validator Tool Within the IC Compiler II Tool. . . . .	1-3
Running the IC Validator Tool With StarRC Parasitic Extraction . . . . .	1-4
Running the IC Validator Tool From the Custom Compiler Platform . . . . .	1-5
Setting Up and Running DRC. . . . .	1-5
Setting Up and Running LVS . . . . .	1-5
Command-Line Options . . . . .	1-6
General Command-Line Options. . . . .	1-7
Compiler Directives. . . . .	1-28
Support for 64-Bit Coordinates . . . . .	1-28
Run-Only Execution. . . . .	1-31
Environment Variables. . . . .	1-32
Providing Additional Information to the Compiler. . . . .	1-32
Creating Preprocessor Definitions from Environment Variables . . . . .	1-33
Pruning Layers. . . . .	1-34
Exit Codes . . . . .	1-34
Inlined Functions. . . . .	1-35
Distributed Processing . . . . .	1-38

License Requirements . . . . .	1-38
Using Multiple Distributed Processing Hosts . . . . .	1-39
Resolving Automounter Issues . . . . .	1-40
Distributed Processing Options Configuration File . . . . .	1-40
Distributed Processing Learn . . . . .	1-41
Runset Caching . . . . .	1-42
Cache Files . . . . .	1-42
Configuring Runset Caching . . . . .	1-43
Matching a Runset to a Cache File . . . . .	1-45
Using IC Validator Command-Line Options With Runset Caching and DP Learn . .	1-45
Compatible Command-Line Options . . . . .	1-46
Command-Line Options That Affect Runset Caching and DP Learn . . . . .	1-46
Incompatible Command-Line Options . . . . .	1-47
Native LSF and SGE Support . . . . .	1-47
Layer Mapping . . . . .	1-49
Runset Layer Mapping . . . . .	1-49
Data Layer Mapping . . . . .	1-51
Using PXL Runset Encryption . . . . .	1-53
 <b>2. Licensing and Resource Requirements</b>	
Licensing Schemes . . . . .	2-2
License Waiting . . . . .	2-2
Which IC Validator Licenses Do I Need? . . . . .	2-3
Native Licensing . . . . .	2-3
Multicore Licensing . . . . .	2-4
Shared Licensing . . . . .	2-4
HERCULES_DEVICE . . . . .	2-5
HERCULES_DRC . . . . .	2-5
HERCULES_ERC . . . . .	2-6
HERCULES_LVS . . . . .	2-6
HERCULES_MASK . . . . .	2-6
NET-TRAN . . . . .	2-8
HERCULES-NETLIST . . . . .	2-9
HERCULES-DP_MT . . . . .	2-9
Resource Requirements . . . . .	2-9

Space Requirements .....	2-9
Memory Usage .....	2-10
Runtimes and Memory Use Report .....	2-12
Operating Systems .....	2-15
System Limits .....	2-15
Networked Environments .....	2-15

### 3. Output Files

Description of Output Files .....	3-2
General DRC and LVS Output Files .....	3-2
Results File .....	3-3
Error File .....	3-6
Summary File .....	3-7
Rules File .....	3-7
Technology File .....	3-8
Tree Structure Output File .....	3-9
Virtual Cell File .....	3-9
Pair Section .....	3-9
Set Section .....	3-9
Explode Section .....	3-9
Distributed Log File .....	3-9
Distributed Statistics File .....	3-10
Distributed Work Directory .....	3-10
Graphical Error Structures .....	3-10
LVS-Specific Output Files .....	3-11
LVS Debugging File .....	3-12
Extracted Layout Netlist File .....	3-13
Extracted Layout SPICE Netlist File .....	3-13
Account File .....	3-13
Bounding Box File .....	3-13
Equivalence Files .....	3-13
Device Leveling Summary File .....	3-14
Compare Log File .....	3-14
Compare Tree Files .....	3-14
Compare Directory Output Files .....	3-15
Individual Equivalence Summary File .....	3-15

Partitioned Schematic Netlist . . . . .	3-16
Partitioned Layout Netlist . . . . .	3-16
Editing Errors . . . . .	3-16
<b>4. PXL</b>	
PXL Architecture . . . . .	4-2
General Definitions . . . . .	4-3
Identifiers . . . . .	4-3
String Literals . . . . .	4-5
Numeric Constants . . . . .	4-6
Keywords . . . . .	4-6
Operators . . . . .	4-22
Comments . . . . .	4-22
Environment Variable Reference . . . . .	4-23
Variables and Operators . . . . .	4-24
Variables . . . . .	4-24
Initializing Variables . . . . .	4-24
Scope of Variables . . . . .	4-25
Operators . . . . .	4-26
Assignment Operators . . . . .	4-27
Relational Operators . . . . .	4-27
Logical Operators . . . . .	4-28
Conditional Operator . . . . .	4-28
Numeric and Bitwise Operators . . . . .	4-29
Operators with Double-Only Outputs . . . . .	4-30
Member Reference . . . . .	4-30
Precedence and Associativity . . . . .	4-31
Data Types and Typing . . . . .	4-32
Supported Data Types . . . . .	4-33
Primitive Types . . . . .	4-33
User-Defined Structures and Types . . . . .	4-36
Type Analysis . . . . .	4-44
Standard Arithmetic Conversions . . . . .	4-45
Comparing Doubles . . . . .	4-45
Type Promotion . . . . .	4-45
Parameter Passing . . . . .	4-46
String Concatenation . . . . .	4-46

Default Injection . . . . .	4-47
Nested Containers . . . . .	4-47
Flow Control . . . . .	4-49
Conditional Statements . . . . .	4-49
Loops . . . . .	4-50
for Loop . . . . .	4-50
foreach Loop . . . . .	4-51
while Loop . . . . .	4-51
Functions . . . . .	4-52
Function Definitions . . . . .	4-52
Function Prototypes . . . . .	4-53
Function Calls . . . . .	4-54
Standard Function Call . . . . .	4-55
Operator-Style Function Call . . . . .	4-56
Argument Bindings . . . . .	4-57
Defaults . . . . .	4-57
Function Overloading . . . . .	4-58
Function Recursion . . . . .	4-58
Program Structure . . . . .	4-59
Compound Statement . . . . .	4-59
Violations . . . . .	4-59
Violation Comments . . . . .	4-60
Violation-Producing Functions . . . . .	4-61
Violation Variables . . . . .	4-61
Violation Blocks . . . . .	4-62
Execution Model . . . . .	4-63
Preprocessor . . . . .	4-63
Predefined Identifiers . . . . .	4-65
Release Version Macros . . . . .	4-66
Date Comparison Macros . . . . .	4-67
Qualifiers . . . . .	4-68
Example of Modularized Code . . . . .	4-70
<b>5. Dynamic-Link Library Support</b>	
Dynamic-Link Functions . . . . .	5-2
Using the Dynamic-Link Functions . . . . .	5-2

**6. Unified Fill**

Overview . . . . .	6-3
Types of Fill Patterns . . . . .	6-4
Polygon Fill Patterns . . . . .	6-5
Single Layer Pattern Definition With Multiple Rectangles. . . . .	6-5
Multilayer Pattern Definition . . . . .	6-6
Multiple Fill Pattern Definition . . . . .	6-7
Adjustable Rectangle Fill Patterns. . . . .	6-8
Stack Fill Patterns . . . . .	6-9
Cell Fill Patterns. . . . .	6-14
Expandable Fill Cell Patterns. . . . .	6-16
Stripe Fill Patterns . . . . .	6-18
Fill-to-Fill Spacing . . . . .	6-20
Spacing Between Instances of the Same Patterns . . . . .	6-20
Spacing Between Different Patterns . . . . .	6-23
Target Region and Fill-to-Signal Spacing . . . . .	6-26
Range and Width Dependent Spacing . . . . .	6-31
Polygon Grouping . . . . .	6-32
Signal Aligned Fills . . . . .	6-34
Defining the Fill Pattern for Signal Aligned Fills. . . . .	6-37
Improving Pattern Insertion. . . . .	6-38
Pitch Aligned Fills . . . . .	6-38
Criteria Analysis . . . . .	6-41
Output Layers . . . . .	6-46
Layer Compression. . . . .	6-48

**7. Working With Edges**

Associating Edge Layers With a Parent Layer . . . . .	7-2
Topological Operations . . . . .	7-2
Logical Operations . . . . .	7-3
Edge Manipulation . . . . .	7-4



## 8. DRC Error Classification

Using DRC Error Classification . . . . .	8-2
Hierarchy Considerations in CELL_LEVEL Mode . . . . .	8-2
Hierarchical DRC Error Classification . . . . .	8-3
Error Classification Best Practices . . . . .	8-3
Database Naming Conventions . . . . .	8-4
DRC Error Classification Flow . . . . .	8-4
DRC Error Classification Flow Example . . . . .	8-6
Running the IC Validator Tool Initially . . . . .	8-6
Creating an Error Classification Database . . . . .	8-8
Running the IC Validator Tool Again . . . . .	8-10
Updating the Existing Error Classification Database . . . . .	8-11
pydb_report Utility . . . . .	8-13
Classify File Format . . . . .	8-15
Using Regular Expression Matching . . . . .	8-16
Examples . . . . .	8-17
pydb_export Utility . . . . .	8-17
Error Classification Database Format . . . . .	8-21
Purge File Format . . . . .	8-21
Examples . . . . .	8-22

## 9. Pattern Matching

Overview . . . . .	9-2
Using the Pattern Matching Flow . . . . .	9-3
Creating the Pattern Library . . . . .	9-4
Preparing the Input Layout . . . . .	9-4
Required Input . . . . .	9-4
Optional Input . . . . .	9-5
Additional Pattern Specifications . . . . .	9-6
Example of Creating a Pattern Library . . . . .	9-7
Examining the Output of Pattern Library Creation . . . . .	9-10
Viewing and Locking a Pattern Library . . . . .	9-12
Checking a Layout . . . . .	9-13
Runset Example of Pattern Matching . . . . .	9-13
Examining Pattern Matching Output . . . . .	9-15
Post Pattern-Matching Operations . . . . .	9-16
Detecting and Repairing Hotspots Within the IC Compiler Tool . . . . .	9-17
Setting Up the Physical Signoff Options . . . . .	9-18

Running Pattern Matching for Hotspot Detection . . . . .	9-18
Automatically Fixing Hotspot Patterns . . . . .	9-18
Repairing Hotspots With User-Defined Fix Guidance. . . . .	9-18

## **Appendix A. IC Validator Architecture**

Basic Architecture. . . . .	A-2
-----------------------------	-----

## **Appendix B. LVL Utility**

Description . . . . .	B-3
Command-Line Options . . . . .	B-3
Using the OpenAccess Library Format. . . . .	B-11
OpenAccess Layer Mapping File. . . . .	B-13
Cell Mapping . . . . .	B-13
Error Output Format for OpenAccess Baseline Library . . . . .	B-14
Setting the Cell View Name . . . . .	B-14
Layer Files . . . . .	B-15
Selecting a Subset of Layers. . . . .	B-15
Mapping Layers. . . . .	B-16
Comparing a Subset of Layers . . . . .	B-17
Comparing by Name . . . . .	B-18
Comparing by Number. . . . .	B-18
Handling of Text . . . . .	B-19
LVL Output . . . . .	B-20
Exit Status . . . . .	B-21
Error Files . . . . .	B-21
Output Directories . . . . .	B-22
Use Models. . . . .	B-23
QuickLVL Utility. . . . .	B-24
Cell-Level Quick LVL . . . . .	B-25
Differences Between LVL and QuickLVL . . . . .	B-28
Auto-Incremental DRC Focused on Differing Layers and Regions . . . . .	B-29

## **Appendix C. Layout Integrity Management**

Using Layout Integrity Management . . . . .	C-2
---	-----

Layout Integrity Management Examples . . . . .	C-3
Basic Example . . . . .	C-3
Running on a Different Layout Format. . . . .	C-4
Reporting of Mismatches From a Merged LIDB . . . . .	C-6
Hierarchical Mode . . . . .	C-6
Hierarchical Cell Check Example . . . . .	C-7
Hierarchical Marker Layer Check Example . . . . .	C-9
Marker Layer Check . . . . .	C-12
Mixed Milkyway and GDSII Flow . . . . .	C-14
The icv_lidb Utility . . . . .	C-15
Command-Line Options. . . . .	C-15
Exit Status . . . . .	C-17
Output Files . . . . .	C-18
The icv_lidb_report Utility . . . . .	C-18
Command-Line Options. . . . .	C-18
Output Files . . . . .	C-19

## Appendix D. IC Validator PXL Debugger

Running the PXL Debugger . . . . .	D-2
PXL Debugger Commands. . . . .	D-3
break . . . . .	D-5
cont . . . . .	D-7
delete. . . . .	D-8
disable . . . . .	D-9
enable . . . . .	D-10
finish . . . . .	D-11
info break. . . . .	D-12
list . . . . .	D-13
next . . . . .	D-15
print . . . . .	D-16
quit. . . . .	D-17
record. . . . .	D-18
source . . . . .	D-20
step . . . . .	D-21
where. . . . .	D-22

**Appendix E. PYDB Perl API**

PYDB Perl API Overview . . . . .	E-2
Accessing the PYDB. . . . .	E-3
Examples of Common Tasks . . . . .	E-4
PYDB Schema . . . . .	E-6

**Appendix F. Third-Party Licenses**

Licensing Overview. . . . .	F-3
Third-Party Software. . . . .	F-4
ANTLR. . . . .	F-4
Boost . . . . .	F-4
cx_Freeze . . . . .	F-5
libdp . . . . .	F-6
libxml2 . . . . .	F-7
MariaDB Connector . . . . .	F-8
MariaDB. . . . .	F-14
MCPD Public Domain Code. . . . .	F-19
patchELF . . . . .	F-19
Python 2.7 License . . . . .	F-31
Shroud-1.0. . . . .	F-32
SQLite . . . . .	F-40
TclLib . . . . .	F-40
Tcl/Tk . . . . .	F-41
zlib . . . . .	F-42

**Index**

# Preface

---

This preface includes the following sections:

- [About This User Guide](#)
- [Customer Support](#)

---

## About This User Guide

This user guide describes PXL, the IC Validator architecture, command-line options, and other basic information.

---

## Audience

This user guide is designed to enhance both beginning and advanced users' knowledge of PXL, the IC Validator architecture, and command-line options.

---

## Related Publications

For additional information about the IC Validator tool, see the documentation on the SolvNet<sup>®</sup> 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:

- Custom Compiler<sup>™</sup>
- IC Compiler<sup>™</sup>
- IC Compiler II<sup>™</sup>
- StarRC<sup>™</sup>

---

## Release Notes

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

To see the *IC Validator Release Notes*,

1. Go to the SolvNet Download Center located at the following address:  
<https://solvnet.synopsys.com/DownloadCenter>
2. Select IC Validator, and then select a release in the list that appears.

---

## Conventions

The following conventions are used in Synopsys documentation.

Convention	Description
Courier	Indicates syntax, such as <code>write_file</code> .
<i>Courier italic</i>	Indicates a user-defined value in syntax, such as <code>write_file design_list</code> .
<b>Courier bold</b>	Indicates user input—text you type verbatim—in examples, such as <code>prompt&gt; write_file top</code>
[ ]	Denotes optional arguments in syntax, such as <code>write_file [-format fmt]</code>
...	Indicates that arguments can be repeated as many times as needed, such as <code>pin1 pin2 ... pinN</code>
	Indicates a choice among alternatives, such as <code>low   medium   high</code>
Ctrl+C	Indicates a keyboard combination, such as holding down the Ctrl 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 SolvNet, 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](mailto: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>



# 1

## IC Validator Basics

---

*This chapter gives a brief overview of the IC Validator physical verification tool. It also describes the IC Validator command-line options and distributed processing.*

The basics of the IC Validator tool are described in the following sections:

- [Using the IC Validator Tool in Your Design Flow](#)
- [Command-Line Options](#)
- [Compiler Directives](#)
- [Exit Codes](#)
- [Inlined Functions](#)
- [Distributed Processing](#)
- [Runset Caching](#)
- [Using IC Validator Command-Line Options With Runset Caching and DP Learn](#)
- [Native LSF and SGE Support](#)
- [Layer Mapping](#)
- [Using PXL Runset Encryption](#)

---

## Using the IC Validator Tool in Your Design Flow

All IC Validator runs are executed based on your IC Validator runset. You can execute the IC Validator tool using command-line execution, interactive execution using VUE, or from various other Synopsys tools, such as the StarRC or IC Compiler tools.

To run the IC Validator tool successfully in your design flow,

- Create and debug a runset file. The runset file includes the following information:
  - Layout database specifics
  - General options
  - Layer assignments
  - Database checks, such as snap and grid checks
  - Design rule checking (DRC), electrical rule checking (ERC), fill patterns, or layout-versus-schematic (LVS) checking
- Run IC Validator DRC and ERC on your layout database, verifying your design against the design rule document.
- Review and debug errors generated in the IC Validator DRC and ERC run using IC Validator error files or VUE.
- Run the IC Validator tool to perform fill-pattern generation, including basic DRC checks on the layers that are filled.
- Run IC Validator LVS checking.
- Review and debug errors generated in the IC Validator LVS run using IC Validator error files or VUE.

See [Appendix A, “IC Validator Architecture.”](#)

---

## Running the IC Validator Tool Within the IC Compiler Tool

You can run the IC Validator tool from inside the IC Compiler tool; this flow is called In-Design physical verification. For this flow, use the following IC Compiler commands:

- `signoff_drc`. Performs DRC checking using a foundry-qualified IC Validator DRC runset.
- `signoff_autofix_drc`. Performs automatic DRC repair. In this flow, the IC Validator tool.
  - Performs signoff DRC checking.

- Generates route guidance that assists the router in performing targeted repairs.
- Validates the repair done by the router.
- `signoff_metal_fill`. Inserts metal and via fill in either pattern-based mode, which uses a foundry-qualified IC Validator metal-fill runset, or in track-based mode, which uses information from the technology file. The fill shapes are written into a FILL view of the Milkyway design.
- `signoff_fix_isolated_via`. Performs automatic fixing of isolated vias.
- `report_metal_density`. Calculates the metal density and density gradient values and annotates the values on the design.

**Note:**

You must make sure that the IC Validator tool can parse and run the runset before using it within the IC Compiler tool.

See the *IC Compiler Implementation User Guide*, which is available on SolvNet, or the IC Compiler man pages for detailed command syntax and usage.

---

## Running the IC Validator Tool Within the IC Compiler II Tool

You can run the IC Validator tool from inside the IC Compiler II tool; this flow is called In-Design physical verification. For this flow, use the following IC Compiler II commands:

- `signoff_check_drc`. Performs DRC checking using a foundry-qualified IC Validator DRC runset.
- `signoff_fix_drc`. Performs automatic DRC repair. In this flow, the IC Validator tool
  - Performs signoff DRC checking.
  - Generates route guidance that assists the router in performing targeted repairs.
  - Validates the repair done by the router.
- `signoff_create_metal_fill`. Inserts metal and via fill in either pattern-based mode, which uses a foundry-qualified IC Validator metal-fill runset, or in track-based mode, which uses information from the technology file. The fill shapes are written into the design view of the IC Compiler II input design.
- `signoff_fix_isolated_via`. Performs automatic fixing of isolated vias.

**Note:**

You must make sure that the IC Validator tool can parse and run the runset before using it within the IC Compiler II tool.

See the *IC Compiler II Implementation User Guide*, which is available on SolvNet, or the IC Compiler II man pages for detailed command syntax and usage.

## Running the IC Validator Tool With StarRC Parasitic Extraction

You can generate IC Validator data for input to the StarRC tool, using the following functions as part of the IC Validator runset:

- The functions for parasitic extraction (PEX) layer mapping are

- `pex_conducting_layer_map()`
- `pex_via_layer_map()`
- `pex_marker_layer_map()`
- `pex_remove_layer_map()`
- `pex_viewonly_layer_map()`
- `pex_ignore_cap_layer_map()`

These functions map IC Validator `polygon_layer` objects to StarRC TCAD\_GRD\_FILE layers to generate a layer mapping file for use with the StarRC tool.

- The function for generating a connected layout database and mapping files for input to StarRC parasitic extraction is

- `pex_generate_results()`

[Table 1-1](#) shows the LVS settings that are not supported for use in the IC Validator–StarRC flow. These settings should be set to `false`.

*Table 1-1 LVS Settings Not Supported for IC Validator–StarRC Flow*

Unsupported setting	Argument default	Required setting
<code>compare(memory_array_compare=true)</code>	<code>true</code>	<code>false</code>
<code>compare(push_down_devices=true)</code>	<code>false</code>	<code>false</code>
<code>compare(push_down_pins=true)</code>	<code>true</code>	<code>false</code>
<code>merge_series(multiple_paths=true)</code>	<code>false</code>	<code>false</code>

See the [IC Validator Reference Manual](#) for detailed syntax and usage of the parasitic extraction functions.

---

## Running the IC Validator Tool From the Custom Compiler Platform

You can run the IC Validator tool for DRC and LVS from the Custom Compiler platform using its Verification menu or from the Custom Compiler command line. See the Custom Compiler documentation, which is available on SolvNet, for more information.

### Setting Up and Running DRC

You can set up and run DRC from the Custom Compiler platform using its Verification menu:

1. Choose Verification > DRC > Setup and Run.
2. Select the IC Validator tool and specify all the required runtime parameters.
3. Click Apply or OK to run the IC Validator tool.

Note:

To view the output, select the View Output check box to open the Custom Compiler Text Viewer. Using tabs in the viewer, you can see the runset, results, errors, and a log of the run.

To run the IC Validator tool with parameters previously set in the Setup and Run menu, choose Verification > DRC > Run.

You can start an IC Validator VUE debugging session from the Custom Compiler platform.

- After an IC Validator run, to run VUE from the Custom Compiler platform, choose Verification > DRC > Debug.
- To run VUE when the Setup menu has not been set to ICV,
  - Set the preference `db::setPrefValue xtDRCTool -value icv`.
  - Choose Verification > DRC > Debug.

To open the Custom Compiler Text Viewer, which provides access to the runset, `cell.RESULTS`, `cell.LAYOUT_ERRORS`, and the `stdout.drc.log` files, choose Verification > DRC > View Output.

### Setting Up and Running LVS

You can set up and run LVS from the Custom Compiler platform using its Verification menu:

1. Choose Verification > LVS > Setup and Run.
2. Select the IC Validator tool and specify all the required runtime parameters.
3. Click Apply or OK to run the IC Validator tool.

**Note:**

To view the output, select the View Output check box to open the Custom Compiler Text Viewer. Using tabs in the viewer, you can see the runset, results, errors, and a log of the run.

To run the IC Validator tool with parameters previously set in the Setup and Run menu, choose Verification > LVS > Run.

You can start an IC Validator VUE debugging session from the Custom Compiler platform.

- After an IC Validator run, to run VUE from the Custom Compiler platform, choose Verification > LVS > Debug.
- To run VUE when the Setup menu has not been set to ICV,
  - Set the preference `db::setPrefValue xtLVSTool -value icv`.
  - Choose Verification > LVS > Debug.

To open the Custom Compiler Text Viewer, which provides access to the runset, `cell.RESULTS`, `cell.LAYOUT_ERRORS`, `cell.LVS_ERRORS`, and the `stdout.lvs.log` files, choose Verification > LVS > View Output.

---

## Command-Line Options

The IC Validator command-line syntax is

```
icv [command-line-options] runsetfile
```

where *runsetfile* specifies the executable runset file.

**Note:**

For netlist-versus-netlist flows the runset file is not needed on the command line.

If an error occurs, an exit code is reported in `cell.err`.

The command-line options are described in the following section:

- [General Command-Line Options](#)

## General Command-Line Options

The command-line options are described in [Table 1-2](#).

*Table 1-2 IC Validator Command-Line Options*

Option	Definition
-64	<p>Runs the IC Validator tool with 64-bit coordinates support. See <a href="#">“Support for 64-Bit Coordinates” on page 1-28</a> for more information.</p> <p>This option is available only with native licensing. See <a href="#">“Which IC Validator Licenses Do I Need?” in Chapter 2</a> for information about IC Validator licensing.</p>
-bct <i>option</i>	<p>Blocks forward data flow analysis through the outputs of specified commands. The selections for <i>option</i> are:</p> <ul style="list-style-type: none"> <li>• <b>extent.</b> The extent functions whose output is blocked are <code>cell_extent()</code>, <code>cell_extent_layer()</code>, <code>layer_extent()</code>, and <code>chip_extent()</code>.</li> <li>• <b>connect.</b> The connect functions whose output is blocked are <code>connect()</code>, <code>incremental_connect()</code>, <code>stamp()</code>, and <code>stamp_edge()</code>.</li> <li>• <b>both.</b> The output of both extent and connect functions is blocked.</li> <li>• <b>none.</b> Forward data flow analysis is not blocked. This value is the default.</li> </ul> <p>The <code>-bct</code> command-line option must be used with the <code>-il</code> or <code>-iln</code> command-line options. You can use all three options together.</p> <p>The selection of the specified commands, however, is not blocked. For example, if there is a data flow path from an incremental assign layer to a check that requires output from a <code>connect()</code> function, the <code>connect()</code> functions are not blocked even if the <code>-il layer_name -bct connect</code> command-line options are used.</p>

Table 1-2 IC Validator Command-Line Options (Continued)

Option	Definition
<code>-C</code>	<p>Runs only a netlist comparison between the existing schematic netlist and a previously generated layout netlist, using the information specified in the runset. DRC and extraction are not done.</p> <p>This command-line option is intended for rerunning compare when the design data has not changed but some compare settings are changed.</p> <p>Do not place compare-related functions, such as extract functions, filter, and merging, in a conditional statement if its execution depends on some condition that is derived by layers. In this situation, the data dependency triggers the restart of data creation, and the IC Validator tool reruns almost everything in the runset.</p>
<code>-c cell</code>	<p>Overrides the <code>cell</code> argument in the <code>library()</code> function and the <code>layout_top_cell</code> argument in the <code>compare()</code> function.</p> <p>By specifying <code>-cell ""</code> you can run the IC Validator tool using the OASIS or GDSII format without specifying the top cell. The IC Validator tool finds the top cell in the database. If there is more than one potential top cell, the IC Validator tool chooses the first one it finds in the database and prints a warning message in the summary file. Do not use <code>-cell ""</code> in these situations:</p> <ul style="list-style-type: none"> <li>• LVS runsets; that is, runsets with a call to the <code>compare()</code> function.</li> <li>• Runsets that call the <code>get_top_cell()</code> function before the end of the ASSIGN section.</li> <li>• Along with the <code>-ndg</code> or <code>-C</code> command-line options.</li> <li>• Runsets that set the <code>select_window</code> or <code>exclude_window</code> argument of the <code>incremental_options()</code> function.</li> <li>• Databases that are not in GDSII or OASIS format.</li> </ul>
<code>-cache-only</code>	<p>Generates a cache file in the local cache directory for a runset without running it through the engine.</p>



Table 1-2 IC Validator Command-Line Options (Continued)

Option	Definition
<code>-checkpoint</code>	<p>Enables the creation of a checkpoint runset. See the <code>checkpoint()</code> and <code>restart()</code> functions for more information.</p> <p>When creating a checkpoint,</p> <ul style="list-style-type: none"> <li>• Distributed processing is not supported.</li> <li>• LVS runsets are not supported.</li> <li>• Many of the standard optimizations are disabled, resulting in longer runtime. To get the best performance, first manually prune your runset using the run-only option, <code>-ro</code>.</li> </ul>
<code>-clf file</code>	<p>Allows you to add command-line options to your IC Validator run using a text file. It performs a left-to-right replacement with the options from the specified file. This command-line option takes only one file, but you can use multiple <code>-clf</code> options on a command line.</p> <p>The content of the text file is not preprocessed by the UNIX shell, such as for wildcard expansion.</p> <p>For example,</p> <pre>% icv -clf normal.clf -il "one" \ -clf select_violation_rules.clf drc22nm.rs</pre> <p>The text file <code>normal.clf</code> contains</p> <pre># Run Options -dp3 -I PXL</pre> <p>The text file <code>select_violation_rules.clf</code> contains</p> <pre># Select Options -svc "*vClxor*" -svc vClnot</pre> <p>Using these two files is equivalent to the command line</p> <pre>% icv -dp3 -I PXL -il "one" \ -svc "*vClxor*" -svc vClnot drc22nm.rs</pre> <p>The IC Validator tool expands text in the file that has a \$ sign at the start of a variable as an environment variable. For example, the <code>MY_INCLUDE</code> environment variable set on the command line as</p> <pre>% setenv MY_INCLUDE /u/path</pre> <p>could be used in the text file to set the path for runset include files:</p> <pre>-I \$MY_INCLUDE/include</pre>

Table 1-2 IC Validator Command-Line Options (Continued)

Option	Definition
<code>-cpp outputfile</code>	Reads the input runset, writes a preprocessed version of the runset to the output file, and then immediately terminates. All preprocessor directives and macros are fully processed in the output runset, and all included user (non-IC Validator) files are expanded. All comments are removed from the output file and encrypted sections are preserved.
<code>-cpydb db_name db_path</code>	<p>Appends an error classification database to the error classification databases specified in the runset.</p> <ul style="list-style-type: none"> <li><code>db_name</code> specifies the error classification database name.</li> <li><code>db_path</code> specifies the error classification database path.</li> </ul>
<code>-create_lvs_short_output</code> <code>[integer ALL]</code>	<p>Creates VUE output and generates LVS Short Finder output. You can specify to report only a set number of shorts or all shorts. The default is 200.</p> <p>A <code>cellname.vue</code> file is created. The output is in the <code>./run_details/short_out_db</code> directory.</p> <p>When there is a set limit, the shorts are reported in the following order, with the lowest level within the hierarchy reported first:</p> <ol style="list-style-type: none"> <li>Shorts between power and ground</li> <li>Shorts between power and power, or between ground and ground</li> <li>Shorts between power and signal, or between ground and signal</li> <li>Shorts between signal and signal</li> </ol> <p>This output can also be enabled by using the <code>create_lvs_short_output</code> argument of the <code>error_options()</code> function. However, you can only change the number of shorts using the command-line option.</p>
<code>-D name[=value]</code>	Defines symbolic names with an optional value. The default is 1.
<code>-debug [-debug_source file]</code>	Runs the IC Validator PXL Debugger. See <a href="#">Appendix D, “IC Validator PXL Debugger.”</a>

Table 1-2 IC Validator Command-Line Options (Continued)

Option	Definition
<code>-define_empty_cell_as_device</code> [NONE BOTH]	<p>Specifies whether empty cells (X-card instances) are ignored (the default) or recognized as GENDEV primitive devices provided that all required conditions are met. See the <code>compare()</code> function in the <i>IC Validator Reference Manual</i> for more information.</p> <p>For an empty cell to be treated successfully as a device, the IC Validator tool must be able to equate it automatically between the schematic and layout netlists.</p>
<code>-dhe TRUE   FALSE</code>	<p>Enables or disables dual hierarchy device extraction. Use this command-line option to overwrite <code>init_device_matrix(dual_hierarchy_extraction =)</code> in the runset.</p>
<code>-dp</code>	<p>Networks a distributed run on a remote host that is defined either through the use of a distributed processing (DP) options configuration file or a combination of the <code>-dpcache</code> and <code>-dphosts</code> command-line options. The default distributed processing options configuration file is <code>dp_options.conf</code> in the current working directory. See <a href="#">“Distributed Processing Options Configuration File” on page 1-40</a> and <a href="#">“Using Multiple Distributed Processing Hosts” on page 1-39</a>.</p> <p>The default distributed processing mode is <code>-dp2</code>, using only as many processors as there are DP clients on the machines for dynamic threading. If you want to use all the processors on the machines for dynamic threading, use the <code>-turbo</code> option.</p>
<code>-dp#_of_hosts</code>	<p>Networks a distributed run on one local machine using the specified number of distributed processing hosts, all running on this one local host. For example,</p> <pre>% icv -dp4 runset.rs</pre> <p>The default is two hosts, <code>-dp2</code>, using only as many processors as there are DP clients on a machine for dynamic threading. If you want to use all the processors on the machine for dynamic threading, use the <code>-turbo</code> option.</p> <p>See <a href="#">“Distributed Processing Options Configuration File” on page 1-40</a> and <a href="#">“Using Multiple Distributed Processing Hosts” on page 1-39</a>.</p>

Table 1-2 IC Validator Command-Line Options (Continued)

Option	Definition
<code>-dpbatchstr "command"</code>	Specifies an LSF or SGE command string. The command string must be delimited by quotation marks. Do not use the <code>-dpbatchstr</code> and <code>-dpbatchfile</code> options together on the command line. See <a href="#">“Native LSF and SGE Support” on page 1-47</a> for more information.
<code>-dpbatchfile /path/command.txt</code>	Specifies the path and file name of the LSF or SGE command file. The text file contains the LSF or SGE command that is read by the IC Validator tool. Do not use the <code>-dpbatchstr</code> and <code>-dpbatchfile</code> options together on the command line. See <a href="#">“Native LSF and SGE Support” on page 1-47</a> for more information.
<code>-dpcache path</code>	Specifies a local group directory for all distributed processing <code>dpcache</code> with <code>-dp</code> or <code>-dp#</code> hosts in a network distributed run. The <code>-dp</code> or <code>-dp#_of_hosts</code> command-line option must be specified before the <code>-dpcache</code> option for a distributed run. For example, <pre>% icv -dp4 -dpcache /SCRATCH/user/_group runset.rs running on local machine % icv -dp -dpcache /SCRATCH/user/_group -dphosts \ host1 host2 runset.rs running on machines host1 and host2</pre>
<code>-dphosts host1</code> <code>[local_group_dir] ... hostn</code> <code>[local_group_dir]</code>	Specifies a set of distributed processing hosts to be used in a network distributed run. To run multiple distributed processing hosts on a multicore or multiprocessor system, a host name can be specified multiple times. The <code>-dp</code> option must be specified before the <code>-dphosts</code> option. For example, <pre>% icv -dp -dphosts host1 host1 host2 runset.rs</pre> For each distributed processing host, an optional local group directory can be specified. This directory must be specified as an absolute path accessible on the distributed processing host. For example, <pre>% icv -dp -dphosts host1 /SCRATCH/user/_group \ host1 /SCRATCH/user/_group \ host2 /SCRATCH/user/_group \ runset.rs</pre>
<code>-dpid host_name</code>	Specifies the host name of the current machine for a network distributed run.

Table 1-2 IC Validator Command-Line Options (Continued)

Option	Definition
<code>-dplearn run_details/ estimator.dat</code>	Runs an adaptive job that uses an estimator.dat file from a previous IC Validator run to improve scaling of a new IC Validator run. The path to <code>run_details/estimator.dat</code> can be either full or relative. See <a href="#">“Distributed Processing Learn” on page 1-41</a> for more information.
<code>-dpm</code>	Runs the <code>dp_monitor</code> utility. This utility displays the status of an IC Validator distributed processing run and supports adding a host to the run.
<code>-dpmem</code>	Optimizes the use of memory over runtime during a distributed processing run.
<code>-dpopts [filename]</code>	<p>Uses <i>filename</i> as the distributed processing options configuration file. Using a file for the distributed processing options allows you to maintain different distributed processing configurations, selecting the desired configuration at the start of the run. The default distributed processing options configuration file is <code>dp_options.conf</code> in the current working directory. See <a href="#">“Distributed Processing Options Configuration File” on page 1-40</a>.</p> <p>The <code>-dp</code> option must be specified before the <code>-dpopts</code> option. For example,</p> <pre>% icv -dp -dpopts my_dp_configuration.txt runset.rs</pre>
<code>-dpstrict</code>	<p>Aborts the IC Validator job if unable to acquire the exact machine configuration that was requested for any of the following situations:</p> <ul style="list-style-type: none"> <li>Any DP host specified cannot be acquired</li> <li>Communication cannot be started with any DP host</li> <li>The licenses needed for the requested DP are not available or any other error is found when trying to check out the licenses</li> </ul>
<code>-dvp LVS   ALL</code>	Extracts <code>ALL</code> (the default) properties or the ones used only by <code>LVS</code> compare. Use this command-line option to overwrite <code>init_device_matrix(device_properties = )</code> in the <code>runset</code> .

Table 1-2 IC Validator Command-Line Options (Continued)

Option	Definition
<code>-e file</code>	<p>Adds the <code>equiv_options()</code> and <code>lvs_black_box_options()</code> function calls in the specified file into the runset. The file can contain one or more calls to these functions.</p> <p>The only IC Validator functions that can be in the file are the <code>equiv_options()</code> and <code>lvs_black_box_options()</code> functions, but other valid PXL commands can be used. The following example shows the use of a <code>foreach</code> loop:</p> <pre>equiv_cells : list of string = {"cellA", "cellB", "cellC", "cellD", "cellE", "cellF"}; foreach( cell in equiv_cells ){     equiv_options({{cell, cell}}); }</pre> <p>If there are calls to the <code>equiv_options()</code> and <code>lvs_black_box_options()</code> functions in the runset, the equivalence points from the runset and the equivalence file are accumulated.</p>
<code>-ece</code>	<p>Enables the reporting of additional error codes to the shell. See <a href="#">“Exit Codes” on page 1-34</a>.</p> <p>See the documentation for the shell you are using regarding how to capture the error code status.</p>
<code>-ep prefix</code>	<p>Overrides the prefix for all error cell names specified in the write functions.</p>
<code>-ex</code>	<p>Runs device extraction only. The <code>compare()</code> function is not invoked, and as a result, all functions that use the <code>compare()</code> function output are not run.</p>
<code>-f format</code>	<p>Overrides the library format (GDSII, Milkyway, NDM, OASIS, or OpenAccess) specified in the <code>library()</code> function.</p>

Table 1-2 IC Validator Command-Line Options (Continued)

Option	Definition
<code>-g directory</code>	<p>Overrides the group path name specified in the <code>group_path</code> argument of the <code>run_options()</code> function. The IC Validator tool does not allow you to set this value to the current working directory (.).</p> <p>This command-line option takes only one directory. If you use multiple <code>-g</code> options, the IC Validator tool uses the directory from the last one.</p> <p>Note:</p> <p>The directory provided to this command-line option must be visible to all hosts. If you provide a local cache directory, see the <code>-dpcache</code> command-line option.</p>
<code>-hlic</code>	<p>Forces use of shared licensing scheme. See <a href="#">“Shared Licensing” on page 2-4</a>.</p>
<code>-i name</code>	<p>Overrides the library names specified in the <code>library_name</code> and <code>merge_libraries</code> arguments of the <code>library()</code> function. When you use the <code>-i</code> command-line option, the <code>library_name</code> and <code>merge_libraries</code> arguments are completely redefined; none of the original values remain.</p> <p>You can specify multiple <code>-i</code> command-line options. The <code>library_name</code> is replaced by the first <code>-i</code> command-line option. The merged libraries are created from subsequent <code>-i</code> values, with the first merged library created from the second <code>-i</code> command-line option. The format for the merge libraries is set to the same format as for the main library. There is no layer mapping.</p>

*Table 1-2 IC Validator Command-Line Options (Continued)*

Option	Definition
<code>-I path</code>	<p>Specifies the preprocessor include directories. This command-line option takes only one path, but you can use multiple <code>-I</code> options on a command line.</p> <p>When you specify multiple paths, order matters. For example, you have the file <code>aaa.rh</code> in both <code>my_path1</code> and <code>my_path2</code>.</p> <ul style="list-style-type: none"><li>• When you specify <code>-I my_path1 -I my_path2</code>, the <code>aaa.rh</code> file from <code>my_path1</code> is used.</li><li>• When you specify <code>-I my_path2 -I my_path1</code>, the <code>aaa.rh</code> file from <code>my_path2</code> is used.</li></ul> <p>Include directories are searched after the current working directory is searched.</p> <p>If an include directory is not in the install directory, then for the <code>-I</code> command-line option, you must specify the absolute path of the directory.</p>



Table 1-2 IC Validator Command-Line Options (Continued)

Option	Definition
<code>-icv_addhost</code>	<p>Starts the IC Validator tool with add host feature. Does not wait for all of the hosts to queue up before running.</p> <p>The <code>icv_addhost</code> command-line option adds worker hosts to a currently executed distributed IC Validator run. This command-line option is used by scripts that interact with batch queuing systems.</p> <p>By default, this command-line option adds the host it is run on to the job that created the <code>run_details</code> subdirectory in the working directory where <code>icv_addhost</code> is run. You can change the default by using the following options:</p> <ul style="list-style-type: none"> <li>• <code>-rd /path/to/run_details</code>. Location of <code>run_details</code> directory associated with the run to which the worker is added. The default is <code>run_details</code> under the directory where <code>icv_addhost</code> is run.</li> <li>• <code>host</code>. The hostname of the host to add. The default is the host where <code>icv_addhost</code> is run.</li> <li>• <code>/cache/dir/path</code>. An absolute path to local disk space on host to use as a data cache. The default is to not use local disk cache.</li> </ul> <p>Limitations: The <code>icv_addhost</code> command-line option must be run after the <code>run_details</code> directory is created, which is after the runset is parsed and the IC Validator tool executes commands.</p> <p>Worker processes cannot be added to a distributed processing job started with <code>-dpN</code> (eg, <code>-dp8</code>). The job must be started using either <code>-dphosts</code>, <code>dp.config</code>, or <code>.dprc</code> to specify a machine configuration.</p> <p>If there is not already a worker process on the host, it must be possible to log into that host without a password, using the program specified by the <code>ICV_RSH_COMMAND</code> environment variable.</p> <p>Return value: If <code>icv_addhost</code> command-line option is successful, it prints the message "OK" and exits with a status of 0. If it is not successful, it prints an error message and exits with a status of 1.</p>
<code>-icvread</code>	<p>Enables the creation of an ICVread matrix. See the <a href="#">init_icvread_matrix()</a> function description. If you use the <a href="#">icvread()</a> function in your runset, you do not need to use this command-line option.</p>

Table 1-2 IC Validator Command-Line Options (Continued)

Option	Definition
<code>-il "layer ... layer"</code>	<p>Defines the incremental layers to be used in an incremental-by-layer run by layer name.</p> <ul style="list-style-type: none"> <li>Put the set of layer names within quotation marks.</li> <li>Specify multiple names with spaces as separators.</li> </ul> <p>For example,</p> <pre>-il "POLY" -il "POLY DIFF METAL"</pre> <p><b>Note:</b></p> <p>Layer names specified in the <code>name</code> argument of any assign function are used for matching incremental assign layer names.</p> <p>When the <code>name</code> argument of an assign function is not used, the layer name for the assign is from the left side assignment, for example, <code>layer_name = assign(...)</code>.</p> <p>If more than one layer name is specified, incremental processing occurs if one or more layer names are found in the ASSIGN section. Layer names that are not found are ignored. If none of the specified layers are found in the ASSIGN section, the IC Validator tool exits with an error message.</p> <p>The violations that are based, directly or indirectly, on the incremental layers are executed in the IC Validator run. The other violations are not executed in the IC Validator run.</p>
<code>-ilic</code>	<p>Forces use of native licensing scheme. See <a href="#">“Native Licensing” on page 2-3</a>.</p>

Table 1-2 IC Validator Command-Line Options (Continued)

Option	Definition
<code>-iln</code> <code>"layer_number1[-layer_number2]</code> <code>[,datatype1[-datatype2]] ... "</code>	<p>Defines the incremental layers to be used in an incremental-by-layer run by number in the runset.</p> <ul style="list-style-type: none"> <li>Put the set of layer numbers within quotation marks.</li> <li>Specify multiple layers with spaces as separators.</li> <li>Use dashes to specify a range of layers or data types.</li> <li>Use commas to separate the layer numbers from the datatypes.</li> </ul> <p>For example, to specify (layers 4 to 5, datatypes 0 to 1), (layer 3), and (layers 1 to 2, datatype 1) use the following command-line option:</p> <pre>-iln "4-5,0-1 3 1-2,1"</pre> <p>If more than one layer number is specified, incremental processing occurs if one or more layer numbers are found in the assign function. Numbers that are not found are ignored. If none of the specified layer numbers are found in the assign function, the IC Validator tool exits with an error message.</p> <p>Violations that are based, either directly or indirectly, on incremental layers are executed in the IC Validator run. The other violations are not executed in the IC Validator run.</p>
<code>-layer_debugger</code>	<p>Enables the storing of layer debugger output without the use of the VUE. For compatibility with VUE, this command-line option enables the creation of VUE output (as if by the <code>-vue</code> command-line option). Also, to match the VUE layer debugger behavior, runset selection is performed based on the union of commands selected by each of the following categories: violation comment (<code>-svc</code> and <code>-uvc</code>), violation name (<code>-svn</code> and <code>-uvn</code>), function name (<code>-sfn</code> and <code>-ufn</code>), and run-only selection.</p>
<code>-lf layermappingfile</code>	<p>Specifies the layer mapping file. This file maps the runset layer numbers to the corresponding layer numbers in the library. See <a href="#">“Layer Mapping” on page 1-49</a>.</p>
<code>-ln filename</code>	<p>Overrides all the filenames specified in the <code>layout_file</code> argument of the <code>read_layout_net()</code> function.</p> <p>When you use the <code>-ln</code> command-line option, you must also use the <code>-lnf</code> command-line option.</p>
<code>-lnf SPICE   VERILOG   ICV</code>	<p>Overrides all the formats specified in the <code>layout_file</code> argument of the <code>read_layout_net()</code> function.</p>

Table 1-2 IC Validator Command-Line Options (Continued)

Option	Definition
<code>-lvs_license</code>	Enables conditional licensing for LVS to omit a DRC license.
<code>-m factor</code>	Overrides the database magnification factor in the <code>library()</code> function.
<code>-milkyway_version</code>	Prints Milkyway schema versions supported by the current IC Validator version.
<code>-ml filename</code>	Puts into the runset a call to the <code>format_merge_library_options()</code> function that is defined in <code>filename</code> . The <code>format</code> can be <code>milkyway</code> , <code>ndm</code> , or <code>openaccess</code> . The syntax for this file is exactly that of the <code>format_merge_library_options()</code> function. Any <code>format_merge_library_options()</code> function in the runset is overwritten.
<code>-ndg</code>	<p>Enables use of streamlined processing flow, which bypasses some stages of runset compilation, generation of dependencies between functions, and runset optimizations that require dependency analysis. Without dependency generation, the IC Validator tool cannot run in distributed processing mode, and all functions in the runset are executed sequentially on a single host.</p> <p>Use the <code>-ndg</code> option with small cells, where the incremental cost of running all functions is significantly lower than the time and memory needed to fully optimize a complex runset.</p> <p>When using the <code>-ndg</code> option, any user function that invokes an inline runset function must be tagged with the <code>inline</code> qualifier. In general, IC Validator <code>OPTIONS</code> and <code>ASSIGN</code> section functions fall into this category. See <a href="#">“Inlined Functions” on page 1-35</a> for a complete list of the runset functions that are tagged with the <code>inline</code> qualifier. See the <code>inline</code> qualifier description in <a href="#">“Qualifiers” on page 4-68</a>.</p> <p>When using the <code>-ndg</code> option, you cannot use the dual-hierarchy extraction flow.</p>

*Table 1-2 IC Validator Command-Line Options (Continued)*

Option	Definition
<code>-ndm_default_layers</code>	<p>The <code>-ndm_default_layers</code> command-line option functionality is equivalent to the <code>-layer_map_format</code> option functionality of the <code>write_gds</code> command in the IC Compiler II tool.</p> <p>This command-line option specifies that the listed layers for which the geometry and text information corresponding to any color-mask configuration are read by the IC Validator tool as default layer data as well.</p> <p>When this command-line option is specified along with layer mapping information for color and mask data, all of the geometry and text layer information is read twice; first as the layer-datatype pair obtained from the layer map considered as color-mask data, and second as the layer-datatype pair obtained from the layer map considered as non-color or mask data.</p>
<code>-ndm_layer_map_format icc2   icc_default   icc_extended</code>	<p>Specifies the way the input layer mapping file is interpreted by the IC Validator tool for NDM input layouts, particularly for Milkyway-formatted layer mapping files that have different layer numbers for system layers in normal versus extended layer mode. This command-line option is equivalent to the <code>-layer_map_format</code> option of the <code>write_gds</code> command in the IC Compiler II tool.</p> <p><b>Note:</b></p> <p>The default if the <code>-ndm_layer_map_format</code> command-line option is not called is to auto-detect either the <code>icc2</code> or <code>icc_default</code> format. This command-line option does not auto-detect layer mapping files in the <code>icc_extended</code> format.</p>
<code>-ndm_design_label</code>	Overrides the <code>design_label</code> argument of the <code>ndm_options()</code> function.
<code>-ndm_version</code>	Prints NDM schema versions supported by the current IC Validator version.
<code>-ned</code>	Overrides the <code>report_error_details</code> argument of the <code>error_options()</code> function. Use this command-line option to not report violation details to the <code>LAYOUT_ERRORS</code> and <code>TOP_LAYOUT_ERRORS</code> files. The information is still stored in the error database (PYDB), but only the violation summary section is written to the <code>LAYOUT_ERRORS</code> file.

Table 1-2 IC Validator Command-Line Options (Continued)

Option	Definition
-nho	Turns off hierarchy optimization. See the <a href="#">hierarchy_options()</a> function.
-norscache	Disables runset caching. See <a href="#">"Runset Caching" on page 1-42</a> for more information.
-nro	Disables runset optimizations.
-nvn	Enables the IC Validator tool to run a netlist-versus-netlist (NVN) flow without a runset. When this option is used, any runset on the IC Validator command line is not read.
-nvn_netlist_mode	Specifies the netlist-versus-netlist (NVN) flow mode as layout-versus-layout (LVL) or schematic-versus-schematic (SVS). When set to LVL, the valid pin and device names used in both input netlists are always referenced to layout names in the runset. When set to SVS, the names are always schematic names. When this command-line option is not specified, the pin and device names are schematic and layout, respectively.
-nwl	NetTran outputs the wireLog file from the schematic, and the engine reads this file to resolve the text short.
-nwl <i>wire_log_file</i>	<p>Provides a wireLog file to guide IC Validator on how to resolve shorts. Overrides the wireLog file generated automatically by NetTran during IC Validator run.</p> <p>You can generate a wireLog file from your schematic netlist. See the "Creating a WireLog File Using NetTran" in the <i>IC Validator LVS User Guide</i> for more information about how you can manually generate a wireLog file running NetTran with its command-line option.</p> <p>The wireLog file allows the IC Validator tool to use the same net as NetTran when there is a short. The IC Validator tool assigns the <i>newNetName</i> to the net.</p> <p>Example of wireLog file syntax:</p> <pre># format: cellName origNetName newNetName add4      VDD2      VDD cs_add    VDD5      VDD cs_add    n36       VDD nor2b     VDD6      VDD5</pre>

Table 1-2 IC Validator Command-Line Options (Continued)

Option	Definition
<code>-oa_cell_map cellmap</code>	Overrides the <code>cell_mapping_file</code> argument specified in the <code>openaccess_options()</code> function.
<code>-oa_color_mapping option</code>	Specifies how to use the locked and unlocked anchor bit for the OpenAccess color layer mapping. The selections for <i>option</i> are <ul style="list-style-type: none"> <li><code>ignore</code>. Uses the color mappings for which the locked or unlocked keyword is not specified, and ignores the color mappings for which locked or unlocked is specified.</li> <li><code>strict</code>. Uses the color mappings for which the locked or unlocked keyword is specified, and ignores the color mappings for which locked or unlocked is not specified.</li> <li><code>full</code>. Sets all shapes that are read with color to locked. The mapping for this setting is the same as the mapping for the <code>compatibility</code> setting.</li> <li><code>compatibility</code>. Follows the industry standard. Ignores color mappings for which the <code>colorAnnotate</code> keyword is located after the color specification. When no color mapping rule matches a color shape, the mapping is used without color.</li> </ul>
<code>-oa_dm4</code>	Uses OpenAccess shared libraries that are compatible with DM4 plug-ins.
<code>-oa_dm5</code>	Uses OpenAccess shared libraries that are compatible with DM5 plug-ins. The default is <code>-oa_dm5</code> .
<code>-oa_layer_map layer_mapping_file</code>	Overrides the <code>layer_mapping_file</code> argument setting specified in the <code>openaccess_options()</code> functions.
<code>-oa_lib_defs filename</code>	Overrides the OpenAccess <code>library_definition_file</code> argument setting specified in the <code>library()</code> function.
<code>-oa_view view_name</code>	Overrides the <code>view</code> argument specified in the <code>openaccess_options()</code> function.
<code>-p path</code>	Overrides the library path specified in the <code>library()</code> function.
<code>-pc outputfile</code>	Reads the runset and writes parser-checked PXL output. Note: The <code>-pc</code> command-line option is deprecated and will be removed in a future release. Use the <code>-pcr</code> or <code>-cache-only</code> command-line option instead.

Table 1-2 IC Validator Command-Line Options (Continued)

Option	Definition
<code>-pcr outputfile</code>	Reads the runset and writes parser-checked PXL output after function inlining and loop unrolling. The output can be pruned using violation selections or run-only execution. See <a href="#">“Run-Only Execution” on page 1-31</a> for more information.
<code>-pec KEEP   EXPLODE</code>	Overrides the setting of the <code>partially_exploded_cells</code> argument in the <code>error_options()</code> function.
<code>-prune_assign</code>	<p>The IC Validator tool reads only the required runset layers from input design, and creates the required runset layers.</p> <p>When this command-line option is not used, the IC Validator tool reads all the runset layers from input design. Whether or not you use the <code>-prune_assign</code> command-line option, unnecessary runset layers are not reported in the summary file.</p> <p>See <a href="#">Pruning Layers</a> for information about the <code>prune_assign</code> compiler directive.</p>
<code>-rd directory</code>	Overrides the run details directory specified in the <code>run_details_path</code> argument of the <code>run_options()</code> function.
<code>-ro</code>	Enables run-only processing. See <a href="#">“Run-Only Execution” on page 1-31</a> .
<code>-rofl file:line</code>	Enables run-only command selection based on the file and line number.
<code>-rsi</code>	<p>Reports input library information. Supports only GDSII and OASIS input formats. Reports</p> <ul style="list-style-type: none"> <li>• GDSII last modified and last accessed</li> <li>• md5sum computed with <code>md5sum -b file</code></li> <li>• format of the input library</li> <li>• path to the input library</li> <li>• command used for the library input</li> </ul> <p>Enumerators must be in a comma-separated list, and the entire list must be within quotation marks. See the <code>report_streamfile_information</code> argument of the <code>run_options()</code> function.</p> <p>Note:</p> <p>The <code>-rsi</code> command-line option overrides its runset setting in the <code>run_options()</code> function.</p>



Table 1-2 IC Validator Command-Line Options (Continued)

Option	Definition
<code>-s <i>schematic</i></code>	<p>Overrides the file name specified in the <code>schematic_file</code> argument of the <code>schematic()</code> function. The <code>schematic_library_file</code> argument of the <code>schematic()</code> function is not overwritten.</p> <p>When you use the <code>-s</code> command-line option, you must also use the <code>-sf</code> command-line option.</p> <p>If the schematic file is not found when LVS compare is run, the IC Validator tool exits with a message stating that it cannot open the file to read.</p>
<code>-sf SPICE   VERILOG   ICV</code>	<p>Specifies the input schematic netlist format, overriding the format specified in the <code>schematic_file</code> argument of the <code>schematic()</code> function. The <code>schematic_library_file</code> argument of the <code>schematic()</code> function is not overwritten.</p>
<code>-small_cell</code>	<p>Overrides automatic small cell determination and forces small cell mode to run as fast as possible on small designs. To prevent slowdown on larger designs, the IC Validator tool exits with an error if it determines that the input design is too large.</p>
<code>-snapshot</code>	<p>The IC Validator tool executes the <code>snapshot()</code> function. Use the <code>-snapshot_resume</code> option to restart the run.</p> <p>When creating a snapshot,</p> <ul style="list-style-type: none"> <li>• Distributed processing is supported.</li> <li>• LVS runsets are supported.</li> </ul>
<code>-snapshot_resume <i>path_to_data</i></code>	<p>The IC Validator tool resumes a run using the specified snapshot data. See the <code>snapshot()</code> function for more information.</p> <p>Any command-line option that changes what runset functions are run and the sequence in which the functions are run must be identical between the snapshot and resumed runs. Being identical includes turning on or off runset optimizations, removal of duplicates, toggling run-only, and selectable rules options such as <code>-iln</code>.</p>
<code>-stc <i>cell</i></code>	<p>Overrides the top cell in the schematic netlist, overriding the <code>schematic_top_cell</code> argument of the <code>compare()</code> function and the <code>cell</code> argument of the <code>schematic()</code> function.</p>

Table 1-2 IC Validator Command-Line Options (Continued)

Option	Definition
<code>-svc violation_comment</code>	<p>Selects the violations that are executed during the run. There are three categories of selection rules: select or suppress by violation comment (<code>-svc</code> and <code>-uvc</code>), by violation name (<code>-svn</code> and <code>-uvn</code>), and by function name (<code>-sfm</code> and <code>-ufm</code>). The arguments</p> <ul style="list-style-type: none"> <li>• Can include the wildcard <code>*</code> and range expressions using <code>[ ]</code>.</li> <li>• Must be enclosed in quotation marks if it uses a wildcard <code>*</code>. (This enclosure prevents processing by the UNIX shell.)</li> </ul> <p>These options take only one argument but you can specify multiple options on a command line. For example,</p> <pre>% icv -svc "*96A1*" -svc "*11B16*"</pre> <ul style="list-style-type: none"> <li>• For each category, selection is based on what is selected and not suppressed. Final selection is the intersection of the violations selected in each category.</li> <li>• If the selection option is not used for a category but the suppression option is used, then all violations excluding those suppressed are selected for that category.</li> <li>• If both selection and suppression options are missing for a category, then for that category all violations are selected. If you do not use any of the options, the whole runset is selected.</li> <li>• If the final selection of violations is empty, the run terminates with an error.</li> </ul>
<code>-uvc violation_comment</code>	
<code>-svn violation_name</code>	
<code>-uvn violation_name</code>	
<code>-sfm function</code>	
<code>-ufm function</code>	
<code>-turbo</code>	Specifies to use all the processors on the machines for dynamic threading. When this option is not used, only as many processors as there are DP clients on a machine are used for dynamic threading.
<code>-U name</code>	Undefines symbolic names that were previously defined with the <code>-D</code> command-line option.
<code>-usage</code>	Displays usage message and exits.
<code>-V</code>	Prints the program version.

Table 1-2 IC Validator Command-Line Options (Continued)

Option	Definition
<code>-verbose</code>	Prints the output of the individual commands to the screen while the IC Validator tool is running rather than only percentage complete messages. The information printed is the same as the information reported to the summary file. However, the information is not displayed in the same order. The screen output reports the information in the order of processing by the IC Validator tool, while the content of the summary file is sorted to match the sequential order of the runset.
<code>-Version</code>	Prints version numbers for the individual components and the library versions of Synopsys software.
<code>-vfn <i>function_name</i></code> <code>-vfn ALL</code>	<p>Reports evaluated parameter values of the specified runset function in the summary file, and on the screen when the <code>-verbose</code> option is used. This command-line option takes only one function name but you can use multiple <code>-vfn</code> options on a command line. You can also use the <code>-vfn ALL</code> option to report the values for all runset functions.</p> <p>The message contains the function name, actual value of each parameter of the function, and the return type of the function. This information is useful for debugging the input parameter value of any function call. Here is an example of the output of the <code>-vfn library</code> option:</p> <pre>&lt;runset&gt;:&lt;line_no&gt;: library (     library_name = CELL_LEVEL     format = MILKYWAY     cell = TOP     library_path = &lt;path&gt;     library_definition_file =     magnification_factor = 1. ) returning void</pre>
<code>-vue</code>	Creates VUE output. Creating VUE output can also be enabled by setting the <code>create_vue_output</code> argument in the <code>error_options()</code> function to <code>true</code> .

Table 1-2 IC Validator Command-Line Options (Continued)

Option	Definition
<code>-vueshort [integer ALL]</code>	<p>Extracts either a specified number of shorts or all shorts. The default is 200.</p> <p>The extracted data is in the <code>./run_details/short_out_db</code> directory.</p> <p>When there is a set limit, the shorts are reported in the following order, with the lowest level within the hierarchy reported first:</p> <ol style="list-style-type: none"> <li>1. Shorts between power and ground</li> <li>2. Shorts between power and power, or between ground and ground</li> <li>3. Shorts between power and signal, or between ground and signal</li> <li>4. Shorts between signal and signal</li> </ol> <p>You can also set the shorts to be extracted using the following arguments of the <code>text_net()</code> function:</p> <pre>create_short_finder_nets and short_debugging = {vue_short_finder=true, output_limit=integer}.</pre>
<code>-w</code>	<p>Allows the network path to the working directory to be explicitly specified. See <a href="#">“Resolving Automounter Issues” on page 1-40</a>.</p>
<code>-zn</code>	<p>Compresses all netlists generated by the <code>netlist()</code>, <code>write_spice()</code>, <code>write_customized_spice()</code> and <code>write_xref_spice()</code> functions. Compresses equivalence netlists generated by the <code>compare()</code> function.</p>

## Compiler Directives

### Support for 64-Bit Coordinates

The IC Validator tool converts coordinate data to a signed 32-bit integer coordinate space. Large die sizes with a small database resolution might require a larger integer space, causing a coordinate overflow error to occur.

To increase the coordinate space, use the `-64` command-line option. Only use the `-64` option when it is required; this is because the memory usage increases.

The value limits for a 32-bit integer are:

- Maximum positive value: 2,147,483,647
- Maximum negative value: -2,147,483,648

In the following example,

- $R_{\text{coord}}$  is a real number xy coordinate value.
- $I_{\text{coord}}$  is the coordinate converted to an integer.

Assume the following values:

- $R_{\text{coord}} = 25,000.1234$  mm (2.50001234 cm from 0.0000)
- Working resolution = 2x

For a database resolution of 0.0001 mm, the  $R_{\text{coord}}$  value does not overflow the 32-bit integer coordinate space:

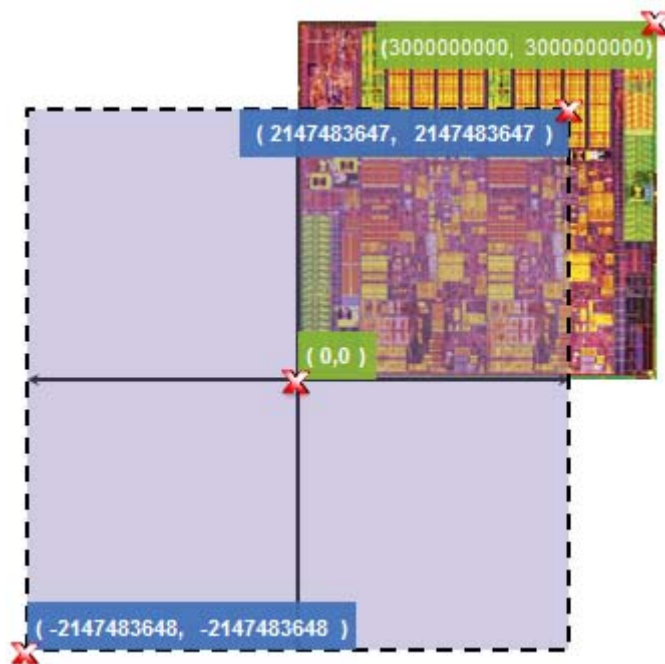
$$I_{\text{coord}} = 25,000.1234 * (1/0.0001) * 2 = 500,002,468 < 2,147,483,647$$

For a database resolution of 0.00001, the  $R_{\text{coord}}$  value overflows the 32-bit integer coordinate space:

$$I_{\text{coord}} = 25,000.1234 * (1/0.00001) * 2 = 5,000,024,680 > 2,147,483,647$$

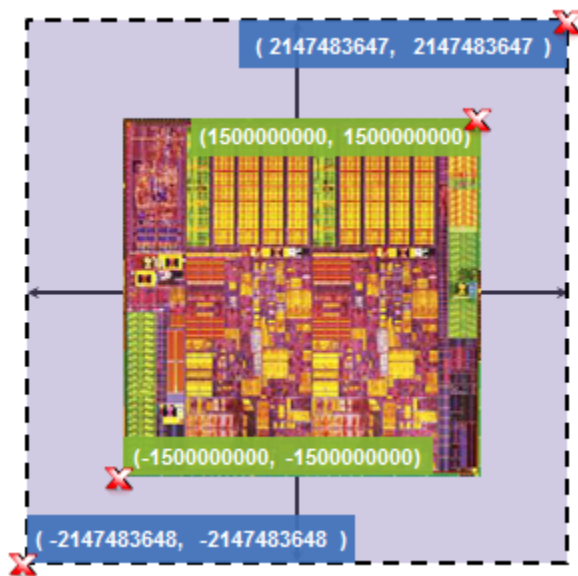
Not all overflows, however, are created equally. Before using the  $-64$  option, make sure that you are using all of the available coordinate space. As shown in [Figure 1-1](#), using only positive coordinate values artificially limits the maximum die size.

*Figure 1-1 Overflow of the 32-Bit Integer Coordinate Space*



Center the die about the origin to maximize available coordinate space, as shown in [Figure 1-2](#). Using the `-64` option is not needed to prevent an overflow.

*Figure 1-2 No Overflow of the 32-Bit Integer Coordinate Space*



The IC Validator error database is enhanced with a variable-width coordinate format that can accommodate either 32-bit or 64-bit integer coordinates. This applies to both the error

databases (PYDB) and the error classification databases (cPYDB), which may be exported for DRC error classification.

The pydb\_export utility is enhanced with the ability to convert error classification databases (cPYDB) among the various coordinate formats to maintain compatibility as necessary.

**Note:**

Current and future IC Validator tools retain support for reading error databases with the older fixed 32-bit or 64-bit coordinate formats. However, IC Validator tools prior to version L-2016.06 will not be able to read the new coordinate format.

---

## Run-Only Execution

Follow these steps for run-only execution.

1. Identify the sections of the runset for which you want the desired runset checks with these directives:

```
#pragma runonly begin
#pragma runonly end
```

You can define multiple run-only sections within a runset.

2. Call the IC Validator tool with `-ro`, the run-only command-line option, to
  - a. Parse the runset.
  - b. Prune the runset functions that do not have data flow into run-only code sections.
  - c. Execute the pruned runset.

**Note:**

If the IC Validator tool does not find a run-only section within a runset, the runset is not executed.

3. Alternately, to stop execution after the runset is parsed and pruned for run-only execution, use the `-pcr` command-line option along with `-ro`. For example,

```
% icv -ro -pcr outfile
```

4. Use the `-nro` command-line option when using pruned run-only output as the source runset. If you do not use the `-nro` option, default pruning optimizations can eliminate functions inside run-only code sections.

**Note:**

All `#pragma` run-only directives remain in the pruned run-only output.

The presence of `#pragma` run-only directives in the pruned run-only output enables the `icv -ro runonly.rs` command line to be used for execution of the output pruned `runonly.rs` runset.

---

## Environment Variables

The \$ sign preceding a variable indicates the variable is an environment variable. For example,

```
$ENV_ABC
```

The variable must be set in the environment; otherwise a parse error occurs. All environment variables are considered strings. For example,

```
if( $ENV_ABC == "XYZ" ) {
    x = abc(XYZ);
}
else {
    x = abc(NOT_XYZ)
}
```

---

## Providing Additional Information to the Compiler

The compiler directive `#pragma envvar` interfaces with environment variables to

- Print a list containing the names and the values of each specified environment variable.

```
#pragma envvar report variable ... variable
```

If any variable does not exist, an error occurs. For example,

```
error: #pragma envvar references undefined environment variable 'e'
```

- Probe the environment for the named variable. For example,

```
// input file
#pragma envvar default x "aa"
#pragma envvar default y "bb"
#pragma envvar default z $x "__" $y // Use $X notation to
                                   // reference defined
                                   // variables

#pragma envvar report x y z

// output from compiler
a.rs:4: x=aa
a.rs:4: y=bb
a.rs:4: z=aa__bb
```

- If the named variable exists, no action is taken.
- If the named variable does not exist, the value of the variable in the environment is set to the concatenation of the rest of the line. The concatenation must be a sequence of string literals or references to other environment variables, or both.

```
#pragma envvar default variable ... variable
```



**Note:**

If there is more than one `#pragma envvar default` statement that defines the same variable, subsequent occurrences see the variable as already set in the environment, as set by the first occurrence.

---

## Creating Preprocessor Definitions from Environment Variables

The preprocessor `#pragma define_from_envvar` creates a preprocessor definition from an environment variable, if the environment variable is set. The syntax is

```
#pragma PXL define_from_envvar cpp_var env_var
```

The preprocessor definition named by *cpp\_var* is given the value of the environment variable named by *env\_var*, as if by a `#define` command. If the environment variable is not set, no action is taken, such that *cpp\_var* remains undefined or retains its previous definition.

The preprocessor `#pragma define_from_envvar` takes two arguments:

- The first argument is the name of the preprocessor definition to redefine if the environment variable is set.
- The second argument is the name of the environment variable to get the value from.

```
#include <icv.rh>

#define str(x) #x
#define qstr(x) str(x)

#if defined(FOO_DEF)
note("Before pragma, defined as: " + qstr(FOO_DEF));
#else
note("Not defined before pragma");
#endif

#pragma PXL define_from_envvar FOO_DEF FOO_ENV

#if defined(FOO_DEF)
note("After pragma, defined as: " + qstr(FOO_DEF));
#if FOO_DEF == 2
note("... and satisfies test for == 2");
#elif FOO_DEF == 3
note("... and satisfies test for == 3");
#endif
#else
note("Not defined after pragma");
#endif
```

When the run is as follows:

```
% export FOO_ENV=3
% icv -verbose -D FOO_DEF=2 define_from_envvar.rs
```

The output is as follows:

```
define_from_envvar.rs:10: Before pragma, defined as: 2
define_from_envvar.rs:17: After pragma, defined as: 3
define_from_envvar.rs:21: ... and satisfies test for == 3
```

Notice that the original value is 2 because the preprocessor definition passed on the command line. The `define_from_envvar` pragma redefines `FOO_DEF` to the value 3. If `FOO_ENV` was unset, `FOO_DEF` would have been left with the value of 2. If `FOO_DEF` had also not been defined on the command line, it would have remained undefined for the entirety of preprocessing.

---

## Pruning Layers

The `#pragma prune_assign` compiler directive tells the IC Validator tool to read only the required runset layers from input design and create the required runset layers. You can use this compiler directive to enable or to disable pruning:

```
#pragma icv compiler [enable|disable] prune_assign
```

When this compiler directive is not used or is `disable` and the `-prune_assign` command-line option is not used, the IC Validator tool reads all the runset layers from input design. In all cases, unnecessary runset layers are not reported in the summary file.

The setting of the `prune_assign` directive in the runset has priority over the `-prune_assign` command-line option.

---

## Exit Codes

[Table 1-3](#) lists the general exit codes. If an error occurs, the IC Validator tool reports an exit code only when the `-ece` option is specified on the command line. See [Table 1-2](#).

*Table 1-3 General Exit Codes*

Number	Type	Explanation
0	EXIT_COMPLETE	Run completed.
30	EXIT_COMPLETE_WERROR	Run completed with errors reported in the <code>cell.LAYOUT_ERRORS</code> file.

*Table 1-3 General Exit Codes (Continued)*

Number	Type	Explanation
32	EXIT_USER_INT	Run was interrupted by the user with Ctrl+C.
33	EXIT_USER_ERR	Error caused by the user, such as a parser error or a file, was not found.
34	EXIT_OS	System error, such as division by zero, bus error, disk full, out of swap space, or could not create directory or file.
35	EXIT_PANIC	An internal fatal error occurred.

[Table 1-4](#) lists the license manager related exit codes.

*Table 1-4 License Manager Related Exit Codes*

Number	Type	Explanation
64	EXIT_LIC_TOO_MANY	Too many licenses checked out.
65	EXIT_LIC_NOT_ALLOWED	Station not authorized to run this product.
66	EXIT_LIC_NO_REPLY	Could not contact license server.
67	EXIT_LIC_DENIED	License denied, no explanation given.
(-1)	EXIT_NONE	An uninitialized exit status variable.

## Inlined Functions

When you run the IC Validator tool with the `-ndg` command-line option,

- Any user function that invokes an inline runset function needs to be tagged with the `inline` qualifier. See [Table 1-5](#) for a list of inline runset functions.

For example,

```
my_options : inline function ( void ) returning void
{
    . . .
    error_options(...);
    gds_options(...);
    . . .
}
```

- User functions that call runset functions with cell lists and call runset functions used to construct cell lists also must be tagged with the `inline` qualifier. See [Table 1-6](#) for a list of functions.

For example,

```
add_strings_to_list : inline function (string_list : in_out list
ofstrings, new_strings : list of strings) returning void
{
    string_list = string_list.merge(new_strings);
}

get_cell_extent : inline function (layer1 : polygon_layer, cells :
listof string) returning result : polygon_layer
{
    result = cell_extent(layer1, cells);
}

cells : list of string = {};

add_strings_to_list(cells, {"AX", "AY", "AZ"});
add_strings_to_list(cells, {"BX", "BZ"});

part_extent = get_cell_extent(part, cells);
```

If a user function needs to be tagged with the `inline` qualifier but it does not have the qualifier, the runtime error is:

```
error: unable to extract cell list for no explode options
```

[Table 1-5](#) is a list of the runset functions that are tagged with the `inline` qualifier.

**Table 1-5 Runset Functions Tagged With the inline Qualifier**

<code>assign()</code>	<code>assign_edge()</code>
<code>assign_openaccess()</code>	<code>assign_openaccess_edge()</code>
<code>assign_openaccess_text()</code>	<code>assign_text()</code>
<code>equiv_options()</code>	<code>error_options()</code>
<code>exclude_milkyway_cell_types()</code>	<code>exclude_milkyway_net_types()</code>
<code>exclude_milkyway_route_guide_layers()</code>	<code>exclude_milkyway_route_types()</code>
<code>exclude_ndm_blockage_types()</code>	<code>exclude_ndm_design_types()</code>
<code>exclude_ndm_net_types()</code>	<code>exclude_ndm_shape_uses()</code>
<code>gds_options()</code>	<code>get_netlist_connect_database()</code>

**Table 1-5** *Runset Functions Tagged With the inline Qualifier (Continued)*

<code>get_top_cell()</code>	<code>hierarchy_auto_options()</code>
<code>hierarchy_options()</code>	<code>incremental_options()</code>
<code>init_device_matrix()</code>	<code>instance_property_number()</code>
<code>layout_drawn_options()</code>	<code>layout_grid_options()</code>
<code>layout_integrity_by_cell()</code>	<code>layout_integrity_by_marker_layer()</code>
<code>layout_integrity_options()</code>	<code>library()</code>
<code>library_create()</code>	<code>library_import()</code>
<code>lvs_black_box_options()</code>	<code>lvs_options()</code>
<code>milkyway_merge_library_options()</code>	<code>milkyway_options()</code>
<code>ndm_options()</code>	<code>ndm_merge_library_options()</code>
<code>net_options()</code>	<code>oasis_options()</code>
<code>openaccess_options()</code>	
<code>pattern_options()</code>	<code>prototype_options()</code>
<code>read_layout_netlist()</code>	<code>resolution_options()</code>
<code>run_options()</code>	<code>text_options()</code>

[Table 1-6](#) is a list of the runset functions with cell lists and functions used to construct cell lists.

**Table 1-6** *Runset Functions With Cell Lists and Used to Construct Cell Lists*

<code>cell_extent()</code>	<code>cell_extent_layer()</code>
<code>compare()</code>	<code>copy_by_cells()</code>
<code>create_ports()</code>	<code>flatten_by_cells()</code>
<code>text_origin()</code>	

---

## Distributed Processing

Distributed processing provides the following features:

- The performance scalability that the IC Validator tool yields increases with the number of hosts on a single matching.
- Performance is degraded and not scalable when the number of hosts requested is higher than the number of cores on a machine.
- Performance is optimized when your run has 100% of its resource dedicated on a machine (RAM, CPU).
- Performance is optimized on a high speed disk versus a slow speed disk because of I/O.
- Hyperthreading is supported.

Distributed processing is discussed in the following sections:

- [License Requirements](#)
- [Using Multiple Distributed Processing Hosts](#)
- [Distributed Processing Options Configuration File](#)
- [Distributed Processing Learn](#)

---

## License Requirements

For shared licensing, a HERCULES-DP\_MT license is required to enable distributed processing. One HERCULES\_MANAGER license enables one core, and an additional HERCULES-DP\_MT license enables another core.

*Table 1-7 License Requirements*

Situation	Licenses required	Explanation
DRC on 16 cores	16 licenses	1 HERCULES_MANAGER license for 1 processor, Plus1 additional DP_MT license for each additional processor
LVS on 4 cores	4 licenses	Combination of HERCULES_MANAGER and HERCULES-DP_MT
Metal fill on 9 cores	9 licenses	Combination of HERCULES_MANAGER and HERCULES-DP_MT

*Table 1-7 License Requirements (Continued)*

Situation	Licenses required	Explanation
Pattern matching on 62 cores	62 licenses	Combination of HERCULES_MANAGER and HERCULES-DP_MT

For native licensing, distributed processing is enabled by default.

You can run the IC Validator tool with one or more ICValidator2-Manager licenses:

- The first license allows you to use up to eight cores.
- Each additional license enables four additional cores.

See [Table 1-8](#) for examples of the number of licenses needed.

*Table 1-8 Native Licensing for Multicore Distributed Processing*

Situation	Licenses required	Explanation
DRC on 16 cores	3 licenses	First license enables 8 cores, plus 2 additional licenses that enable 4 cores each
LVS on 4 cores	1 license	
Metal fill on 9 cores	2 licenses	First license enables 8 cores, plus 1 core from second license
Pattern matching on 62 cores	15 licenses	First license enables 8 cores, plus 54 cores from 14 licenses

For more information about IC Validator licensing, see the [Licensing and Resource Requirements](#) chapter in the *IC Validator User Guide*.

## Using Multiple Distributed Processing Hosts

By default, the IC Validator tool executes the runset sequentially, using a single distributed processing (DP) host. You can specify multiple distributed processing hosts to make use of multiple systems on the same network, multiple processors on a single system or a combination of the two.

To use multiple distributed processing hosts, the following command-line options are available:

- `-dp#_of_hosts` for multiple processors or cores on a single system. The IC Validator tool executes the runset in dependency order, rather than sequentially.
- `-dp` for multiple processors or cores on multiple systems. The IC Validator tool executes the runset in a dependency order. This command-line option must be used with the `-dphosts` or `-dpopts` command-line options to specify the hosts.

Table 1-2 lists all of the distributed processing command-line options.

## Resolving Automounter Issues

When running in a directory automounted with indirect mounts, the IC Validator tool is unable to determine the correct network path to the working directory. The `-w` command-line option allows this path to be specified.

For example, if a distributed run uses host1, host2, and host3 machines, host3 is used as a file server. The host1 and host2 machines access the files on host3 via the `/net/host3/scratch` directory. If you run on host1 from the `/net/host3/scratch` directory,

```
% icv -dp -dphosts host2 runset
```

the `/tmp_mnt/net/host3/scratch` directory is mounted on host1, that is, the working directory where the `run_details` directory is created. When a distributed process starts on host2, the IC Validator tool attempts to run in the `/tmp_mnt/net/host3/scratch` directory. However, nothing happened on host2 to mount it.

The `-w` command-line option notifies the IC Validator tool of the path to the working directory (`/net/host3/scratch`).

---

## Distributed Processing Options Configuration File

The `dp_options.conf` file contains options used for enabling or disabling IC Validator distributed processing features.

The `dp_options()` function is used in the `dp_options.conf` file.

### Syntax

```
dp_options (
    hosts      = {{hostname = "string", path = "string"}, ...},
    recovery   = true | false                                //optional
);
```



## Arguments

### hosts

Required. Specifies distributed processing hosts to be used for a network distributed run. At least one distributed processing host must be specified.

- `hostname`. Required. Specifies a host name.
- `path`. Optional. Specifies an optional local group directory path.

### recovery

Optional. The default is `true`.

- `true`. Reschedules commands if a host experiences a fatal error. The command is sent to a different distributed processing host for processing.
- `false`. Does not reschedule commands. This causes the IC Validator tool to halt on any host failure.

## Example

```
dp_options (
    recovery = true,
    hosts = {
        {"host1"},
        {"host2", "/SCRATCH/user/group_dir"}
    }
);
```

---

## Distributed Processing Learn

To improve scaling of an IC Validator run that uses distributed processing and multithreading, you can generate adaptive threading information for your runset then apply this information to subsequent runs.

There are two steps in using distributed processing learn (DP learn).

1. Execute the IC Validator tool as a training run.

For the training run, execute your runset using a large farm of machines, each with at least 8 CPUs. This large farm ensures that there is room for threading. For example:

```
% icv -dp32 runset.rs
```

After the training run has completed, the IC Validator tool writes an `estimator.dat` file in the `run_details` directory. The subsequent runs access this file for adaptive threading information.

2. Run the adaptive job.

Use the `-dplearn` command-line option on the command line.

**Important:**

If you run an adaptive run in the same working directory as the training run directory, do not overwrite the `run_details` directory in the training run. Use the `-rd` command-line option to specify a different name for the `run_details` directory for the adaptive job.

For example:

```
% icv -dp32 -dplearn run_details/estimator.dat \  
-rd adapt_run_details runset.rs
```

- The number of distributed processing hosts for the two runs can be different.
- The runset must be the same.
- The input data can vary for each adaptive job.
- The IC Validator tool version must be the same for the training run and the adaptive jobs.

See [“Using IC Validator Command-Line Options With Runset Caching and DP Learn”](#) for information about using command-line options with DP learn.

---

## Runset Caching

The IC Validator runset caching feature reduces the startup time of the IC Validator tool. After a runset is parsed and optimized, the internal data structures for the runset are saved to a runset cache file. Subsequent IC Validator runs that use the same runset load the runset cache file rather than parsing and optimizing the runset again.

You can turn runset caching off by using the `-norscache` command-line option. You might want to use this option when debugging a run.

**Note:**

If you use runset caching with parasitic extraction (PEX) functions for multiple corners, the IC Validator tool does not flag any file naming conflicts until runtime.

See [“Using IC Validator Command-Line Options With Runset Caching and DP Learn”](#) on [page 1-45](#) for information about using command-line options with runset caching.

---

## Cache Files

When the IC Validator tool writes a runset cache file, the following messages are written to screen indicating the path of the cache file:

```
Saved runset to cache: /remote/us54hl/myname/.icvrscache/
e3b32d5de337d888106672bbba370041-rod-3908.rscache
Parsing Finished
Runset Compile Time=0:00:03 User=3.37 Sys=0.16 Mem=186.883
```

When the IC Validator tool reads a runset cache file, the following messages are written to the screen indicating the path of the cache file that was read:

```
Loaded runset from cache:
/remote/us54hl/myname/.icvrscache/
e3b32d5de337d888106672bbba370041-rod-3908.rscache
Cached Runset Compile Time=0:00:00 User=0.25 Sys=0.04 Mem=110.242
```

You can determine which cache file in your cache directory is associated with a specific runset. When you look at the content of the cache file, you'll see something like this text:

```
Cached runset file created from ICV runset:
/remote/seg-scratch/myname/S475264/pxl.ev
. . .
```

The second line is the full path of the runset that was used to generate the cache file.

---

## Configuring Runset Caching

You can use several environment variables to configure runset caching.

**ICV\_RUNSET\_CACHE\_DIR\_SHARED**

Sets the shared runset cache directory used by the IC Validator tool.

```
% setenv ICV_RUNSET_CACHE_DIR_SHARED directory
```

This cache directory is not user-specific. Multiple users can point to the same shared cache directory. When enabled, the IC Validator tool performs the following checks:

- Checks the shared cache directory for cache that either matches or misses the runset criteria
- If missed, the IC Validator tool checks the local cache directory
  - If matched, the IC Validator tool uses the local cache file.
  - If missed, the IC Validator tool compiles and stores cache in the local cache directory (local to the user).

You must move cache files to the shared directory manually so that they can be used by multiple users.

**ICV\_RUNSET\_CACHE\_DIR**

Sets the runset cache directory used by the IC Validator tool when storing cache files.

```
% setenv ICV_RUNSET_CACHE_DIR directory
```

The default runset cache directory is `~/.icvrscache`.

Runset cache directories are user-specific. That is, multiple users cannot point to the same runset cache directory by setting their `ICV_RUNSET_CACHE_DIR` variables to the same directory.

#### ICV\_RUNSET\_CACHE\_MAXFILES

Limits the number of runset cache files that the IC Validator tool writes to the runset cache directory.

```
% setenv ICV_RUNSET_CACHE_MAXFILES num_files_to_keep
```

When the IC Validator tool is run, it does not delete any preexisting runset cache files until either the `ICV_RUNSET_CACHE_MAXFILES` or the `ICV_RUNSET_CACHE_MAXSIZE` limit is reached. When either limit is reached, the IC Validator tool deletes files in the least-recently used order; that is, cache files that have been loaded recently are retained while the older files are deleted.

#### ICV\_RUNSET\_CACHE\_MAXSIZE

Limits the amount of disk space the IC Validator tool can use to store runset cache files. By default, the IC Validator tool limits the space used for runset cache files to 200 MB.

```
% setenv ICV_RUNSET_CACHE_MAXSIZE max_size
```

The *max\_size* value is a numeric value followed by an optional letter indicating the unit type. You can use a unit type of K for kilobytes, M for megabytes, or G for gigabytes. If no unit type is used, the *max\_size* value is assumed to be in megabytes.

#### ICV\_RUNSET\_CACHE\_DEBUG

Reports messages that help you pinpoint a configuration problem or the source of the unexpected behavior. Use this variable if runset caching is behaving in an unexpected way. The messages show configuration parameters and give a reason for loading or not loading each candidate cache file in the runset cache directory.

```
% setenv ICV_RUNSET_CACHE_DEBUG
```

For example,

```
...
RUNSET-CACHE: cache directory : ./RSCACHE/
RUNSET-CACHE: cache max size  : 307200
RUNSET-CACHE: cache max files : -1
RUNSET-CACHE: version string  : <platform> <ICV version> <date>
RUNSET-CACHE: opendir() opened cachedir : ./RSCACHE/
RUNSET-CACHE: readdir() read filename :
3cb51a5a7c3d1da800be5d6ee862da29-rod-16372.rscache
RUNSET-CACHE: 0 cache files matched runset MD5:
```

```
1de3cbcl1c76cd3f7b05877718fa83cae  
RUNSET-CACHE: no matching cache file found!
```

ICV\_DISABLE\_RUNSET\_CACHE

Disables runset caching.

```
% setenv ICV_DISABLE_RUNSET_CACHE
```

You can also use the `-norscache` command-line option.

---

## Matching a Runset to a Cache File

The IC Validator tool matches a runset to an existing runset cache file only when the following criteria are met:

- Preprocessing directives must match the directives stored in the cache file. Some of these directives are internally generated by the IC Validator tool, but others come directly from the IC Validator command line via the `-D`, `-U`, and `-I` options.
- The `-svn`, `-uvm`, `-svc`, `-uvc`, `-sfn`, and `-ufn` command-line options have not changed between runs.
- The preprocessed runset must match that of the cache file, as determined by a hashing algorithm.
- The current IC Validator version must match that of the cache file.
- The endianness of the architectures that IC Validator is run on during the cache save and the current run (the cache load attempt) must match. For example, AIX and Sun are big endian platforms and, therefore, can share a cache file.
- UNIX environment variables referenced in the runset must have the same values as they had when the cache file was generated.

Note:

Using the `-ln` and `-lnf` to override filenames and formats specified in the `layout_file` argument of the `read_layout_net()` function does not cause cache misses.

---

## Using IC Validator Command-Line Options With Runset Caching and DP Learn

After a runset is run the first time with runset caching enabled, subsequent runs of the same runset normally result in a cache file being read rather than a full parse of the runset. For DP learn, the `estimator.dat` file is used for subsequent runs.

The content of the runset can be altered by certain IC Validator command-line options, which affects the use of the `-dplearn` option and runset caching. These changes cause the IC Validator tool to write a new cache or estimator.dat file. Anytime the runset is modified, DP learn and runset caching are affected.

Note:

Options not specifically mentioned in the following sections are generally compatible with runset caching and DP learn.

---

## Compatible Command-Line Options

The following IC Validator options are compatible with runset caching and DP learn. These options can be added, removed, or modified on the command line. The IC Validator tool still uses the previously generated cache or estimator.dat file.

- c
- g
- i
- p
- f
- rd
- stc
- dp
- dpcache
- dphosts
- dpopts

---

## Command-Line Options That Affect Runset Caching and DP Learn

The following options cause changes in the content of the preprocessed runset. Therefore, when you add these options to the command line or change their arguments, the preprocessed runset does not hash to the same cache file. The runset parses normally and writes a new cache file. If these options are not changed, subsequent runs of the same runset hash to this cache file. For DP learn, a new estimator.dat file is needed.

- 64
- D
- U
- il
- iln
- sfn
- svc
- svn
- ufn

```
-uvc  
-uvn  
-verbose
```

---

## Incompatible Command-Line Options

Because of the unique flows used to handle the following options, these options disable runset caching and DP learn when used:

```
-checkpoint  
-debug  
-debug_source  
-ndg  
-snapshot  
-snapshot_resume
```

The following option affects runset caching and DP learn because it changes internal options in the parser. When you use this option, the IC Validator tool generates a separate cache file or a new estimator.dat file:

```
-nro
```

---

## Native LSF and SGE Support

You can use native LSF and SGE batch queuing systems to run IC Validator jobs. An IC Validator distributed processing license is required when you use LSF or SGE.

Note:

To successfully use LSF or SGE when running the IC Validator tool, you must already know how to use these batch queuing systems.

Two IC Validator batch distributed processing command-line options support the use of LSF or SGE. Use either of these options to specify the batch queuing system command that launches the IC Validator job:

- `-dpbatchstr`. Specifies an LSF or SGE command string. The command string must be delimited by quotation marks. The syntax is  

```
-dpbatchstr "command"
```
- `-dpbatchfile`. Specifies the path and file name of an LSF or SGE command file. The text file contains the LSF or SGE command that is read by the IC Validator tool. The syntax is  

```
-dpbatchfile /path/file.txt
```

**Note:**

Do not use the `-dpbatchstr` and `-dpbatchfile` options together on the command line. Additional IC Validator command-line options must be placed after these batch distributed processing command-line options.

The following are examples of LSF or SGE commands for specifying the needed resources of the LSF or SGE compute farm:

- Run 10 CPUs

```
bsub -n 10 -J jobname -o stdfilefilename
```

- Run 20 CPUs

```
bsub -n 20 -J jobname -o stdoutfilename
```

When either of the batch distributed processing options are on the IC Validator command line, the tool executes a Perl script, `native_batch_script.pl`, that uses the specified LSF or SGE commands. The script has a section that executes the `icv_slave` program on all machines. Another section launches the `icv_master` program and, when all the slaves are running, executes the IC Validator tool on the first machine. After communication is established between master and slave, the Perl script finishes.

**Note:**

The `native_batch_script.pl` script must either be in your path or in the current run directory.

Before running the IC Validator tool using LSF or SGE, you must source the LSF or SGE profile that contains all the necessary environment variables to run in LSF or SGE mode.

1. Determine the path for the profile file. For example,  

```
/global/cust_apps/lsf/myloc/profile.lsf
```
2. Set the `$LSF_ENVDIR` environment variable to the profile path. This environment variable is critical for running the requested LSF or SGE machines.
3. Add the `$LSF_ENVDIR` environment variable to the `.cshrc` file. This gives the requested distributed machines a common profile.

On the IC Validator command line, specify the LSF or SGE command using one of the batch distributed processing options. For example,

```
% icv -dpbatchstr "bsub -n 20 -J jobname -o stdoutfilename" runset.rs
```

```
% icv -dpbatchfile /path/lsfcommand.txt runset.rs
```

The `native_batch_script.pl` script gives you flexibility in how you implement the batch queuing system. If a custom configuration is required, you can modify the Perl script. The Perl script is in the `contrib` directory that is included with the IC Validator release. The `contrib` directory is at the top level of the release install directory.



**Note:**

The scripts in the contrib directory are not supported. See the README file in the contrib directory for more information.

---

## Layer Mapping

There are two types of layer mapping:

- [Runset Layer Mapping](#): This mapping modifies your *runset* to create a new runset.
- [Data Layer Mapping](#): This mapping modifies the *data* as it is read in by changing the layer and datatype attributes of the data.

You can use both types of mapping when you are running on multiple libraries.

---

### Runset Layer Mapping

Runset layer mapping is the process of substituting layer/datatype numbers that are specified in the ASSIGN section of a runset with the actual layer/datatype numbers of data in a database. You need to map layers, for example, when the layer/datatype numbers in your database are not consistent with the layer/datatype numbers specified in a foundry supplied runset.

Specify the mapping file with the `-lf` command-line option. See [“Command-Line Options” on page 1-6](#).

The first line of the mapping file must always be

```
RUNSET(*,*) = Library(*,*)
```

This statement maps each layer to itself (identity mapping). This identity mapping ensures that there is a mapping for each layer. Subsequent lines of the mapping file override the identity map for selected layers.

For example,

```
Runset(*,*) = Library(*,*)  
Runset(15,0) = Library(1,0)  
Runset(16,0) = Library(2,0)  
Runset(17,0) = Library(3,0)
```

Mapping is specified within the file using the following formats:

**Note:**

`Runset(i,j)` refers to layer/datatype numbers in the ASSIGN section of the runset.

`Library(i,j)` refers to the actual layer/datatype numbers of the data in the database.

- `Runset(A,B) = Library(C,D)`

In the following example, an assign reference to layer/datatype (1,5) is replaced with layer/datatype (2,0).

```
Runset(1,5) = Library(2,0)
```

- `Runset(A,B) = Library(C1...Cm,D1...Dn)`

In the following example, any assign reference to layer/datatype (5,0) is replaced with the four layer/datatype pairs (10,0), (10,4), (12,0), and (12,4).

```
Runset(5,0) = Library(10 12,0 4)
```

- `Runset(A1...Am,B1...Bn) = Library(C1...Cm,D1...Dn)`

**Note:**

The number of Runset layer elements and Library layer elements must be equal.

The number of Runset datatype elements and Library datatype elements must be equal.

In the following example, four layer/datatype substitutions are made:

```
assign (2,6) is replaced with (10,0)
assign (2,8) is replaced with (10,4)
assign (4,6) is replaced with (12,0)
assign (4,8) is replaced with (12,4)
```

```
Runset(2 4,6 8) = Library(10 12,0 4)
```

The syntax rules for a mapping file are

- Wildcards are allowed in place of a list. For example,

```
Runset(1,*) = Library(2,*)
```

means that assign layer 1, all datatypes, is replaced with layer 2, all datatypes.

- Layer/datatype ranges are specified by a dash. For example,

```
Runset(2 5-8,*) = Library(10-12 17 20,*)
```

- List elements must be in ascending order.

When the IC Validator tool is executed with a layer mapping file, any substitution of assign layer/datatype numbers is reported below the assign statement in the *cell.sum* file. Here is an example report:

```
L1_all = assign({ { 11 } })
Function: assign
Inputs: L1_all.polygonlayer.0001
Outputs: L1_all.polygonlayer.0002
Layer list {{11}} was replaced with {{1}}
```

---

## Data Layer Mapping

Use data layer mappings when you merge multiple libraries with corresponding data on different layers or datatypes. For example, if you have a main design library with metal1 on layer 41 and a secondary library with metal1 on layer 31, you need to merge these libraries using data layer mapping so that all metal1 data is read in on the same layer. This layer is layer 41 in this example.

Typically, you specify the main design library with the `library()` function. The `library()` function does not provide data layer mapping capabilities for the main library. However, data layer mapping is available when you specify additional libraries. Therefore, you can map the data from the additional libraries to the corresponding layers and datatypes used by the main design library.

If you use the `library_create()` function instead of the `library()` function, all of the libraries are specified with the `library_import()` function and have data layer mapping available. For the purposes of data layer mapping, you can use any library as the main library and layer map the remaining libraries to that main library.

In addition to data layer mapping, you can also use runset layer mapping in the same run to map the runset layers to the layers used by your main library. The following examples illustrate this scenario.

In this example, the main and secondary libraries are specified in the `library()` function:

- The main library, `one.oasis`, is specified with the `library()` function. It has metal1 on layer 41.
- A secondary library, `two.oasis`, has metal1 on layer 31.
- The runset assumes that metal1 is on layer 5.

The metal1 data on layer 31 in the `two.oasis` library is mapped to layer 41 as it is read in. This mapping ensures that all input metal1 data is read onto layer 41. The runset layer mapping then transforms the `assign({{5}})` assignment into `assign({{41}})` to ensure that the run executes as expected. Here is a snippet of the runset:

```
library(library_name    = "one.oasis",
        cell           = "A",
        format          = OASIS,
        merge_libraries = {{library_name = "two.oasis", format=OASIS,
                                layer_map_file="two_layer_map"}});
```

```
metall = assign({{5}});
```

The `two_layer_map` file has the data layer mapping for `two.oasis`:

```
41  31
```

The runset layer mapping file, specified with the `-lf` command-line option, has this mapping:

```
RUNSET(*,*) = LIBRARY(*,*)
RUNSET(5,*) = LIBRARY(41,*)
```

In the following example, the main library is specified in the `library()` function and the secondary library in the `library_import()` function. Use this method to import multiple libraries when you do not want cell merging to occur. Each library remains distinct from the others so that your assigns functions can import data from one library but not another.

- The main library, `one.oasis`, is specified with the `library()` function. It has `metal1` on layer 41.
- A secondary library, `two.gds`, has `metal1` on layer 31.
- The runset assumes that `metal1` is on layer 5.

The `two_layer_map` file and the runset layer mapping are the same as in the previous example. Here is a snippet of the runset:

```
lib1 = library(library_name = "one.oasis",
               cell         = "A",
               format        = OASIS);

lib2 = library_import(library_name = "two.gds",
                     cell         = "A",
                     cell_prefix  = "TWO_",
                     format       = GDSII,
                     layer_map_file = "two_layer_map");

metall = assign({{5}});
```

In the following example, the main library is specified in the `library()` function and the secondary library in the `milkyway_merge_library_options()` function. The libraries are not merged nor kept separate. Milkyway cell data is replaced with the equivalent GDSII data on a cell-by-cell basis. See the [milkyway\\_merge\\_library\\_options\(\)](#) function for more information.

- The main Milkyway library, `one`, is specified with the `library()` function. It has `metal1` on layer 41.
- A secondary GDSII library, `two.gds`, has `metal1` on layer 31.
- The runset assumes that `metal1` is on layer 5.

The `two_layer_map` file and the runset layer mapping are the same as in the first example. Here is a snippet of the runset:

```
library(library_name = "one",
        cell         = "A",
```

```

        format          = MILKYWAY,
        library_path = ".");

milkyway_merge_library_options(
    libraries = {{"one", "./", {{"two.gds", GDSII,
                                layer_map_file="two_layer_map"}}}},
    missing_cell = ABORT);

metall = assign({{5}});

```

---

## Using PXL Runset Encryption

Runset encryption is a method used to conceal the content of the runset from an end user. Use encryption to limit the end user's knowledge of foundry process parameters or of the steps necessary to perform certain design rule checks or transformations.

### Operation

After a runset is created as plain text, certain content can be transformed into encrypted blocks. The content of the encryption blocks can contain PXL constructs and any of the standard preprocessor directives. Plain text is also referred to as clear text.

The `pxlencrypt` utility is used to encrypt the runsets. It encrypts only blocks of cleartext. This utility is included in the IC Validator release install directory. The syntax is

```
% pxlencrypt cleartext_runset encrypted_runset
```

The `pxlencrypt` utility cannot reverse the encryption process.

The encrypted runset is then delivered to the end user, who executes the IC Validator tool using it.

All runset functions that are encrypted are reported as “encrypted function” in the summary file and on standard output when the `-verbose` option is used. Although the functions are listed as encrypted functions, the machine usage, number of polygons generated, and names of smart-deleted layers (layers that are no longer needed because nothing depends on them) are still reported.

### Encryption Directives

The directives that mark the beginning and ending points of an encryption section are

```
#pragma PXL encrypt begin
#pragma PXL encrypt end
```

More than one section of a runset can be encrypted in a runset. However, an encrypted section cannot be within another encrypted section. Here is an example of an encryption section in a runset:

```
...
#pragma PXL encrypt begin
e2 @= { @ "e2 violation";
        external2( l1, l2, distance < 2.0, extension = NONE,
                   look_thru = COINCIDENT);}
#pragma PXL encrypt end
...
```

Partial constructs cannot be encrypted. For example, this partial construct is not allowed:

```
...
external2( l1, l2, distance < 2.0, extension = NONE,
#pragma PXL encrypt begin
        look_thru = COINCIDENT);}
#pragma PXL encrypt end
...
```

The pxlcrpt utility automatically breaks encryption blocks around included files. For example, if the runset has this code:

```
#pragma PXL encrypt begin
...
#include "file1.rh"
...
#pragma PXL encrypt end
```

Then, the code is encrypted as:

```
#pragma PXL encrypt begin
...
#pragma PXL encrypt end
#include "file1.rh"
#pragma PXL encrypt begin
...
#pragma PXL encrypt end
```

# 2

## Licensing and Resource Requirements

---

*This chapter contains the setup instructions for the IC Validator tool. Make sure that you have all the correct directories and files, that your environment accommodates the IC Validator tool, and that you have incorporated the proper licensing information.*

The licensing and resource requirements are described in the following sections:

- [Licensing Schemes](#)
- [License Waiting](#)
- [Which IC Validator Licenses Do I Need?](#)
- [Resource Requirements](#)

For additional information about Synopsys licensing and installing the IC Validator tool, see the following documents:

- The Synopsys Licensing QuickStart Guide, available at <http://www.synopsys.com/Support/LI/Licensing/Pages>.
- Information about installing the IC Validator tool is available at <http://www.synopsys.com/Support/LI/Installation/Pages/default.aspx>.

---

## Licensing Schemes

The IC Validator tool has two types of licensing schemes. The IC Validator tool is run using one of these schemes:

- Native licensing
- Shared licensing

Native licensing is the default scheme. If the job does not use license waiting or the `-hlic` and `-ilic` command-line options, the IC Validator tool first checks for native licenses. If native licenses are not available, the tool checks for shared licenses. If neither is available, the job ends with a license denied error.

The two licensing schemes can coexist on a system as available licensing schemes, but an IC Validator job uses only one of them. They cannot be used together.

If both types of licenses are available, you can use the `-hlic` and `-ilic` command-line options to select the licensing scheme to use for a specific job. See [“Command-Line Options” on page 1-6](#) for more information.

---

## License Waiting

The IC Validator tool supports license waiting in accordance with the standard Synopsys Common Licensing (SCL) guidelines. To enable this function, you must set the `ICV_LICENSE_WAIT` environment variable. No argument is required for this environment variable.

When you run the IC Validator tool with license waiting,

- If only native licensing is installed, the job queues for native licenses.
- If only shared licensing is installed, the job queues for shared licenses.
- If both native and shared licensing are installed,
  - If you use the `-hlic` command-line option, the job queues for shared licenses. Native licenses are ignored.
  - If you do *not* use the `-hlic` command-line option, the job queues for native licenses. Shared licenses are ignored.

Note:

There is no timeout when you use the `ICV_LICENSE_WAIT` environment variable. The tool waits until a license becomes available.



---

## Which IC Validator Licenses Do I Need?

The IC Validator functions in your runset determine the required licenses. The following sections discuss the needed licenses:

- [Native Licensing](#)
- [Shared Licensing](#)

---

### Native Licensing

Native licensing uses the licenses as shown in [Table 2-1](#). ICValidator2-Manager is the main license. It is checked out every time the IC Validator tool is run.

*Table 2-1 Native Licenses*

Technology	License required
DRC functions	ICValidator2-GeometryEngine
Data creation functions	ICValidator2-GeometryEngine
LVS extraction functions	ICValidator2-GeometryEngine
LVS compare functions	ICValidator2-CompareEngine
Metal fill functions	ICValidator2-GeometryEngine
In-design technology	ICValidator2-GeometryEngine
Pattern matching functions	ICValidator2-GeometryEngine
Double-patterning technology (DPT) functions 20 nm DRC verification and automatic repair in the IC Compiler tool	ICValidator2-GeometryEngine
Fill-to-target functions	ICValidator2-GeometryEngine
VUE	ICValidator2-GeometryEngine <sup>1</sup>
NetTran and other utilities	ICValidator2-GeometryEngine <sup>1</sup>

*Table 2-1 Native Licenses (Continued)*

Technology	License required
Error database (PYDB) queries	ICValidator2-GeometryEngine <sup>1</sup>
Milkyway library read and write functions	Milkyway

1. Only checks for the presence of a valid license. No license is checked out.

## Multicore Licensing

The IC Validator tool can be run with one or more ICValidator2-Manager licenses:

- The first license allows you to use up to eight cores.
- Each additional license enables four additional cores.

See [Table 2-2](#) for examples of the number of licenses needed.

*Table 2-2 Multicore Licensing Examples*

Situation	Licenses required
DRC on 16 cores	3 licenses (first license enables 8 cores, plus 2 additional licenses that enable 4 cores each)
LVS on 4 cores	1 license
Metal fill on 9 cores	2 licenses
Pattern matching on 62 cores	15 licenses

## Shared Licensing

Shared licensing is based on individual functions. Every time you run the IC Validator tool using shared licensing, the tool uses a HERCULES\_MANAGER license. In addition to this license, the IC Validator tool uses licenses based on functions in your runset. The following sections list functions that require an additional IC Validator license.

## HERCULES\_DEVICE

The IC Validator tool requires a HERCULES\_DEVICE license when you use any of the functions listed in [Table 2-3](#).

*Table 2-3 Commands Requiring HERCULES\_DEVICE License*

buildsub()	extract_devices()	netlist()
pex_generate_lpp_map()	pex_generate_process_map()	pex_generate_results()
write_annotation_file()	write_spice()	write_xref_spice()

## HERCULES\_DRC

The IC Validator tool requires a HERCULES\_DRC license when you use any of the functions listed in [Table 2-4](#).

*Table 2-4 Commands Requiring HERCULES\_DRC License*

adjacent_edge()	area()	aspect_ratio()
center_to_center1()	center_to_center1_edge()	center_to_center2()
center_to_center2_edge()	contained_by()	covered_by()
delta_edge()	density()	enclose()
enclose_corner()	enclose_corner_edge()	enclose_edge()
enclose_error()	external_corner1()	external_corner1_edge()
external_corner2()	external_corner2_edge()	external1()
external1_edge()	external1_error()	external2()
external2_edge()	external2_error()	gradient_density()
internal_corner1()	internal_corner1_edge()	internal_corner2()
internal_corner2_edge()	internal1()	internal1_edge()
internal1_error()	internal2()	internal2_edge()
internal2_error()	length_edge()	
not_adjacent_edge()	not_area()	not_aspect_ratio()

*Table 2-4 Commands Requiring HERCULES\_DRC License (Continued)*

<code>not_contained_by()</code>	<code>not_covered_by()</code>	<code>not_delta_edge()</code>
<code>not_enclose_edge()</code>	<code>not_external1_edge()</code>	<code>not_external2_edge()</code>
<code>not_internal1_edge()</code>	<code>not_internal2_edge()</code>	<code>not_length_edge()</code>
<code>not_rectangles()</code>	<code>rectangles()</code>	<code>smartstep_density()</code>
<code>wide()</code>	<code>wide_edge()</code>	

## HERCULES\_ERC

The IC Validator tool requires a HERCULES\_ERC license when you use any of the functions listed in [Table 2-5](#).

*Table 2-5 Commands Requiring HERCULES\_ERC License*

<code>device_connected_to()</code>	<code>device_net_count()</code>	<code>device_not_connected_to()</code>
<code>net_device_count()</code>	<code>net_path_check()</code>	<code>net_polygon_select()</code>
<code>net_property_select()</code>	<code>net_select()</code>	
<code>sconnect()</code>	<code>soft_check()</code>	

## HERCULES\_LVS

The IC Validator tool requires a HERCULES\_LVS license when you use the `compare()` function.

## HERCULES\_MASK

The IC Validator tool requires a HERCULES\_MASK license when you use any of the functions listed in [Table 2-6](#).

*Table 2-6 Commands Requiring HERCULES\_MASK License*

<code>and()</code>	<code>and_edge()</code>	<code>and_overlap()</code>
<code>angle_edge()</code>	<code>assign()</code>	<code>assign_edge()</code>
<code>assign_openaccess()</code>	<code>assign_openaccess_edge()</code>	<code>assign_openaccess_text()</code>
<code>assign_text()</code>	<code>cell_extent()</code>	<code>cell_extent_layer()</code>

*Table 2-6 Commands Requiring HERCULES\_MASK License (Continued)*

chip_extent()	coincident_edge()	coincident_inside_edge()
coincident_outside_edge()	contains()	copy()
copy_by_cells()	copy_edge()	copy_error()
cutting()	data_limit()	data_limit_edge()
donut_holes()	donuts()	edge_extents()
edge_grow()	edge_shrink()	edge_size()
empty_layer()	empty_layer_edge()	empty_violation()
enclosing()	error_merge()	error_merge_edge()
extend_edge()	extent()	flatten_by_cells()
grouped_by()	grow()	inside()
inside_hole()	inside_point_touching_edge()	inside_touching_edge()
interacting()	interacting_edge()	intersections()
layer_extent()	layer_statistics()	layout_integrity_by_cell()
layout_integrity_by_marker_layer()	level()	level_edge()
level_to()	level_to_edge()	library()
library_create()	library_import()	mos_select()
move()	move_edge()	negate()
negate_in_window()	not()	not_angle_edge()
not_coincident_edge()	not_coincident_inside_edge()	not_coincident_outside_edge()
not_contains()	not_cutting()	not_donuts()
not_edge()	not_enclosing()	not_extent()
not_inside()	not_inside_hole()	not_inside_point_touching_edge()

*Table 2-6 Commands Requiring HERCULES\_MASK License (Continued)*

not_inside_touching_edge()	not_outside_point_touching_e dge()	not_point_touching_edge()
not_interacting()	not_interacting_edge()	not_outside()
not_outside_touching()	not_outside_touching_edge()	not_rectangles_interacting()
not_touching()	not_touching_edge()	not_vertices()
not_vertices_edge()	off_grid()	off_grid_xy()
or()	or_edge()	outside()
outside_point_touching_edge()	outside_touching()	outside_touching_edge()
point_touching_edge()	polygon_extents()	pull_down()
pull_down_edge()	pull_down_to()	pull_down_to_edge()
rectangle_overlap()	rectangles_interacting()	remove_fill()
replace_text()	res_select()	shrink()
size()	size_inside()	size_outside()
size_overlap()	snap()	snap_edge()
text_origin()	touching()	touching_edge()
vertex()	vertex_edge()	vertices()
vertices_edge()	xor()	xor_edge()

## NET-TRAN

The IC Validator tool requires a NET-TRAN license when you use any of the functions listed in [Table 2-7](#).

*Table 2-7 Commands Requiring NET-TRAN License*

read_layout_netlist()	schematic()
-----------------------	-------------

## HERCULES-NETLIST

The IC Validator tool requires a HERCULES-NETLIST license when you use any of the functions listed in [Table 2-8](#).

*Table 2-8 Commands Requiring HERCULES-NETLIST License*

netlist()	write_annotation_file()	write_spice()
write_xref_spice()		

## HERCULES-DP\_MT

This license enables distributed processing in the IC Validator tool.

## Resource Requirements

The initial database file and the files that the IC Validator tool creates and uses determine the system requirements. With some knowledge of the design to be checked, you can estimate physical memory, disk space, and possible swap space requirements for your IC Validator runs.

A contrib directory, which is included with IC Validator releases, provides scripts for analyzing IC Validator performance. The contrib directory is at the top level of the release install directory. *These scripts are not supported.* See the README file in the contrib directory for more information.

## Space Requirements

The IC Validator program performs layout checking operations on group files that are created from the design database and stored in the group directory. The IC Validator tool first creates the primary group files, one file for each layer in an assign function of the runset. Each primary group file is a database describing that layer's existence throughout the design, maintaining all hierarchy. As the IC Validator run progresses, output files are generated from certain checks. These outputs are also written as group files. For example, the files in a group directory could be:

HRCHY.ERROR	inst_indx.dat	pgate.group	toxcont.group
cont.group	met1.group	poly.group	via.group
gate.group	met2.group	psel.group	well.group
inst_data.dat	ngate.group	tox.group	

A general rule is to have a minimum of two to three times the space of the initial database, if it is in GDSII file format, reserved for group file creation, assuming the design has a good

hierarchical style. A mostly flat design, or one with unwanted data interactions, might require four to six times the initial database space for its group file creation. Thus, for a GDSII file database of 10 MB, a minimum of 20 to 30 MB is required for group file creation space, if the design is hierarchical in nature. By contrast, at least 40 to 60 MB are required for a flat design.

In addition to the group files, the other significant output files from the IC Validator run are the run summary files and the output database files. The summary files are the *cell.LAYOUT\_ERRORS* and *cell.sum* files. These files are ASCII log files that document the results of the IC Validator run on the cell named in the runset.

The size of the *cell.LAYOUT\_ERRORS* file and the output database depends on the number of errors encountered in the IC Validator run. As with the group files, if the design is hierarchical, the graphical output database is smaller than the graphical output for a more flat design. The graphical output database is smaller because the IC Validator tool only needs to report errors internal to repeated modules one time, while for a flat design the error is reported for every instance of the module.

---

## Memory Usage

You can manage the memory space needed for error output reporting by using a logical approach to locating errors. Start by running the IC Validator tool on small modules of the design and address errors as they are encountered. After the errors are fixed, run the IC Validator tool on hierarchically higher modules. Using this procedure keeps your *cell.LAYOUT\_ERRORS* file and output database reasonably small. Working up through the hierarchy in this way manages the use of memory needed for error output storage space.

The IC Validator tool also requires physical memory and swap space to complete its checks. The tool attempts to load into memory all data required to complete the checks currently being performed. Any overflow of physical memory uses swap space. Generally speaking, any time the IC Validator program needs to access swap space, you can expect performance degradation, especially if swap space is distributed over a large network with considerable network traffic. If enough physical memory is available to load all data and complete a check without accessing swap space, you can expect maximum throughput. A machine with three to four times as much memory as the GDSII file size should perform well with a hierarchical design. Flat designs require more hardware. Set the swap space so that any overflow of physical memory does not exhaust the swap space and cause the IC Validator run to terminate.

The output *cell.sum* file includes a list of the memory used and the number of page faults required to complete each check. If you see that certain checks require more memory than the machine has allocated, causing large amounts of data to be swapped out to disk, you might need a machine with more memory for future runs. [Example 2-1](#) shows an example of the reported memory usage. The following section, “[Runtimes and Memory Use Report](#),” gives information about the reported runtimes and memory usage.



**Example 2-1 Partial Example of Memory Usage Information in cell.sum File**

```

length_edge() at sf.rs:162
gate_gt_3_5 = length_edge(gate_e, distance > 3.5)
  Function: length_edge
  Inputs: gate_e.edgelay.0001
  Outputs: gate_gt_3_5.edgelay.0001
8 unique edge sets written.
  Check time = 0:00:00 User=0.00 Sys=0.00 Mem=15.269
external2() at sf.rs:166
external2(gate_le_1_5, POLY, distance < 0.15, extension = RADIAL,
look_thru = COINCIDENT)
  Comment: "R.POLY.G1: Min Space POLY to Gate for Gate Width <=1.5 must be > 0.15"
  Function: external2
  Inputs: gate_le_1_5.edgelay.0001, POLY.polygonlayer.0002
WARNING - 6 spacing violations found.
  Check time = 0:00:00 User=0.00 Sys=0.00 Mem=16.393
external2() at sf.rs:169
external2(gate_gt_1_5_le_2_5, POLY, distance < 0.25, extension = RADIAL,
look_thru = COINCIDENT)
  Comment: "R.POLY.G2: Min Space POLY to Gate for 1.5 < Gate Width <=2.5 must be
> 0.25"
  Function: external2
  Inputs: gate_gt_1_5_le_2_5.edgelay.0001, POLY.polygonlayer.0002
WARNING - 6 spacing violations found.
  Check time = 0:00:00 User=0.00 Sys=0.00 Mem=16.409
external2() at sf.rs:172
external2(gate_gt_2_5_le_3_5, POLY, distance < 0.35, extension = RADIAL,
look_thru = COINCIDENT)
  Comment: "R.POLY.G3: Min Space POLY to Gate for 2.5 < Gate Width <=3.5 must be
> 0.35"
  Function: external2
  Inputs: gate_gt_2_5_le_3_5.edgelay.0001, POLY.polygonlayer.0002
WARNING - 3 spacing violations found.
  Check time = 0:00:00 User=0.00 Sys=0.00 Mem=16.409
external2() at sf.rs:175
external2(gate_gt_3_5, POLY, distance < 0.45, extension = RADIAL,
look_thru = COINCIDENT)
  Comment: "R.POLY.G4: Min Space POLY to Gate for Gate Width > 3.5 must be > 0.45"
  Function: external2
  Inputs: gate_gt_3_5.edgelay.0001, POLY.polygonlayer.0002
WARNING - 6 spacing violations found.
  Check time = 0:00:00 User=0.00 Sys=0.00 Mem=16.409

```

For maximum performance on a networked system, keeping as many files as required on the local machine results in faster completion. Enough disk space local to the machine on which the job is run should be available to accommodate the Milkyway library (or, GDSII or OASIS files), the group files being created, and all ASCII or physical error files being created.

## Runtimes and Memory Use Report

This section has information about the runtime and memory usage reported in the summary file for each stage of an IC Validator run. Note that some processing falls outside of the cumulative memory reporting.

Note:

Generally, only memory allocated by the IC Validator tool is reported in the peak usage or accumulative memory usage. For example, Milkyway memory is allocated by the Milkyway library, so it is not included in the report of IC Validator peak memory usage or machine accumulative memory usage.

Table 2-9 shows the definitions of the times and memory usage terms used in the summary file.

Table 2-9 Terms Used in the Summary File

Term	Definition
Time	Wall time
User	CPU time for calculation
Sys	CPU time for I/O
Mem	Maximum memory usage

The following examples are from different stages in a summary report:

### 1. Runset compile stage

- When the IC Validator tool is run without runset caching, the report looks like this:

```
Runset Compile Time=0:01:03 User=61.08 Sys=0.81 Mem=1091.016
```

- When the IC Validator tool is run with the `-rscache` option for runset caching, the report looks like this:

```
Cached Runset Compile Time=0:00:04 User=0.81 Sys=0.25 Mem=207.574
```

### 2. Distributed processing setup stage

```
Virtual Machine Init Time=0:00:02 User=1.09 Sys=0.12 Mem=1091.016
System Startup Time=0:00:02 User=0.00 Sys=0.52 Mem=1091.016
```

### 3. Preprocessing stages

- Hierarchy reading and optimization stage:** The Milkyway and OASIS memory usage is not accounted for in the peak memory report.

```
Oasis memory used: 27.0M
Milkyway memory usage: 4714.789M
Reading hierarchy Time=0:00:10  User=8.29 Sys=2.87 Mem=212.158
Updating hierarchy Time=0:00:00  User=0.07 Sys=0.00 Mem=58.059
Exploding Time=0:00:00  User=0.44 Sys=0.02 Mem=59.246
```

- **Virtual cell pairing stage**

```
Pairing Iteration 1 Time=0:00:00  User=0.53 Sys=0.04 Mem=100.541
Pairing Iteration 2 Time=0:00:00  User=0.54 Sys=0.05 Mem=99.556
Pairing Iteration 3 Time=0:00:00  User=0.51 Sys=0.04 Mem=98.572
Pairing Iteration 4 Time=0:00:00  User=0.45 Sys=0.03 Mem=100.588
Pairing Time=0:00:01  User=2.03 Sys=0.16 Mem=100.588
Combined VCELL Time=0:00:02  User=2.67 Sys=0.23 Mem=100.588
Post VCell Time=0:00:00  User=0.12 Sys=0.00 Mem=61.480
Prototype Pass 2 Time=0:00:00  User=0.04 Sys=0.00 Mem=62.620
```

- **Regioning stage**

```
Determine Region Time=0:00:00  User=0.02 Sys=0.00 Mem=60.620
```

- **Create Layer stage**

```
Create Layer Setup Time=0:00:00  User=0.06 Sys=0.01 Mem=59.605
Hierarchy Cleanup Time=0:00:01  User=0.89 Sys=0.04 Mem=60.620
```

#### 4. Command execution

- **Command statistic report**

```
Check Time=0:00:00  User=0.00 Sys=0.00 Mem=17.146
```

- **Reading error hierarchy**

```
Elapsed Time=0:00:00  User=0.08 Sys=0.00 Mem=24.419
```

- **text\_net()**

```
Total Text Time=0:00:00  User=0.05 Sys=0.00 Mem=65.956
```

- **extract\_devices()**

```
Device Connect Time=0:00:04  User=1.41 Sys=2.84 Mem=150.382
```

- **netlist()**

```
Netlisting Time=0:00:03  User=1.63 Sys=0.68 Mem=71.073
```

- **write\_milkyway():** The Milkyway memory usage is not accounted for in the peak memory report.

```
Milkyway memory usage: 4129.207M
Compression Time=0:05:07  User=275.34 Sys=27.80 Mem=3613.472
Writing Time=0:10:32  User=516.55 Sys=47.36 Mem=3903.721
```

- **write\_oasis():** The OASIS memory usage is not accounted for in the peak memory report.

```
Merging input layout from <input OASIS>
Merge Time=0:00:00  User=0.00 Sys=0.00 Mem=25.184
Writing Time=0:00:00  User=70.00 Sys=0.00 Mem=25.184
```

- **write\_gds():**

```
Writing Time=0:00:00  User=0.00 Sys=0.00 Mem=25.184
```

## 5. Host summary

- **Per host summary of total runtime and peak memory usage.** This time excludes memory usage for reading and writing Milkyway data and OASIS files.

```
Host number 0 total check Time=0:04:40  User=182.60 Sys=37.42
Mem=810.255
```

- **Per host summary of total runtime.** This time includes preprocessing and peak memory usage but excludes memory usage for reading and writing Milkyway data and OASIS files.

```
Host number 0 total check and preprocess Time=0:04:56  User=195.38
Sys=40.62 Mem=810.255
```

## 6. Error writing summary report

```
Overall error storage time: User=0.06 Sys=0.11 Mem=14.640
Generation Time=0:00:00  User=0.01 Sys=0.00 Mem=15.006
```

## 7. Overall memory and runtime reports

- **Per machine accumulative peak memory usage.** This memory usage excludes reading and writing Milkyway data and OASIS files.

```
IC Validator Peak Machine Memory Report
igcae209      : Peak = 2111(Mb)
igcaew008     : Peak = 3904(Mb)
igcae191     : Peak = 1303(Mb)
igcae177     : Peak = 1430(Mb)
igcae188     : Peak = 1437(Mb)
igcae205     : Peak = 1300(Mb)
igcae178     : Peak = 1433(Mb)
igcae174     : Peak = 1438(Mb)
```

- **Overall runtime and peak memory report that is for all stages and functions.** This report excludes memory usage for reading and writing Milkyway data and OASIS files.

```
Overall engine Time=0:06:12 Highest command Mem=810.255
```

- **Master memory usage.** This usage is not accounted for in the peak memory report.

```
Overall Master Mem=251.453
```

---

## Operating Systems

The IC Validator tool supports many hardware architectures and operating systems. See the installation guide for a list of the supported operating systems:

<http://www.synopsys.com/Support/Licensing/Installation/Pages/default.aspx>

## System Limits

If system limits are set incorrectly, you might encounter an error during the IC Validator run. For example, the following commands set the proper system limits for an IC Validator run on a Linux operating system. (The numerical values are in kilobytes.)

```
unlimit filesize
unlimit datasize
limit descriptors 1024
limit stacksize 8192
```

**Note:**

The descriptors setting controls how many files you can have open at any time during a run.

To verify the limits on a host, run the `limit` shell command.

## Networked Environments

Accessing files over a networked environment, such as when a remote group directory is used, can degrade performance. You might observe large differences in the wall time (overall time) and total CPU time (user time plus system time) reported by the IC Validator tool.

**Note:**

These differences could also be caused by:

- Other processes running on the same system.
- System overhead.

The read- and write-transfer sizes (`rsize` and `wsizes`) of the NFS mounts should be at least 32768. For example, for static mounts set:

```
% mount -t nfs -o rsize=32768,wsizes=32768 server:/path /mnt/nfs
```



# 3

## Output Files

---

*This chapter provides an overview of IC Validator design rule checking (DRC) and layout-versus-schematic (LVS) output files.*

The information described in this chapter is:

- [Description of Output Files](#)
- [General DRC and LVS Output Files](#)
- [LVS-Specific Output Files](#)

---

## Description of Output Files

The output files produced by the IC Validator tool are specific to the type of run performed. If you are performing DRC, the tool produces error and summary files. The error files list the DRC errors in your design. The summary files list information about your design and the completed run.

If you are performing device extraction, the IC Validator tool produces error, summary, and layout netlist files. The layout netlist file is the file that is compared to a schematic netlist during LVS comparison. Both DRC and device extraction runs produce tree structure, virtual cell, and technology files in the `run_details` directory.

If you are performing LVS comparison, the IC Validator tool produces error, log, and equivalence summary files. The error file lists the nets and devices from the layout that do not match the schematic.

The error, summary, and layout netlist files are written to current working directory. There are additional files, which are located in the `run_details` directory, that help in debugging LVS errors.

All output files are named `cell.abbreviation`, where `cell` is the name specified by the `cell` argument in the `library()` function. The extension following `cell` is an abbreviation for the explanation of the file. For example, the account file name is `cell.acct`.

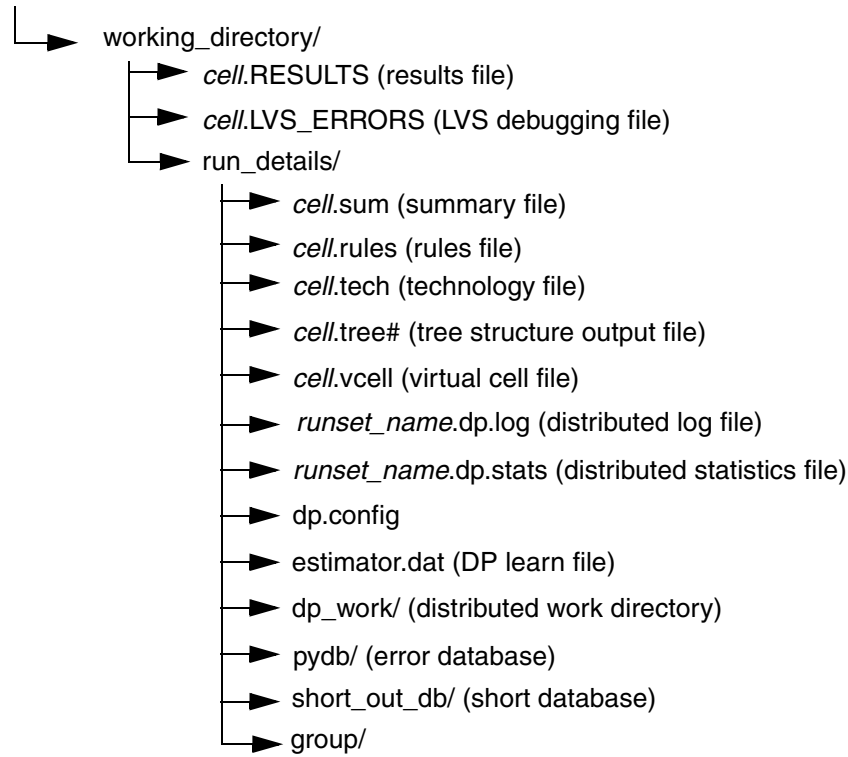
---

## General DRC and LVS Output Files

[Figure 3-1](#) shows the output files and directories common to both DRC and LVS runs. The files are described in the following sections.



Figure 3-1 Output Files and Directories Common to DRC and LVS




---

## Results File

The IC Validator tool creates a RESULTS file, *cell.RESULTS*, that provides the results from key areas of the run. The RESULTS file also contains the IC Validator command line used for the run. This file is the starting point for analyzing the results of an IC Validator run.

The *cell.RESULTS* file is divided into the following sections:

- Results Header

If the DRC run completed with no violations, the IC Validator tool returns a single line that states, "RESULTS: CLEAN".

If the DRC run completed, but with violations, the IC Validator tool returns a single line that states, "RESULTS: NOT CLEAN".

If the LVS run completed, the IC Validator tool returns two results in the header section:

- The first result is a single line that states, "LVS Compare Results: PASS | FAIL".
- The second result is a single line that states, "DRC and Extraction Results: CLEAN | NOT CLEAN".

A DRC or LVS run that does not complete returns a single line that states, "RESULTS: RUN ABORTED".

- Execution

This section consists of two parts:

- The first part prints the IC Validator banner and copyright information, and how the IC Validator tool was called.
- The second part consists of six items: user name, layout format of input library, input library file name and path, top cell name, and the start and end time of the IC Validator run.

- Results Summary: DRC Run

This section consists of two parts:

- The first part reports the rule and DRC error statistics, including the total number of rules that are run, the number of rules that are not executed, the number of rules that contain violations, and the total number of violations. See the `cell.LAYOUT_ERRORS` file for detailed violation data.
- The second part consists of a table showing each rule name and the total number of violations found for that rule. This part is output only if the `verbose_results_file` argument of the `run_options()` function contains the `ERRORS` option. Rule names are extracted from rule comments based on the `rule_name_delimiter` argument of the `error_options()` function. If you run DRC error classification, the number of waived violations is excluded from the number of violations found. Any rule that was not executed shows a "Not Executed" status. Violation counts for individual functions under a rule are not reported, with the exception of functions without a rule. These functions are reported (indented by two spaces) under the default rule name, "Violation." Violation counts are preceded with the "v =" string for easy searching of a specific violation count.

- Results Summary: LVS Run

This section consists of three parts:

- The first part reports the following LVS compare statistics: top equivalence points, number of successful black-box cells, number of failed black-box cells, number of successful equivalences, number of failed equivalences, and number of priority errors. See the `cell.LVS_ERRORS` file for the detailed compare violation data.
- The second part reports LVS device extraction error statistics: total number of rules that were run, number of rules not executed, number of rules with violations, and total number of violations. See the `cell.LAYOUT_ERRORS` file for the detailed LVS device extraction violation data.

- The third part consists of a table showing each rule name and the total number of violations found for that rule. This part is output only if the `verbose_results_file` argument of the `run_options()` function contains the `ERRORS` option. Rule names are extracted from rule comments based on the `rule_name_delimiter` argument of the `error_options()` function. If you run DRC error classification, the number of waived violations is excluded from the number of violations found. Any rule that was not executed shows “Not Executed” status. Violation counts for individual functions under a rule are not reported, with the exception of functions without a rule. These functions are reported (indented by two spaces) under the default rule name, “Violation.” Violation counts are preceded with the “v =” string for easy searching of a specific violation count.
- Assign Layer Statistics
 

This section consists of two parts:

  - The first part reports four items: total number of utilized assign layers, number of empty utilized assign layers, number of non-empty utilized assign layers, and number of pruned assign layers.
  - The second part consists of a table showing each layer name, hierarchical and flat polygon counts for the layer (or “Pruned” if the layer was pruned), and the layer or datatype assignments for the layer. This part is output only if the `verbose_results_file` argument of the `run_options()` function contains the `ASSIGNS` option.

The table is divided into three sections: 1) non-empty utilized layers 2) empty utilized layers 3) pruned layers. The assign statement for each layer contains layer datatype assignments and all other user-supplied assignment options.
- Run Configuration
 

This section consists of four parts:

  - The `run_options()` function reports the `verbose_results_file` argument.
  - The `hierarchy_options()` function reports the `delete` argument.
  - The `error_options()` function reports the following items: `error_limit_per_check`, `match_errors`, `report_empty_violations`, and `clean_classifications` arguments.
  - Distributed processing reports the distributed cores and turbo, followed by a list of host machines with their specifications (model name, CPU, cores, RAM, swap space). Each host machine also displays the number of distributed processes and the number of CPU on the host next to the host name.
- Performance Statistics
 

This section reports five items:

- IC Validator runtime.
- Peak single command memory.
- Peak disk usage.
- Distributed host peak memory usage (this is a table showing each host, the distributed processing and number of CPU on that host, and the host peak memory and RAM).
- Overall distributed utilization.

---

## Error File

The IC Validator tool creates an error file, *cell.LAYOUT\_ERRORS*, during DRC or netlist extraction runs. These types of errors are referred to as layout errors. The *LAYOUT\_ERRORS* file contains the results of a run, a summary list of the errors, and a more detailed list of the errors.

The top of the file states either CLEAN or ERRORS so that you can quickly determine if there are layout errors in the design.

The next section is the summary section. This section provides the comment for a function, the function itself, and the number of errors for that function if it had errors.

The last section is the details section. Each error report contains the following: an error description that identifies the check being performed; a structure name where the error occurred; a position or a bounding box value where the error occurred; and other information specific to a check. The position coordinates are reported hierarchically, that is, relative to the named structure and only one time for each referenced structure. These errors can also be viewed graphically using the VUE debugging tool.

The following section of the *LAYOUT\_ERRORS* file shows a minimum area rule for metal1 (Met1). The rule was coded as follows in the runset:

```
//R.Met1.A
{
  @"R.Met1.A:For Metal 1, minimum area must be 0.04";
  area( Met1, value < 0.04);
};
```

The *LAYOUT\_ERRORS* file reports this rule as follows:

```
LAYOUT ERRORS RESULTS: ERRORS

#####  #####  #####  ###  #####  ###
#      #  #  #  #  #  #  #  #  #
#####  #####  #####  #  #  #####  ###
#      #  #  #  #  #  #  #  #  #
#####  #  #  #  #  #####  #  #  #####
```

```

=====
Library name:      ../../lab_data/INLIB_TRAINING.GDS
Structure name:    top
Generated by:      IC Validator <platform> <release> <date>

Runset name: icv runset.rs
User name:  juser
Time started: 2016/03/02 07:35:12AM
Time ended: 2016/03/02 07:35:51AM

Called as: icv runset.rs
           ERROR SUMMARY

R.Met1.A: Minimum area must be 0.04
  area ..... 1 violation found.

           ERROR DETAILS

-----
R.Met1.A: Minimum area must be 0.04
-----

sf.rs:238:area
-----
Structure ( lower left x, y ) ( upper right x, y) Area
-----
top      (37.2700, 4.0300)   (37.4800, 4.1700)   0.0294

```

---

## Summary File

The IC Validator tool creates a summary file, *cell.sum*, during DRC or netlist extraction runs.

If you are using the `-verbose` command-line option, the information printed to the screen is the same as the information reported to the summary file. The screen output, however, prints the information in the order of processing by the IC Validator tool, while the content of the summary file sorted to match the sequential order of the runset.

---

## Rules File

Rules to be executed are written to the *cell.rules* file in the run details directory. Violation names and violation comments that remain in the runset after all forms of runset selection have been performed are reported in the *cell.rules* file. Examples of runset selection include preprocessor directives, selectable rules, incremental layers, and run-only pruning.

Violation names and violation comments are harvested for all rules in the final parse tree. Because runtime conditional commands (`if/then/else` statements and `for/foreach/while`

loops) might remain, rules in the final parse tree are not guaranteed to execute. Rule output is included for all commands in the final parse tree, including commands that do not execute due to runtime conditions.

Violation names and violation comments might not always be statically resolved. In this case the file and line number are provided in the output file instead of the rule name.

Only the violation comment is present in the output if there is no "@=" violation block for the runset rule.

The default comment is "Violation" and "@ \"Violation\"" appears in the rules to be output when a rule uses the default comment.

The following is an example *cell.rules* file:

```

                                ICV_Master (R)
    Copyright (c) 1996 - 2015 Synopsys, Inc.
    This software and the associated documentation are proprietary to
    Synopsys, Inc. This software may only be used in accordance with the terms
    and conditions a written license agreement with Synopsys, Inc. All other
    use, reproduction, or distribution of this software is strictly
    prohibited.

Called as: icv runset.rs -i test.gds
-----
User name:      synopsoid
Layout format:  GDSII
Input file name: test.gds
Top cell name:  top
Time started:   2015/11/03 02:43:08PM
-----
                                Rules to be Executed
-----
    Unresolved violation name/comment found at runset.rs:100
    @ "Rule comment with no violation block"
    @ "Violation"
    VIOL_101 @= { @ "VIOL_101_comment" }
    VIOL_102 @= { @ "VIOL_102_comment" }
    VIOL_103 @= { @ "VIOL_103_comment" }
    VIOL_104 @= { @ "VIOL_104_comment" }
    VIOL_105 @= { @ "VIOL_105_comment" }
```

---

## Technology File

The IC Validator tool creates a technology file, *cell.tech* file, that displays the options which are set for the IC Validator run. The list of options includes the default as well as the user-specified values. It also lists all the cells that are exploded by these options.

---

## Tree Structure Output File

The IC Validator tool creates tree structure files, named `cell.tree`, during DRC or netlist extraction runs. Tree structure files contain information about the design hierarchy tree and reference statistics for every cell in the design tree. The `cell.tree0` file represents the original data. The `cell.tree1` file is created after all of the exploding options are completed. Additional tree files are created when using the hierarchy options. The `cell.treeN` file with the highest number (*N*) is the hierarchy that was processed during the run. The tree files are printed if the `print_preprocess_files` argument of the `run_options()` function specifies the `TREE` option.

---

## Virtual Cell File

The virtual cell file, `cell.vcell`, contains information about pairing, creating sets, and exploding of virtual cells depending on the options you use in the IC Validator runset and the data in your design. It includes the Pair, Set, and Exclude sections.

### Pair Section

The Pair section reports the cell pairs that were created during each virtual cell pass iteration. The `iterate_max` option of the `pairs` argument of the `hierarchy_options()` function sets the maximum number of iterations.

### Set Section

The Set section reports the sets that were created during each virtual cell pass iteration. The `iterate_max` option of the `sets` argument of the `hierarchy_options()` function sets the maximum number of iterations.

### Explode Section

The Explode section of the output file displays all of the branches that were exploded to bring together interacting placements under a common parent. For more information, see the `explode` argument of the `hierarchy_options()` function in the *IC Validator Reference Manual*.

---

## Distributed Log File

The distributed log file, named with the runset name and a `dp.log` extension, contains more information than a summary file, as it includes details about when a function is sent to a host, and the results produced when the host completed the function. The distributed log file is in order of execution rather than in order of the runset.

---

## Distributed Statistics File

A distributed statistics file, named with the runset name and a `dp.stats` extension, contains distributed processing statistics.

---

## Distributed Work Directory

A distributed work directory, named `dp_work`, is created for a distributed run in the run details directory. The directory contains subdirectories and files used during the distributed run. If this directory exists before a run, it and all of its content are deleted and new files are created. Because this directory is deleted before the run, do not name any individual directories `dp_work` in the run details directory of a run.

---

## Graphical Error Structures

The IC Validator tool generates an error database, PYDB, that can be viewed in a layout editor, such as VUE. It contains error polygons and coordinates that are located in proximity to design rule violations. The shapes produced by different checks can be written to separate layers and data types in the error structure, as specified in the runset file. See [Chapter 8, “DRC Error Classification,”](#) for a description of the error database and of the `pydb_report` and `pydb_export` utilities that you can use to classify errors.

The error database contains all the error data generated by violation blocks and all the information needed for a VUE debugging session. By default, this database is in `run_details/pydb`. You can change the location by using the `db_path` argument of the `error_options()` function. This database allows for data mining, and you can create custom reports.

Note:

You can start reviewing the error database before the IC Validator tool completes the run.

The IC Validator tool generates a hierarchy of error structures that mimics the database hierarchy that is being checked. Thus, any errors that are found in a particular structure in the database appear in a corresponding error structure in the error hierarchy. Error structures are generated only for the database structures that contain errors or that reference other structures with errors. The IC Validator tool does not create an error hierarchy if no errors were found.

Error structures are named with the default error prefix, `ERR_`. This prefix precedes the structure name. For example, the default error structure for the cell `CLOCK` is `ERR_CLOCK`. You can redefine the error prefix using the `error_cell_prefix` argument in the `write` functions, such as the `write_gds()` function.



For example,

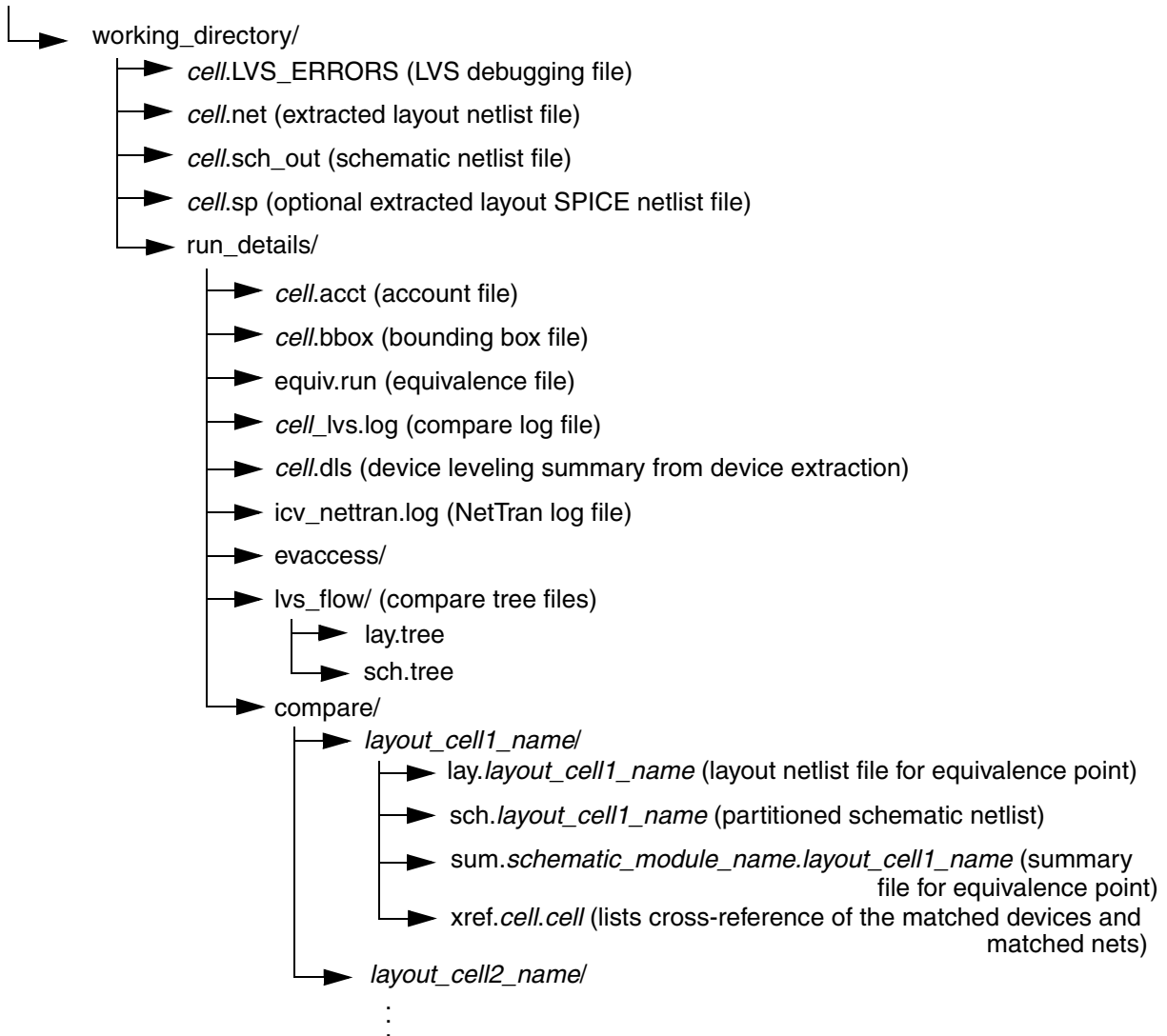
Database hierarchy	Error hierarchy
CHIP {	ERR_CHIP {
ALU {	ERR_ALU {
BIT0	ERR_BIT0
BIT1 }	ERR_BIT1 }
CLOCK {	ERR_CLOCK {
CLK1	ERR_CLK1
CLK2	ERR_CLK3 }
CLK3 }	}
}	

In the error hierarchy, there is no occurrence of the error structure ERR\_CLK2. Either the original CLK2 cells were exploded during preprocessing, or the structure CLK2 did not contain any errors or structures that contain errors. If a previous IC Validator run with errors is corrected and rerun, previous error hierarchy structures are retained. The IC Validator tool does not overwrite these structures if no errors are found, but the new error hierarchy does not have references to these previous error containing cells.

## LVS-Specific Output Files

[Figure 3-2](#) shows the output files and directories for LVS runs. The files are described in the following sections.

Figure 3-2 Output Files and Directories for LVS



## LVS Debugging File

The `compare()` function creates a debugging file, `cell.LVS_ERRORS`, that indicates whether or not the schematic-versus-layout comparison was successful. This file is the starting point for debugging LVS issues.

Depending upon whether the comparison was successful or not, the file lists the PASS/FAIL information and layout errors if any. The `LVS_ERRORS` file then lists the important failed equivalence points, sorted in high and low priorities, and provides diagnostics designed to assist you in troubleshooting the design.

---

## Extracted Layout Netlist File

The IC Validator tool creates an extracted layout netlist file, *cell.net*, whenever a `netlist()` extraction function is specified in the runset file. This file is a hierarchical description of the structures found in the layout database.

---

## Extracted Layout SPICE Netlist File

The IC Validator tool creates an extracted SPICE layout netlist file, *cell.sp*, whenever the `write_spice()` function is specified in the runset file. This file is a hierarchical SPICE description of the structures found in the layout database. See the `write_spice()` function in the *IC Validator Reference Manual* for further information.

---

## Account File

The IC Validator tool creates an account file, *cell.acct*, that lists the number of devices and their types extracted from each cell in the hierarchy. The cumulative count shown is the total count of all devices contained in that structure.

---

## Bounding Box File

The IC Validator tool creates a bounding box file, *cell.bbox*, that lists the total number of cells and the bounding box coordinates of each cell in the design.

---

## Equivalence Files

If you do not supply an equivalence file, then an LVS compare run automatically produces an equivalence file that is used to compare the layout and schematic netlists. When producing an equivalence file, the IC Validator tool recognizes that some equivalence points are extraneous and do not aid in the comparison process. After all analysis is completed, the IC Validator tool generates an equivalence file, *equiv.run*, that lists all the equivalence cells that were used during the run. See the `equiv_options()` function in the *IC Validator Reference Manual* for more information.

---

## Device Leveling Summary File

The device leveling summary (DLS) file gives you a way to trace how a device was extracted in terms of hierarchy level. You can determine the root cause of why a device was extracted at the wrong level of hierarchy (that is, leveled) by the diagnostic information in the *cell.dls* file, which is in the *run\_details* directory.

The following is an example *cell.dls* file.

```
+-----+
|                                     |
|                               ICV Device Leveling Summary                               |
|                                     |
+-----+

Device Leveling Summary Results:

Body Layers          | Cell | Layer Interacted:
+-----+-----+
ngt.polygonlayer.0001  A      nsd.polygonlayer.0001, NW.polygonlayer.0003
pgt.polygonlayer.0001  B      psd.polygonlayer.0001
...
```

The device leveling summary file shows body layers for devices that cannot be pushed back to their original cells after device recognition and property calculations. The device body layers that are successfully pushed back are not shown in this file. The file shows

- **Body Layers.** Derived layer of device body that is leveled out of the cell in which it originally resides.
- **Cell.** Original cell of the leveled devices.
- **Layer Interacted.** Layer that interacted with device body, causing device leveling. If there is an interaction among body layer polygons from the same body layer across multiple levels of hierarchy, the body layer name is printed in this column.

---

## Compare Log File

The IC Validator tool creates a log file, *cell\_lvs.log*, during an LVS run. This compare log file contains all of the messages related to the `compare()` function.

---

## Compare Tree Files

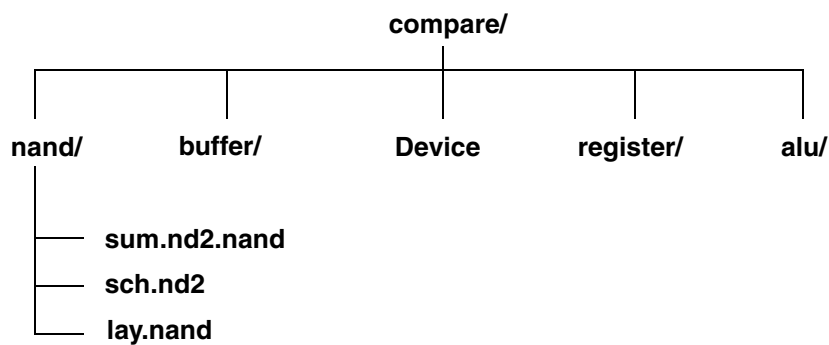
The `compare()` function produces two tree files. One is named *lay.tree*; it provides the hierarchical structure of the layout netlist. The second is named *sch.tree*; it provides the hierarchical structure of the schematic netlist. These files are in the *run\_details/lvsflow* directory.

---

## Compare Directory Output Files

The detailed output files from an LVS comparison are placed in the compare directory as specified by the COMPARE\_PATH section of the ICVread specification file. (See the [icvread\\_spec\\_file\(\)](#) function in the *IC Validator Reference Manual* for further information.) A subdirectory of this compare directory is created for each equivalence point in the equivalence file as the equivalence points are evaluated. These subdirectories are named based on the layout structure name. [Figure 3-3](#) shows an example of a compare directory.

Figure 3-3 Compare Directory Structure Example




---

## Individual Equivalence Summary File

The results of the comparison of each structure are written to a summary file named `sum.schematic_module_name.layout_structure_name`, generally referred to as the `sum.cell.cell` file. The file is located in the compare subdirectory of the working directory, as shown in [Figure 3-3](#). This file is not generated if the `remove_equiv_sum_files` argument of the `compare()` function is `NONE` or if the particular equivalence point is completed without errors or warnings. The information in the summary file includes the following: the number of occurrences of each device found; the total number of nets that were found; and the number of devices and nets that were removed or created by merging. If all of the devices and nets could not be matched, the summary includes a listing of the discrepancies.

The `sum.cell.cell` file is divided into the following sections:

- Comparison Result
- Diagnostic Error
- Unmatched Nodes
- Error Information
- Warning Information

- Preprocessing Information
- Comparison Information
- Merged Node Information
- Cross-Referencing Information
- Statistics Report

---

## Partitioned Schematic Netlist

The schematic data used for this portion of the comparison process is written to the `sch.cell` file. This file is located in the `compare` subdirectory of the working directory, as shown in [Figure 3-3](#). This file is not generated if the `write_equiv_netlists` argument of the `compare()` function is `FAILED` and the particular equivalence point is completed without errors or warnings, or if the argument is `NONE`. This file is the schematic netlist that results after exploding nonequivalent structures; it generally provides little or no useful information for analyzing comparison results.

---

## Partitioned Layout Netlist

The layout data used for this portion of the comparison process is written to a file called `lay.cell`. This file is located in the `compare` subdirectory of the working directory, as shown in [Figure 3-3](#). However, this file is not generated if the `write_equiv_netlists` argument of the `compare()` function is `FAILED` and the particular equivalence point is completed without error or warnings, or if the argument is `NONE`. This file is the layout netlist that results after exploding nonequivalent structures; it generally provides little or no useful information for analyzing comparison results.

---

## Editing Errors

After the error structure has been placed into the database, you can begin editing and correcting the errors. The `LAYOUT_ERRORS` file lists the location and the structure name for each error encountered. The summary file, `cell.sum`, lists the checks that have errors.

Because the output of the IC Validator tool is hierarchical, you need to correct an error only one time, no matter how many times the structure is placed. You continue this process until all errors are eliminated.

# 4

## PXL

---

*This chapter describes PXL, a programmable and extensible language, as well as the basics of function definitions and the use of functions in the IC Validator tool.*

PXL is described in the following sections:

- [PXL Architecture](#)
- [General Definitions](#)
- [Variables and Operators](#)
- [Data Types and Typing](#)
- [Flow Control](#)
- [Functions](#)
- [Program Structure](#)
- [Example of Modularized Code](#)

---

## PXL Architecture

PXL is fully programmable, with general programmability features such as

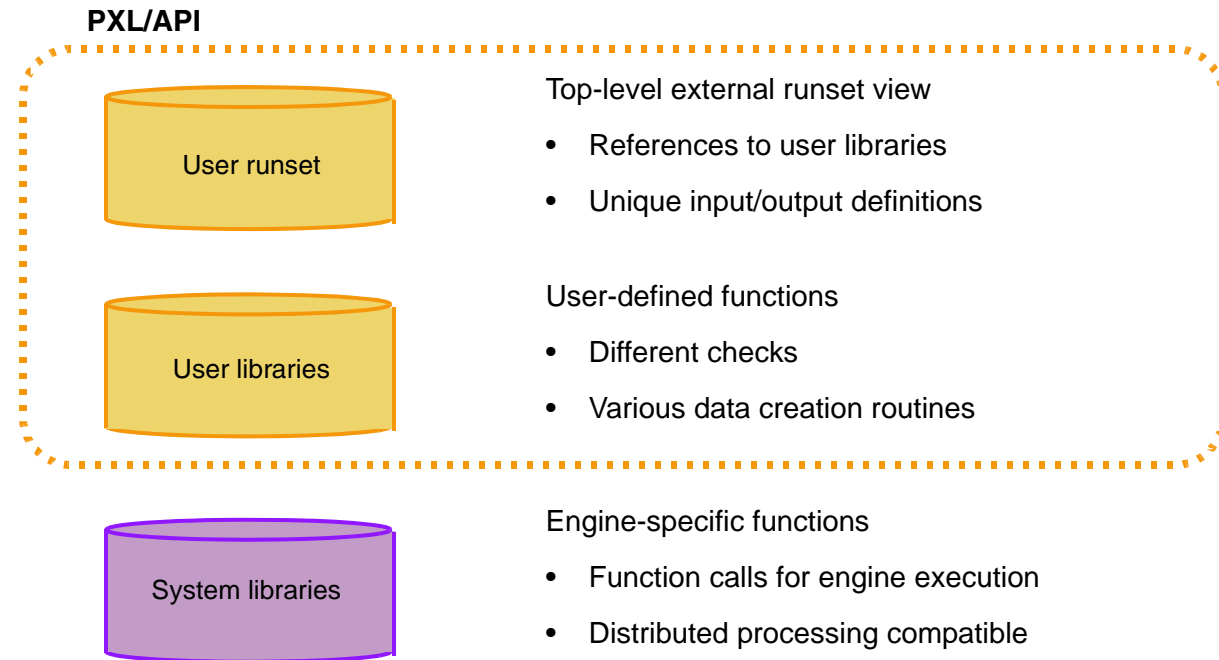
- Variables
- Flow Control
- Lists (arrays) with standard indexing
- Structures
- Hashes
- Enumerated types
- User-defined functions
- Mathematical, logical, and string operations
- Macros with command-line option support
- Environment variables
- Static scope

These features allow for

- Straightforward runset development
- Easy to maintain runsets
- Succinct code
- Code reuse with subroutines
- Support user-defined functions

[Figure 4-1](#) shows the basic structure of PXL.



*Figure 4-1 PXL Structure*

---

## General Definitions

This section provides definitions and characteristics of some syntactic elements in PXL, such as identifiers, string literals, keywords operators, and comments. Understanding the purpose and function of syntactic elements is helpful while creating runsets.

---

### Identifiers

Identifiers for any PXL components can include alphabetic characters, numbers, and underscores in any order.

PXL is case-sensitive. The identifiers listed here are all different:

- abCD
- ABCD
- abcd

The following naming restrictions apply to identifiers:

- Two leading underscores (\_\_). Identifiers with two leading underscores are not allowed. They are reserved for internal language use.

- One leading underscore (\_). Identifiers with one leading underscore are not allowed. They are reserved for product use.
- All uppercase. Reserved for use by the application programming interface. You can, however, use uppercase identifiers as enumerated type elements.
- Reserved keywords. Reserved keywords must not be used as identifiers. For a list of reserved keywords in PXL, see [“Keywords” on page 4-6](#).

You cannot use these names as identifiers:

- Function names: The functions are defined in the *IC Validator Reference Manual* and the IC Validator header files. Example function names are `capacitor`, `check_property`, `internal1`, and `xor`.
- Container type names: These names of lists, hashes, and structures are defined in the IC Validator header files. Here are some examples of container type names:
  - Lists: `coordinate_l` is used in defining the `property()` function
  - Hashes: `layer_list_h` is used in defining the `density()` function
  - Structures: `check_property_s` is used in defining the `check_property()` function

Examples of valid identifiers are

```
This IsAnIdentifier
```

```
another_identifier_using_underscores
```

```
layer20
```

Examples of reserved (invalid) identifiers are

```
__SYSTEMNAME
```

```
(Two leading underscores; reserved for internal language usage)
```

```
_productname
```

```
(One leading underscore; reserved for product use)
```

```
HIERARCHICAL
```

```
(Reserved keyword)
```

---

## String Literals

A string literal is a group of characters beginning and ending with a double quotation mark (").

**Note:**

The first character of a text string cannot be an integer or a space followed by an integer.

String literals must not contain a new line. If a new line is needed in a string literal, the new-line character (\n) must be used.

The following groups of characters are examples of string literals:

```
"hello"
```

```
"\""
```

```
"a string with\nmultiple lines"
```

```
"abc" "def"
```

String literals are automatically merged with any adjacent string literals. In particular, they are merged with preprocessor variables that contain string values, such as `__FILE__`, as well as references to environment variables, such as `$(id)`.

Use escape sequences within strings to set aside parts of the string as instructions to the compiler. The available set of escape sequences in PXL are shown in [Table 4-1](#).

*Table 4-1 Escape Sequences*

Escape	Definition
\n	New-line character
\"	Double quotation mark character
\\	Single backslash character
\NNN	Octal value of the character
\t	Tab character
\xNN	Hexadecimal value of the character

---

## Numeric Constants

Numeric constants in PXL are defined as shown in [Table 4-2](#).

*Table 4-2 Numeric Constants*

Number type	Condition	Example
double	Contains an exponent.	4e5, 10.305e+10
double	Contains a decimal.	3.141592
decimal integer	Contains a leading numeral (1-9).	1023
octal number	Contains a leading zero (0) and any of the digits from zero (0) to seven (7).	0346
hexadecimal number	Contains a leading 0x or 0X, and can contain only the hexadecimal digits (0-9a-fA-F).	0xDEADBEEF

---

## Keywords

Keywords are built-in words that are recognized by the language. Keywords are reserved and therefore cannot be used as identifiers.

A few of the uses of keywords are

- Flow controllers, such as *if*, *foreach*, and *while*
- Operators, such as *in* and *contains*
- Constants, such as *true* and *false*

The following tables list the reserved words in PXL.

Note:

See [“Identifiers” on page 4-3](#) for more information about naming restrictions.

[Table 4-3](#) lists keywords that are reserved by the system and are unavailable for external use.

*Table 4-3 System Keywords*

barrier	broadcast	builtin
deprecated	intrinsic	method

*Table 4-3 System Keywords (Continued)*

obsolete	preprocess	primary
proxy_out	published	required
shutdown	unsafe	unique
volatile		

[Table 4-4](#) lists keywords use by the In-Design flow. Do not use variables with these names in an IC Validator runset.

*Table 4-4 In-Design Flow Keywords*

INCREMENTAL_MILKYWAY_OPTIONS_INCLUDE_VIEW	INCREMENTAL_OPTIONS_SELECT_WINDOW
INDESIGN_DO_FLOATING_VIA_FILL_PURGE	INDESIGN_DONT_UPDATE_TECHNOLOGY_FILE
INDESIGN_ENABLE_MY_ASSIGN	METAL_FILL_BLOCKAGE_LAYERS
METAL_FILL_BLOCKAGE_VIEW	METAL_FILL_DATATYPES
METAL_FILL_DENSITY_DEBUG	METAL_FILL_DENSITY_CHECK
METAL_FILL_DENSITY_DELTA_WIN_SIZE	METAL_FILL_DENSITY_MIN
METAL_FILL_FLATTEN	METAL_FILL_INLIB
METAL_FILL_INLIB_CELL	METAL_FILL_INLIB_EXCLUDE_FILL
METAL_FILL_INLIB_PATH	METAL_FILL_INSTALL_ICV_RH
METAL_FILL_INSTALL_PRIMEYIELD_RH	METAL_FILL_LAYER_NUMBERS
METAL_FILL_OUTLIB	METAL_FILL_OUTLIB_APPLY_PREFIX
METAL_FILL_OUTLIB_CELL	METAL_FILL_OUTLIB_CELL_PREFIX
METAL_FILL_OUTLIB_MODE	METAL_FILL_OUTLIB_PATH
METAL_FILL_OUTLIB_TEMP_CELL	METAL_FILL_OUTLIB_VIEW
METAL_FILL_OUTLIB_RF_CELL_PREFIX	METAL_FILL_RUNSET2LIBRARY_LAYER_MAP
METAL_FILL_SELECT_WINDOW	METAL_FILL_USER_RUNSET

**Table 4-4** *In-Design Flow Keywords (Continued)*

<code>_METHODOLOGY_FUNCTIONS_RS_</code>	<code>SAVE_METAL_FILL_INLIB_EXCLUDE_FILL</code>
<code>_SMF_ICV_RH_</code>	<code>SNPSINDESIGN</code>
<code>VIA_FILL_ALLOWED_ONE_SIDE</code>	<code>VIA_FILL_ASSOCIATED_METAL</code>
<code>VIA_FILL_DATATYPES</code>	<code>VIA_FILL_LAYER_NUMBERS</code>
<code>VIA_FILL_MASK_NAMES</code>	<code>VIA_FILL_MET_ENCLOSURE</code>
<code>VIA_FILL_OUTLIB_RF_CELL_PREFIX</code>	<code>_WRITE_AREFS</code>

[Table 4-5](#) lists reserved application identifiers that cannot be used as identifiers.

**Table 4-5** *Reserved Application Identifiers*

<code>binary</code>	<code>boolean</code>	<code>break</code>
<code>by</code>	<code>const</code>	<code>constraint</code>
<code>constraint_category</code>	<code>CONSTRAINT_EQ</code>	<code>CONSTRAINT_GE</code>
<code>CONSTRAINT_GELE</code>	<code>CONSTRAINT_GELT</code>	<code>CONSTRAINT_GT</code>
<code>CONSTRAINT_GTLE</code>	<code>CONSTRAINT_GTLT</code>	<code>CONSTRAINT_LE</code>
<code>CONSTRAINT_LT</code>	<code>CONSTRAINT_NE</code>	
<code>continue</code>	<code>defined</code>	<code>double</code>
<code>elif</code>	<code>else</code>	<code>elseif</code>
<code>enum</code>	<code>false</code>	
<code>for</code>	<code>foreach</code>	<code>function</code>
<code>hash</code>	<code>if</code>	<code>in</code>
<code>in_out</code>	<code>integer</code>	<code>list</code>
<code>newtype</code>	<code>of</code>	<code>on</code>
<code>out</code>	<code>return</code>	<code>returning</code>
<code>step</code>	<code>string</code>	<code>struct</code>

**Table 4-5** *Reserved Application Identifiers (Continued)*

substrate	SUBSTRATE	thru
to	true	unary
undefined	violation	void
while		

Enumerated type elements cannot be used as identifiers. [Table 4-6](#) lists the enumerated type elements used by the IC Validator tool.

**Table 4-6** *Enumerated Type Elements*

ABORT	ABSOLUTE	ABSTRACT_DESIGN
ACCEPT	ACUTE	ADJUST
ALIEN	ALL	ALL_CELLS
ALL_POLYGONS	ALLOW	ALWAYS
AMBIT_ON_BLOCK	ANALOG_DESIGN	ANALOG_GROUND_NET
ANALOG_POWER_NET	ANALOG_SIGNAL_NET	ANNOTATION_FILE
ANTENNA_CELL	ANTIDISHING_BRIDGE	ANY_DEVICE_NAME
ANY_LEVEL	APPEND	APPEND_NEW_VERSION
AREA_FILL	AREA_FILL_USE	ARC
ARC_PLAN_CELL	ASSIGNS	
ASYMMETRIC	ASYMMETRIC_INTERSECTING	AUTO
AUTO_DETECT	AUTOMATIC	AXIS
BACKSLASH	BALANCED	BASE
BISECTOR	BLACK_BOX_DESIGN	
BLOCK_RADIAL	BLOCK_SQUARE	BLOCKAGE_OBJECT
BODY_EXTENTS	BODY_POLYGONS	

**Table 4-6 Enumerated Type Elements (Continued)**

BOTH	BOUNDARY	BOTTOM
BOTTOM_HEIGHT	BREAK_PATH	BRIEF
BULK	BUS	BV_CELL
CAPACITOR	CASE_INSENSITIVE_NAME	CATEGORY_BLOCKAGE
CDL	CELL_LEVEL	CELL_ROW
CENTER	CENTER_TO_CENTER	CENTERLINE
CHECK	CHIP	CHIP_CONTEXT
CHIP_CELL	CHIPWIDE_AVERAGE	CLASSIFY
CLEAN	CLIP	CLIPPED_DELTA_WINDOW
CLIPPED_CENTER	CLOCK	CLOCK_0_SKEW
CLOCK_AND_SPLIT_CLOCK	CLOCK_BUFFER_CELL	CLOCK_NET
CLOCK_RING	CLOCK_STRIPE	CLOSED_OUTER_BOUNDARY
CLOSEST	CMPSIM	COINCIDENT
COLLECTOR	COLOR_1	COLOR_2
COLOR_LAYER_TYPE	COLOR_LINE_END	COLOR_RING
COLOR_SIDE	COMBINE_COLOR	
COMMENT	CONCAVE_TO_CONCAVE	CONCAVE_TO_EDGE
CONDUCTING_LAYER_TYPE	CONDUCTIVE_HEIGHT	CONNECT
CONFLICT	CONTACT	
CONTACTARRAY	CONTINUE	CONVERT_TO_POLYGON
CONVERT_TO_RECT	CONVEX_TO_CONCAVE	CONVEX_TO_CONVEX
CONVEX_TO_EDGE	COORDINATES_AND_DEVICE_TY PES	CORE_AREA_OBJECT
CORE_OF_BLOCK	CORE_WIRE	CORE_WIRE_USE



**Table 4-6 Enumerated Type Elements (Continued)**

CORNER	CORNER_PAD_CELL	CORNER_PAD_DESIGN
CORNER_TO_CORNER	CORNER_TO_CORNER_OR_EDGE	CORNER_TO_EDGE
COUNT	COVER_CELL	COVER_DESIGN
CREATED_PORTS	CROSS	
CUMULATIVE_TOPOGRAPHY	CUSTOM	CUTTING
DATATYPE_ZERO_ONLY	DEEP_NWELL_NET	DEEP_PWELL_NET
DEFOCUS	DELTA_WINDOW	DENSITY
DENSITY_GRADIENT	DENSITY_NORMAL	
DENSITY_PERIMETER	DENSITY_SMARTSTEP	DENSITY_TOTARGE
DENSITY_UNDER_1	DENSITY_UNDER_2	
DEPTH_OF_FOCUS_COLUMNS	DEPTH_OF_FOCUS_COLUMNS_HIGH	DEPTH_OF_FOCUS_COLUMNS_LOW
DEPTH_OF_FOCUS_ROWS	DEPTH_OF_FOCUS_ROWS_HIGH	DEPTH_OF_FOCUS_ROWS_LOW
DESIGN	DESIGN_VIEW	
DETAIL_ROUTE	DETAIL_ROUTE_USE	DEVICE_NAME
DEVICE_PIN	DIAMOND	DIELECTRIC_HEIGHT
DIFFERENT_NET	DIFFERENT_POLYGON	DIFFERENT_SIZE_RECTANGLES
DIODE_DESIGN	DISCARD	
DONUT	DOT	
DOUBLE	DOUBLE_HEIGHT_STANDARD_CELL	DOUBLE_LIST
DRC	DROP	DUAL
DYNAMIC_SHIELD	EDGE	EDGES
EDGE_TO_CONCAVE	EDGE_TO_CONVEX	EDIF

**Table 4-6 Enumerated Type Elements (Continued)**

EIGHT	ELLIPSE	ELLIPTICAL
ELPC	EMITTER	EMPTY
ENCLOSE	END_CAP_DESIGN	
EQUATE_NETS	ERROR	ERRORS
ERROR_CLASSIFICATION	ERROR_NO_ABORT	EXCLUSIVE
EXPANDED_EDGE	EXPLODE	EXTENDED
EXTENTS	EXTERIOR	EXTERIOR_INTERIOR
EXTRACT	FAIL	
FAILED	FAILED_INSERTION	
FC_BUMP_CELL	FC_DRIVER_CELL	FEEDTHRU_BLOCKAGE
FEMTO	FILL	FILL_BLOCKAGE
FILL_BLOCKAGE_USE	FILL_DESIGN	
FILL_RINGS	FILL_TRACK	
FILL_USE	FILLER_CELL	FILLER_DESIGN
FIRST_PATTERN_LAYER	FIXED	FLIP_CHIP
FLIP_CHIP_DRIVER_DESIGN	FLIP_CHIP_PAD_CELL	
FLIP_CHIP_PAD_DESIGN	FLIP_FLOP_CELL	FLOATING
FOLLOW_PIN	FOLLOW_PIN_USE	FORTY_FIVE
FOUR	FRAM	FRAME_VIEW
FULL	FULL_NAME	FULL_NAME_CASE_INSENSITIVE
FULL_NAME_CASE_SENSITIVE	GALAXY_AND_XO_CELL	GALAXY_CELL
GDSII	GENERIC	GLOBAL
GLOBAL_REPLACE	GLOBAL_ROUTE	GLOBAL_ROUTE_USE

**Table 4-6 Enumerated Type Elements (Continued)**

GROUND	GROUND_NET	GROW
GUIDANCE	HALF_WIDTH	HIERARCHICAL
HIGH	HIGHEST_TEXT	HOLDING_CELL
HORIZONTAL	HORIZONTAL_AXIS	HORIZONTAL_TRACK
HORIZONTAL_TRACK_OBJECT	HORIZONTAL_WIRE_TRACK	
HORIZONTALWIRE	HYBRID	HYBRID_BAIL_EARLY
HYBRID_PATTERN_ONLY	ICV	ICV_LCC_AOI
ICV_LCC_BLOCKING	ICV_LCC_CO	ICV_LCC_CONTOUR_AREA
ICV_LCC_M1	ICV_LCC_M1_DUMMY	ICV_LCC_M1_OPC_DUMMY
ICV_LCC_M2	ICV_LCC_M2_DUMMY	ICV_LCC_M2_OPC_DUMMY
ICV_LCC_M3	ICV_LCC_M3_DUMMY	ICV_LCC_M3_OPC_DUMMY
ICV_LCC_M4	ICV_LCC_M4_DUMMY	ICV_LCC_M4_OPC_DUMMY
ICV_LCC_M5	ICV_LCC_M5_DUMMY	ICV_LCC_M5_OPC_DUMMY
ICV_LCC_M6	ICV_LCC_M6_DUMMY	ICV_LCC_M6_OPC_DUMMY
ICV_LCC_M7	ICV_LCC_M7_DUMMY	ICV_LCC_M7_OPC_DUMMY
ICV_LCC_M8	ICV_LCC_M8_DUMMY	ICV_LCC_M8_OPC_DUMMY
ICV_LCC_M9	ICV_LCC_M9_DUMMY	ICV_LCC_M9_OPC_DUMMY
ICV_LCC_M10	ICV_LCC_M10_DUMMY	ICV_LCC_M10_OPC_DUMMY
ICV_LCC_MAX		
ICV_LCC_OD	ICV_LCC_OD_DUMMY	ICV_LCC_OD_OPC_DUMMY
ICV_LCC_PO	ICV_LCC_PO_DUMMY	ICV_LCC_PO_OPC_DUMMY
ICV_LCC_VIA1	ICV_LCC_VIA2	ICV_LCC_VIA3
ICV_LCC_VIA4	ICV_LCC_VIA5	ICV_LCC_VIA6

**Table 4-6 Enumerated Type Elements (Continued)**

ICV_LCC_VIA7	ICV_LCC_VIA8	ICV_LCC_VIA9
IF_INSERTION_FAILS	IGNORE	IGNORE_CAP_LAYER_TYPE
IMAGE_CELL	IN	INCLUSIVE
INCLUSIVE_PARALLEL	INDIVIDUAL	INDUCTOR
INNER	INPUT_LAYOUT	INSERTION
INSIDE	INSIDE_TO_OUTSIDE	
INSTANCE_NAME	INSTANCES	
INTERACT	INTERACTING	INTERIOR
INTERNAL_DESIGNS	INTERSECTION	
IO_PAD_CELL	KEEP	
KEEP_ALL	KEEP_ENCLOSED	KEEP_INTERACTING
KEEP_ONE	KILO	LATCH_CELL
LAYER	LAYER_INTENT_BASE	LAYER_INTENT_FILL
LAYER_INTENT_INTERCONNECT	LAYER_INTENT_METAL	LAYER_INTENT_VIA
LAYER_MAPS	LAYER1	LAYER2
LAYOUT_UNTEXTED	LCC	LEVEL_SOURCE_DRAIN
LI_BASE	LI_FILL	LI_INTERCONNECT
LI_METAL	LI_VIA	LIB_CELL_DESIGN
LIB_CELL_PIN_CONNECT	LIB_CELL_PIN_CONNECT_USE	LIBRARIES_AND_VIEWS
LINE	LINE_END_DIFF_COLOR	LINE_END_SAME_COLOR
LINE_END_TO_LINE_END	LINE_END_TO_SIDE	LINE_TO_LINE
LINEAR_CONTEXT	LINK	
LOCAL	LOGIC_0_NET	LOGIC_1_NET

**Table 4-6 Enumerated Type Elements (Continued)**

LOOP	LOW	LOWER
LOWER_CELLS	LOWEST	LVS
LVS_NETLIST_FLOW_ICV	LVS_NETLIST_FLOW_SPICE	
LVS_USER_UNIT_METER	LVS_USER_UNIT_MICRON	
M1_ROUTE_GUIDE	M10_ROUTE_GUIDE	M11_ROUTE_GUIDE
M12_ROUTE_GUIDE	M13_ROUTE_GUIDE	M14_ROUTE_GUIDE
M15_ROUTE_GUIDE	M2_ROUTE_GUIDE	M3_ROUTE_GUIDE
M4_ROUTE_GUIDE	M5_ROUTE_GUIDE	M6_ROUTE_GUIDE
M7_ROUTE_GUIDE	M8_ROUTE_GUIDE	M9_ROUTE_GUIDE
MACRO_CELL	MACRO_DESIGN	MACRO_PIN_CONNECT
MACRO_PIN_CONNECT_USE	MAIN	MARKER_LAYER_TYPE
MASK_FIVE	MASK_FOUR	
MASK_ONE	MASK_ONE_HARD	MASK_ONE_SOFT
MASK_THREE	MASK_TWO	MASK_TWO_HARD
MASK_TWO_SOFT	MAX	MAX_AREA
MAX_CONDITION	MAX_CONDITION	MAX_INSERTION
MEEF	MEGA	MERGE
MERGED	MERGE_ALL	MERGE_CONNECTED
MERGE_CONNECTED_AND_TOP	MERGE_TOP_THEN_CONNECTED	MERGE_TOP_PORTS_THEN_CONNECTED
MERGED	METAL_HEIGHT	
METER	MICRO	MICRON
MID_CELL	MILKYWAY	MILLI
MIN	MIN_AREA	MISSING_CELLS

**Table 4-6 Enumerated Type Elements (Continued)**

MISSING_COLOR	MIXED	MODULE_CELL
MODULE_DESIGN	MORE_THAN_TRIPLE_HEIGHT_CELL	MUTUAL
MUTUAL_NON_ORTHOGONAL	NAME_MATCHED	NANO
NDB_ALL	NDB_ANY	NDB_BJT
NDB_CACHE	NDB_CAP	NDB_CELL
NDB_DIODE	NDB_IGNORE	NDB_IND
NDB_INST	NDB_MOS	NDB_NET
NDB_NETLIST	NDB_NMOS	NDB_NONE
NDB_NP	NDB_NPN	NDB_ONE_WAY
NDB_PIN	NDB_PMOS	NDB_PN
NDB_PNP	NDB_RES	NDB_SUBCKT
NDB_UNDEFINED	NDB_TWO_WAY	NDB_UNCACHED
NDM		
NEGATIVE_45	NEIGHBORING_GAP	NEITHER
NET_NAME	NET_PORTPROPERTY	NETLIST_ANNOTATION_FILE_SPICE
NETLIST_ONLY	NETLIST_PEX_SPICE	NETLIST_XTR_SPICE
NETS	NEVER	
NEW_AREF_ONLY	NINETY	
NMOS	NO_COLOR	NO_EXPLODE
NODE	NOMINAL	NOMINAL_VARIATION
NON_CONT_PG	NON_ORTHOGONAL	NON_RECTANGULAR_SHAPES
NONE	NONE_DENSITY	NONE_DENSITY_PERIMETER

**Table 4-6 Enumerated Type Elements (Continued)**

NONE_INCLUSIVE	NORMAL	NOT_A_DOUBLE
NOT_ADJACENT	NOT_CONTAINED	NOT_CORNER
NOT_DEFINED_LAYER_TYPE		
NOT_TEXTED	NOTCH	NP
NPN	NULL_CONNECT_DATABASE	NWELL_NET
NULL_PARAMETRIX_LAYER	OASIS	OCTAGONAL
ODD_LOOP_ONE_LINK	ODD_LOOP_ALL_LINKS	ODD_LOOP_ALL_NODES
ODD_LOOP_ALL_LOOP	ODD_LOOP_INSIDE_RING	OFF
OLD_AREF_ONLY	ON	ONE
ONE_EIGHTY	ONE_OR_NEITHER	ONE_POLYGON_PER_LAYER
ONE_X	ONLY_COLOR_1	ONLY_COLOR_2
ONLY_COORDINATES	OPC	OPC_USE
OPEN_OUTER_BOUNDARY	OPENACCESS	ORTHOGONAL
OTHER_NETS	OUT	OUTPUT_CELL
OUTPUT_ERRORS	OUTSIDE	OVERLAP
OVER_EXPOSURE	OVERSIZE	OVERWRITE
OWNER_NET	OXIDE_HEIGHT	PAD_DESIGN
PAD_FILLER_CELL	PAD_SPACER_DESIGN	PARALLEL
PARALLEL_POINT_PROJECTION	PARTITION	PARTITION_CONTEXT
PATH	PATH_OBJECT	
PATH_SEGMENT	PAUSE	PERCENT
PERIMETER	PERIMETER_DENSITY	PERPENDICULAR
PG_FOLLOW_PIN	PG_PIN	PG_PIN_ONLY_CELL

**Table 4-6 Enumerated Type Elements (Continued)**

PG_RING	PG_STRIPE	PHYSICAL_ONLY_DESIGN
PICO	PIN	PIN_BLOCKAGE
PIN_NAME	PIN_OBJECT	PIN_TEXT
PINS	PLACEMENT_BLOCKAGE	PLACEMENT_HARD_BLOCKAGE
PLACEMENT_HARD_MACRO_BLOCKAGE	PLACEMENT_PARTIAL_BLOCKAGE	
PLACEMENT_SOFT_BLOCKAGE	PM_EXACT	
PM_EDGE_NONUNIFORM	PM_EDGE_UNIFORM	PM_EQ
PM_FUZZY	PM_GE	PM_GT
PM_LE	PM_LT	PM_MARKER_CENTER
PM_MARKER_EDGE	PM_NONE	PM_ONE_DIMENSIONAL
PM_TWO_DIMENSIONAL	PM_TWO_DIMENSIONAL_RUN_LENGTH	PM_UNIFORM
PMOS	PN	PNP
POINT	POINT_TO_POINT	POINT_TOUCH
POLY_ROUTE_GUIDE	POLYGON	POLYGON_OBJECT
POLYGON_TEXT	PORT_NAME	PORT_TEXT_MATCHED
PORT_TOPOLOGY_MATCHED	POSITIVE_45	POWER
POWER_AND_GROUND	POWER_NET	POWER_OR_GROUND
POWER_XOR_GROUND	PRE_COLOR_ALL_NODES	PRE_COLOR_ENVELOPE_PATH
PRE_COLOR_MIN_PATH	PRE_THREE_COLOR_ALL_NODES	PRE_THREE_COLOR_MIN_PATH
PREFIX_NAME	PRIMARY_AXIS	
PRINT	PRINT_ALL	PRINT_RESET_ALL
PROJECTING	PROJECTION	PROPERTY



**Table 4-6 Enumerated Type Elements (Continued)**

PROPERTY_ERRORS_ONLY	PROPERTY_TEXT	PWELL_NET
QCMP	QTF_LAYER_TYPE	RADIAL
RADIAL_INSIDE	RADIAL_INTERSECTION	RADIAL_OUTSIDE
RAM_CELL	REASSIGN	
REASSIGNED_SHORTED	RDL_USE	
RECT	RECTANGLE	RECTANGLE_OBJECT
REDUCTION	REGION	REGION_CONTEXT
REGULAR_TEXT	RELATED_COINCIDENT	RELATIVE
REMAINDER	REMOVE_LAYER_TYPE	
REMOVE_MULTI_EQUIVS	RENAME	
RENAMED	REPLACE	REPORT
RESET	RESET_NET	RESISTOR
RETAIN_45	RIGHT	RING
RING_USE	ROM_CELL	ROTATE_180
ROTATIONAL	ROUND	ROUTE_GUIDE_OBJECT
ROUTE_TYPE	ROUTING_BLOCKAGE	ROUTING_FOR_TOP_LEVEL_BLOCKAGE
RPGROUP_BLOCKAGE	SAME	SAME_COLOR
SAME_DEVICE_NAME_ONLY	SAME_MASK	SAME_NET
SAME_POLYGON	SCAN_NET	SCREEN_BLOCKAGE
SERIES	SERIES_PARALLEL	SHIELD_ROUTE
SHIELD_ROUTE_USE	SHORT_BY_LONG	SHORTED
SHRINK	SIDE_DIFF_COLOR	SIDE_SAME_COLOR
SIDE_TO_SIDE	SIGNAL	SIGNAL_CONTEXT

*Table 4-6 Enumerated Type Elements (Continued)*

SIGNAL_DETAIL	SIGNAL_GLOBAL	SIGNAL_LOCAL
SIGNAL_NET	SIGNAL_USER	SIMPLE
SIMULATE	SINGLE	SINGLE_DEVICE_EQUIVS
SINKS	SIXTEEN	SKIP
SKIP_PARENTS	SLASH	SLEEP_CONTROL
SLOT_BLOCKAGE	SOFT_MACRO_IO_FIXED_CELL	SOG_DESIGN_CELL
SOURCES_ON_SAME_NET	SPECIAL_POWER	SPECIAL_VIA_CELL
SPICE	SPICE_BJT	SPICE_CAP
SPICE_CELL	SPICE_DIODE	SPICE_IND
SPICE_MOS	SPICE_RES	SPLIT_CLOCK
SQUARE	SQUARE_CENTERED	SQUARE_EXTENDED
SQUARE_INSIDE	SQUARE_OUTSIDE	STACK_VIA_CELL
STANDARD_CELL	STANDARD_FILLER_CELL	START
STATISTICAL_COUNT	STOP	STRING
STRIPE	STRIPE_USE	SUBSTRING_NAME
SUM	SYMMETRIC	SYMMETRIC_NON_INTERSECTIN G
TAP_CELL	TECH	TERMINAL
TERMINAL_NAME	TERMINAL_TEXT	TEXT
TEXT_DISPLAY	TEXTED	TIE_HIGH
TIE_LOW	TOP	TOP_CELL
TOP_OF_NET	TOP_PORT	TOUCH
TREE	TRIANGLE	TRIANGLE_ONE_LINK

**Table 4-6 Enumerated Type Elements (Continued)**

TRIANGLE_ALL_LINKS	TRIANGLE_ALL_NODES	TRIPLE_HEIGHT_STANDARD_CELL
TRUNCATE	TSV_CELL	
TWO	TWO_SEVENTY	TWO_X
UF_ADJUSTABLE	UF_CELL	UF_CHECKERED
UF_DPT_SPACING	UF_EXPANDABLE	UF_LAYER
UF_LINE_END	UF_LINE_SIDE	UF_PATTERN
UF_POLYGON	UF_STACK	UF_STRIPE
UNCONNECTED	UNDER_EXPOSURE	
UNDERSIZE	UNSET_NET	UNKNOWN_L_MODEL_CELL
UNMATCHED_PINS	UNMATCHED_TEXT	UNSPECIFIED
UNSPECIFIED_DOUBLE	UNUSED	UNUSED_CELLS
UNUSED_MARKERS	UPDATE	UPPER
USE	USE_ALL_EQUIV	USE_MILKYWAY
USE_OPENACCESS	USE_NDM	USED_CELLS
USER_DEFINED	USER_ROUTE	USER_ROUTE_USE
VCELL	VCMP	VERBOSE
VERILOG	VERTICAL	VERTICAL_AXIS
VERTICAL_TRACK	VERTICAL_TRACK_OBJECT	
VERTICAL_WIRE_TRACK	VERTICALWIRE	VERTICES
VIA_BLOCKAGE	VIA_LAYER_TYPE	VIA_OBJECT
VIEWONLY_LAYER_TYPE	VIEWS	
VIRTUAL_AMBIT	VIRTUAL_CORE	WAIVE
WARN	WARNING	WATCH

*Table 4-6 Enumerated Type Elements (Continued)*

WELL_TAP_DESIGN	WINDOW	WIRING_BLOCKAGE
WITH_PORTS	X_BY_Y	XO_CELL
ZERO_SKEW	ZERO_SKEW_USE	

## Operators

Operators are symbols that have a particular meaning. For example, + indicates that two numbers are added. The operator types are shown in [Table 4-7](#).

*Table 4-7 Operator Types*

Operator type	Number of arguments	Example
unary	1	Logical not (!)
binary	2	Addition (+) Exponentiation (**)
ternary	3	Question operator (e1 ? e2 : e3)

## Comments

Comments are text used to document the program. Comments are useful in indicating the intent of the code to readers. Comments are treated as white space. The comment types, shown in [Table 4-8](#), are based on how many lines each comment contains.

*Table 4-8 Comment Types*

Comment type	Starts with	Ends with	Length
line	//	Next new line	One line
block	/*	*/	Any length

Block comments do not nest. If \*/ is seen within a comment, the preprocessor reports a warning. For example, if the runset contains

```

/*      - 1st comment start
/*      - 2nd comment start
*/      - 2nd comment end
*/      - 1st comment end

```

The "1st comment start" matches with "2nd comment end"; the "1st comment start" does not match with "1st comment end".

Here are some examples of comments:

```

/* this is a single-line block comment */

// this is a line comment, so /* is ignored

/* block comments /* cannot nest. Therefore this comment evokes a
warning from the preprocessor */

/*
 * block comments can span
 * several lines, such
 * as this one does.
 */

/*
  A leading * is not required on each line of a
  block comment but does make the comment stand out
  from the program.
*/

```

---

## Environment Variable Reference

Environment variables are a set of dynamic values that can be used to control the way processes are run.

Environment variable references have the dollar sign character (\$) followed by any normal identifier. If an undefined environment variable is referred to, it is rejected with a compile-time error.

Environment variables are replaced by a string literal containing the value of the environment variable. Replacement is only done on tokens of the form \$<id>. Specifically, there is no replacement within string literals; replacement is not required because adjacent string literals are concatenated.

Note:

If the value of the environment variable is an empty string, it is converted to a string literal whose value is an empty string.

Here are some examples of environment variable usage:

```
x : string = $y;
```

```

/*
 * assuming environment variable my_value contains the value abcd,
 * the following code evaluates to true; otherwise evaluates to false.
 */
res : boolean = ( $my_value == "abcd" );

/*
 * if environment variable aaa contains the value
 * /home/dir/path, the following code produces a string
 * containing a path to the file "cases" in that directory.
 */
path : string = $aaa "/cases";

/*
 * if environment variable bbb is defined but empty, the
 * following code produces the string abcdef.
 */
mystr : string = "abc" $bbb "def";

```

---

## Variables and Operators

This section introduces the concepts of variables and operators in PXL.

---

### Variables

Variables are used to store information temporarily in the program.

When declaring variables, the variable name must be followed by a colon (:) and the variable type. Variables can also be given an initial value when declared. See the examples in the following section.

Note:

See the [Identifiers](#) section for information about naming restrictions.

### Initializing Variables

You can initialize variables to any of the supported data types in two ways:

- Using a literal value.

```
y : boolean = true;           // initialize by literal value
```

- Using an expression.

```
z : boolean = !y;             // initialized by expression
```

The following examples show different methods of initialization:

```
x : integer;                  // no initial value
```

```

pi : double = 3.141592;           // initialized by value

pi2 : double = pi * 2;           // initialized by expression

s : string = "abc" "def";        // concatenated

s1 : string = "abc" + pi;        // converted to string, then concatenated

c : constraint of double = (10.0,200]; // initialized by value

```

## Scope of Variables

Scope in PXL is the meaning that variables and expressions have within several contexts in a program. The various rules and conditions that relate to scope of variables are described here by using examples.

Local scope is defined by a set of braces (`{ }`). That is, compound statements, defined with curly braces, define a new variable scope. See [“Compound Statement” on page 4-59](#) for the use of braces.

Flow control statements create a new scope.

```

if ( var == 1) {                // Start of new scope
    my_var: integer = 1;
    ...
}                               // End of scope.

```

- Variables declared in a local scope do not exist outside of that scope. That is, if a variable is declared inside of a particular scope, the variable cannot be used outside of that scope. For example,

```

{
    x = y + b;
}
z = x;           // x is not defined; it only exists in the local scope

```

- Variables are global from outermost scope to innermost scope. For example,

```

{
    x : integer = 1;
    {
        a = x + b; // x exists and can be seen within this scope
                  // (global from outer scope)
        x = y + b; // global value of x is overwritten
    }
    z = x;         // x is defined here and contains the value of y + b
}
z = x;           // x is not defined here - parser error

```

- Variables declared in an outer scope that are redeclared in an inner scope are shadowed to prevent value collisions.

- If variable shadowing occurs and the scope of the shadowed variable is not defined by a function declaration, the program generates a warning.
- Within function calls, shadowing is expected and there is no warning. For example,

```

y: integer = 1;
x: integer = 1;
{
    y : integer = 3; // The original y value is shadowed.
                    // This is a new y.
    x = x + 1;       // x is equal to 2 here
    y = y + 1;       // y is equal to 4 here
}
                    // y is equal to 1 here - the shadowed value
                    // returns when exiting the inner scope
                    // x is equal to 2 here

```

Here is another example of variable shadowing. The function declaration:

```

fxy : function (
    x : integer;
    y : integer;
) returning z : integer {
    x = x + 1;
    y = y + 1;
    z = x + y;
};

```

The function call:

```

x1 : integer = 1;
y : integer = 1;

z1 = fxy(x1, y); // within the function y is shadowed
                  // y in the local scope is "2"
x2 = x1;          // x1 is "1"
y1 = y;           // y is still "1"
z2 = z1;          // z1 is "4"

```

---

## Operators

Operators are symbols used within an expression to specify certain operations that are to be performed while evaluating that expression.

PXL supports the following operators:

- [Assignment Operators](#)
- [Relational Operators](#)
- [Logical Operators](#)



- [Conditional Operator](#)
- [Numeric and Bitwise Operators](#)
- [Operators with Double-Only Outputs](#)

## Assignment Operators

An assignment operator evaluates an expression and saves it to a target lvalue.

An lvalue is an expression that can contain a value and can be used on the left side of an assignment statement.

Assignments involving the use of > or >= in constraint values must have white space before and after the assignment operator. You can also place the constraint value within parentheses following the assignment =.

For example,

```
abc = >=3;
xyz = (>5);
```

Note:

Assignment operators cannot cascade. This example is not legal:

```
a = b = c
```

The following examples show proper assignment:

```
x: integer; // declare a variable named x

point: newtype struct of { x: integer; y: integer; }; // declare a struct
// of integers

p: point;
x = 42; // assign x a value of 42
p.x = 10; // assign p.x a value of 10
p.y = 27; // assign p.y a value of 27
```

## Relational Operators

A relational expression evaluates an expression to a Boolean value, either true or false.

[Table 4-9](#) defines the relational operators.

*Table 4-9 Relational Operators*

Syntax	Description	Supported for
<code>e1 == e2</code>	true if e1 is equal to e2, false otherwise	All completely defined data types
<code>e1 != e2</code>	true if e1 is not equal to e2, false otherwise	All completely defined data types

*Table 4-9 Relational Operators (Continued)*

Syntax	Description	Supported for
<code>e1 &lt; e2</code>	true if e1 is less than e2, false otherwise	Numbers
<code>e1 &lt;= e2</code>	true if e1 is not greater than e2, false otherwise	Numbers
<code>e1 &gt;= e2</code>	true if e1 is not less than e2, false otherwise	Numbers
<code>e1 &gt; e2</code>	true if e1 is greater than e2, false otherwise	Numbers

## Logical Operators

A logical expression evaluates to a Boolean value. A logical operator requires that all arguments to be of boolean type. Expressions are evaluated from left to right, and stop when the result is defined. [Table 4-10](#) defines the logical operators.

*Table 4-10 Logical Operators*

Syntax	Description
<code>e1 &amp;&amp; e2</code>	if e1 is false, then false, else e2
<code>e1    e2</code>	if e1 is true, then true, else e2
<code>! e1</code>	if e1 is true, then false, else true

## Conditional Operator

A conditional expression takes three arguments; the first argument must be of boolean type. [Table 4-11](#) defines the conditional operator.

*Table 4-11 Conditional Operator*

Syntax	Description
<code>e1 ? e2 : e3</code>	if e1 is true, then e2, else e3

Consider a conditional operator with three arguments e1, e2, and e3. The conditional operator is evaluated in the following order:

1. The first argument (e1) is evaluated.
2. If the first argument (e1) evaluates to true, the second argument is evaluated.

3. If the first argument (e1) evaluates to false, the third argument (e3) is evaluated.

## Numeric and Bitwise Operators

All standard mathematical operators are supported in PXL. The bitwise operators are supported only for integers. [Table 4-12](#) defines the numeric and bitwise operators.

*Table 4-12 Numeric and Bitwise Operators*

Syntax	Description	Allowed for	Comments
e1 + e2	Addition	Integer and double	Returns a double if either argument is a double.
e1 - e2	Subtraction	Integer and double	Returns a double if either argument is a double.
e1 * e2	Multiplication	Integer and double	Returns a double if either argument is a double.
e1 % e2	Integer remainder	Integer and double	Returns a double if either argument is a double.
e1 << e2	Logical left shift (zeros shifted in)	Integer	
e1 >> e2	Logical right shift (zeros shifted in)	Integer	
e1 & e2	Bitwise AND	Integer	
e1   e2	Bitwise OR	Integer	
e1 ^ e2	Bitwise XOR	Integer	
! e1	Bitwise NOT	Integer	

## Operators with Double-Only Outputs

Division (/) and exponentiation (\*\*) treat the arguments as doubles and always return only doubles. [Table 4-13](#) defines these operators.

*Table 4-13 Operators with Double-Only Outputs*

Syntax	Description	Accepts	Comments
<code>e1 / e2</code>	Division	Integer and double	e1 and e2 are forced to be doubles. The operation always returns a double.
<code>e1 ** e2</code>	Exponentiation	Double	e1 and e2 are forced to be doubles. The operation always returns a double.

Note:

Any expression with a division operator (/) or exponentiation operator (\*\*) cannot be used as the index of a list or hash table.

To perform an integer operation, you must wrap the operation in a call of the `dttoi()` function. This function converts a double into an integer. For example,

```
var : integer = dttoi(10/3);
```

## Member Reference

A member reference expression accesses one member of a compound object. The member reference takes the type of that single member. The member is an lvalue and functions as the target of an assignment. [Table 4-14](#) defines member reference expressions. See [“Assignment Operators” on page 4-27.](#) for more information about lvalue.

*Table 4-14 Member Reference*

Syntax	Description
<code>e1 . e2</code>	Structure member access by name
<code>e1 [ e2 ]</code>	List or hash member access by index

## Precedence and Associativity

Table 4-15 lists the precedence and associativity of all operators in PXL. The top of the table is the highest precedence, and the bottom is the lowest.

*Table 4-15 Precedence and Associativity of Operators (Highest Precedence First)*

Operators	Associativity
( ) [ ] .	Left to right
**	Right to left (exponentiation)
! + -	Right to left (unary operators)
== != < <= > >=	Illegal (unary operators)
* / %	Left to right
+ -	Left to right
<< >>	Left to right
< <= > >=	Nonassociative
== !=	Nonassociative
&	Left to right
^	Left to right
	Left to right
&&	Left to right
	Left to right
? :	Right to left
= @= ?=	Illegal
User-defined operators	Nonassociative

[Table 4-16](#) defines each type of associativity.

*Table 4-16 Associativity*

Associativity	Definition
Left to right	Expressions are evaluated from left to right.
Right to left	Expressions are evaluated from right to left.
Nonassociative	Parenthesis are required.
Illegal	The operator does not cascade. Operators do not produce the same type as they consume.

The following examples show how associativity works:

```
/*
 * pairs of expressions show evaluation order
 */
```

```
A + B + C
( A + B ) + C
```

```
A ** B ** C
A ** ( B ** C )
```

```
A * B + C >> 2
( ( A * B ) + C ) >> 2
```

```
A >= B == C
( A >= B ) == C
```

```
A < B == C > D
( A < B ) == ( C > D )
```

## Data Types and Typing

This section describes PXL-supported data types and the factors that determine how typing works within PXL.

Variables can be explicitly declared with a type. For example,

```
i : integer = 1 ; // i is declared as an integer
```

Variables can be implicitly typed based on the return type of the operator. For example,

```
a : double = 1.1203;
b : double = 2.0213;
```

```
c = a + b;           // c is implicitly the type of "double"
```

---

## Supported Data Types

PXL supports most primitive types and some tool-specific data types.

### Primitive Types

[Table 4-17](#) shows the primitive data types supported by PXL.

*Table 4-17 Primitive Types*

Data type	Description
<a href="#">integer</a>	Integer; size is implementation-defined, generally 32-bit signed values.
double	Double-precision floating-point; size is defined at the time of implementation, generally 64-bit values.
boolean	True or false
<a href="#">string</a>	Sequence of characters; supporting escape sequences to include special characters.
<a href="#">handle</a>	An opaque data type, used to track data objects external to the PXL program itself.
<a href="#">constraint of double and constraint of integer</a>	Concise representation of a set of contiguous double values or integer values.

---

#### integer

Here are examples of integer declarations:

```
i : integer;        // declared but not initialized
i : integer = 1;    // declared and initialized
```

#### string

Defines a string of zero or more characters. The string type is initialized from a string literal. For example,

```
s : string = "xyzzzy";
```

The string type supports the operations shown in [Table 4-18](#) on any expression of string type.

**Table 4-18** *String Type Operations*

Operation	e2 type	Return	Modifies e1	Comment
<code>e1.empty()</code>	--	boolean	no	Returns true if the string is empty, that is the string is ""; otherwise returns false.
<code>e1.size()</code>	--	integer	no	Returns the number of characters in the string.

### handle

An internal representation for an externally managed object. The type handle has a string representation that is unique for each externally managed object.

### constraint of double and constraint of integer

Constraint of double and constraint of integer are mathematical ranges for dimensional rule values. These constraints provide an intuitive mechanism for describing parameters such as DRC spacing distances.

If the type of constraint is unspecified, it implies that the range is a constraint of double.

The constraint data types support the operations shown in [Table 4-19](#) on an expression of this type.

**Table 4-19** *Constraint of Double and Constraint of Integer Operations*

Operation	e2 type	Return	Modifies e1	Comment
<code>e1.contains(e2)</code>	item	boolean	no	Returns true if the e2 value is within the range specified by the e1 constraint.
<code>e1.category()</code>	--	enum	no	Returns the enumerator of the constraint from the predefined enumeration constraint category. See <a href="#">Table 4-20</a> .
<code>e1.lo()</code>	--	integer/ double	no	For binary constraints, returns the lower bound of the constraint: a double for a constraint of double; an integer for a constraint of integer. <sup>1</sup>



Table 4-19 Constraint of Double and Constraint of Integer Operations (Continued)

Operation	e2 type	Return	Modifies e1	Comment
<code>e1.hi()</code>	--	integer/ double	no	For binary constraints, returns the upper bound of the constraint: a double for a constraint of double; an integer for a constraint of integer. <sup>1</sup>

1. For unary constraints, such as `==` and `>`, both `e1.lo()` and `e1.hi()` return the single limit.

The constraint data type can only be created by assignment from another like-typed constraint or from a literal constant, as shown in [Table 4-20](#).

Table 4-20 Constraint Categories

Literal	Description	Constraint category
<code>[a,b]</code>	Closed range from a to b, including both endpoints	CONSTRAINT_GELE
<code>(a,b)</code>	Open range from a to b, omitting the endpoints	CONSTRAINT_GTLT
<code>[a,b)</code>	Half-open range from a to b, including a but not b	CONSTRAINT_GELT
<code>(a,b]</code>	Half-open range from a to b, omitting a but including b	CONSTRAINT_GTLE
<code>&gt;= a</code>	Range of all numbers greater than or equal to a	CONSTRAINT_GE
<code>&gt; a</code>	Range of all numbers greater than a	CONSTRAINT_GT
<code>&lt;= a</code>	Range of all numbers less than or equal to a	CONSTRAINT_LE
<code>&lt; a</code>	Range of all numbers less than a	CONSTRAINT_LT
<code>== a</code>	Range containing only the number a	CONSTRAINT_EQ
<code>!= a</code>	Range containing all numbers except a	CONSTRAINT_NE

Both `lo()` and `hi()` are valid for all constraints. For single-ended constraints, both functions are defined to return the single endpoint, 'a'. For double-ended constraints, `lo()` returns 'a' and `hi()` returns 'b'.

## Examples

Here are some examples.

- Constraint of integer

`[2,8]` contains the values 2, 3, 4, 5, 6, 7, 8

`(2,8)` contains the values 3, 4, 5, 6, 7

- Constraint of double

`[2,8]` contains all real numbers in the range including 2.0 and 8.0

`(2,8)` contains all the real numbers in the range except 2.0 and 8.0

- Declaration example

```
a_value : constraint of double = [2.0,8.0];
```

- Assignment examples

```
distance <= 2.0           // All values less than or equal to 2.0
distance = <= 2.0        // All values less than or equal to 2.0
distance = [ 2.0, 8.0 )   // All values >= 2.0 and < 8.0
count = 5                 // Count is exactly equal to 5
count = ==5               // Count is exactly equal to 5
```

**Note:**

The symbols such as `==` and `<=` are part of the constraint. The assignment operator `=` is optional for variables expecting a constraint type. However, if a variable of the constraint type is on the right side, the assignment operator is necessary:

```
my_constraint: constraint of double = < 5.0;
external1(layer1, distance = my_constraint );
external1(layer1, distance <5);           // same as above
external1(layer1, distance = < 5);       // same as above
```

## User-Defined Structures and Types

In addition to the primitive types, PXL supports the structures and types shown in [Table 4-21](#).

*Table 4-21 User-Defined Types*

User-defined type	Salient features
<a href="#">newtype...</a>	Defines a type.
<a href="#">enum of...</a>	Defines a list of enumerated elements.
<a href="#">list of...</a>	Defines a list of elements.
<a href="#">hash of...</a>	Converts a list of elements of one data type to another data type.
<a href="#">struct of...</a>	Defines a composite type consisting of multiple variables.

Table 4-21 User-Defined Types (Continued)

User-defined type	Salient features
<code>function</code>	Defines a user-defined function that can be passed to another user-defined function and can be called as if it were an ordinary variable.

**newtype...**

A newtype declares a type of new variable. It does not define a variable. Each type is unique.

The following example is of a valid type definition:

```
/*
 * define a new type, direction.
 */

direction: newtype enum of {
    up, down, left, right, all // trailing comma is allowed
};

/* define two distinct types */
T1: newtype integer;
T2: newtype integer;

/* define objects of the two types */
o1: T1;
o2: T2;
```

The following example is of an invalid type definition:

```
/* given the definitions above, o1 and o2 are of
 * different types, therefore they cannot be compared
 * and this is an error
 */

if (o1 == o2) { ... }
```

Here is an example of defining and using a new data type called `direction`. It has four allowed values.

```
direction : newtype enum of {
    NORTH,
    SOUTH,
    EAST,
    WEST
};

compass : direction; // Declare compass as type "direction"
compass = EAST;      // Assign a legal value to "compass"
```

**enum of...**

Defines a set of names (enum literals) distinct from other types. The PXL coding standard is for enum literals to be all uppercase characters.

For example,

```
t3: newtype enum of {
    X,
    Y,
    Z,                // Trailing comma is not needed; silently discarded.
};
```

Here is another example:

```
direction : newtype enum of {
    NORTH, SOUTH, EAST, WEST
};

compass : direction;           // Declare compass as type "direction"
compass = EAST;                // Assign a legal value to "compass"
```

An enum can share names with other enumeration types without error:

```
t1_e : newtype enum of {
    NONE, TOP, BOTTOM, ALL
};
t2_e : newtype enum of {
    NONE, LEFT, RIGHT, ALL
};
```

Cross-type operations are not allowed. Also, within one type definition, you cannot use the same name more than one time.

This example shows a cross-type operation. This operation is not legal:

```
v1 : t1_e;    // NONE, TOP, BOTTOM, ALL
v1 = NONE;    // Legal assignment
v1 = TOP;     // Legal assignment
v1 = RIGHT;   // Illegal assignment

v2 : t2_e;    // NONE, LEFT, RIGHT, ALL
v2 = NONE;    // Legal assignment

if (v2 == v1) // Illegal
```

**list of...**

Lists are arrays of objects.

- All of the objects are of the same type. This type is allowed to be empty or to contain duplicate objects.
- It can be processed using the `foreach()` looping construct.

- Lists can be addressed by an index number using brackets ([ ]).

See [“Nested Containers” on page 4-47](#) for information about a list of lists.

This type supports the operations shown in [Table 4-22](#).

*Table 4-22 List Operations*

Operation	e2 type	Return	Modifies e1	Comment
<code>e1.empty()</code>	--	boolean	no	Returns true if the list contains no elements; otherwise returns false.
<code>e1.size()</code>	--	integer	no	Returns the number of elements in the list.
<code>e1.contains(e2)</code>	item	boolean	no	Returns true if e2 is an element of the e1 list.
<code>e1.front()</code>	--	item	no	Returns a copy of the first element of the list.
<code>e1.back()</code>	--	item	no	Returns a copy of the last element of the list.
<code>e1.chop_back()</code>	--	list	no	Returns a copy of the e1 list with the last element removed.
<code>e1.chop_front()</code>	--	list	no	Returns a copy of the e1 list with the first element removed.
<code>e1.append(e2)</code>	item	list	no	Returns a copy of the e1 list with e2 added to the back of the list.
<code>e1.prepend(e2)</code>	item	list	no	Returns a copy of the e1 list with e2 added to the front of the list.
<code>e1.push_front(e2)</code>	item	void	yes	Adds a copy of the e2 expression to the front of the list.
<code>e1.push_back(e2)</code>	item	void	yes	Adds a copy of the e2 expression to the back of the list.
<code>e1.pop_front()</code>	--	void	yes	Removes the first element of the list.
<code>e1.pop_back()</code>	--	void	yes	Removes the last element of the list.

*Table 4-22 List Operations (Continued)*

Operation	e2 type	Return	Modifies e1	Comment
<code>e1.merge(e2)</code>	list	list	no	Returns a copy of the e1 list with the e2 list concatenated to the end.
<code>e1.remove(e2)</code>	item	void	yes	Removes all values in the e1 list that are equal to the e2 expression.

The following are examples of legal code:

```
x: layer = ... ;

MyPartition: list of layer = PartitionLayer(L1);

foreach (part in MyPartition) { ... }

if (MyPartition.contains(x)) { ... }

i : integer = MyPartition.size();
```

The following example shows how to concatenate a list:

```
list1 : list of integer = { 1, 2, 3 };
list2 : list of integer = { 10, 20, 30 };
list3 : list of integer = list1.merge(list2);
```

list3 now has the value { 1, 2, 3, 10, 20, 30 };

Here is an example of initializing and using a list of strings.

```
// declare myList to be a list of strings, uninitialized
myList : list of string;

// declare myList to be a list of strings, initialized
myList : list of string = { "this", "is" , "a", "list" };
// myList[0] = "this"
// myList[3] = "list"
```

Built-in list operator functions are supported. For example,

```
myList : list of string = { "this", "is" , "a", "list" };

s = myList.size();           // returns the size of a list: s is "4"
a = myList.front();          // a contains a copy of L1[0]: a is "this"
myList.pop_front();          // removes the first element of the list,
                             // the list is now { "is", "a", "list" }
myList.pop_back();           // removes the last element of the list,
                             // the list is now { "is", "a" }
myList.push_front("xyz");    // adds a new element containing "xyz"
                             // to the front of the list.
```

```

myList.push_back("bar");    // the list is now { "xyz", "is", "a" }
                           // adds a new element containing "bar"
                           // to the back of the list.
                           // the list is now { "xyz", "is", "a", "bar" }

```

### hash of...

Defines a mapping from one data type to another. The source type is called the key type; the destination type is called the value type. That is, the hash defines an array of key/value pairs.

This type requires that the key type be string, integer, or enum, or a type derived from one of these three types. It cannot be a double. The value type can be of any previously defined type.

The following example creates a hash that converts integers to strings. In this case, the key type is integer and the value type is string. This type is initialized using the `=>` operator.

```

t1_h : newtype hash of integer to string;
t2_h : newtype hash of string to integer;

h1 : t1_h = {
  1  => "a",
  2  => "b",
  26 => "z"
};

h2 : t2_h = {
  "xyzzy" => 1,
  "plugh"  => 10,
  "help"   => 100
};

```

Here is an example of how a derived type can be used:

```

t1 : newtype integer;
h1_h : newtype hash of t1 to string;
x : h1_h = { 1 => "a" } ;

```

The `hash of...` type can be assigned element-wise using subscripts:

```

x = h1[203];
h1[30] = "ad";

```

The `hash of...` type supports the member operations shown in [Table 4-23](#).

**Table 4-23 Hash Operations**

Operation	e2 type	Return	Modifies e1	Comment
<code>e1.contains_key(e2)</code>	key type of hash		no	Returns true if e2 is a key to the e1 hash table. Effectively the same as <code>e1.keys().contains(e2)</code> but with possibly better runtime performance.
<code>e1.contains_value(e2)</code>	value type of hash		no	Returns true if e2 is a value for some entry in the e1 hash table. Effectively the same as <code>e1.values().contains(e2)</code> but with possibly better runtime performance.
<code>e1.empty()</code>	--	boolean	no	Returns true if the hash contains no elements; otherwise returns false.
<code>e1.keys()</code>	--	list of key type	no	Returns a list of all key values; the list does not have duplicate values.
<code>e1.size()</code>	--	integer	no	Returns the number of elements in the hash.
<code>e1.values()</code>	--	list of value type	no	Returns a list of all values; the list can contain duplicates, based on the content of the hash.
<code>e1.merge(e2)</code>	hash matching e1 type	hash	no	Returns a copy of the e1 hash with the keys and values of e2 concatenated to the end; any key that exists in e1 and e2 takes its value from e2.
<code>e1.remove(e2)</code>	item	void	yes	Removes the e2 key from the e1 hash. There is no effect if the e2 key does not exist.



Here is another example:

```
alpha_h : newtype hash of string to integer;
alpha : alpha_h = {           // declare and initialize key => value pairs
    "a" => 1,
    "b" => 2,
    "c" => 3
};
y = alpha["b"];               // The value of y is "2"
keyList = alpha.keys();       // A list of the keys in alpha
```

### **struct of...**

Defines a distinct type identified by the user-supplied name. It must have at least one member and member names must be unique. The members can be of different types.

Here is an example structure that defines a point by its geometric coordinates:

```
point: newtype struct of { x: integer; y: integer; };
```

The `struct of...` type can contain initial values that are used as defaults for structure creation.

```
t3_s : newtype struct of {
    x: integer;
    y: integer = 76;    // initializer
    z: string = "hi";   // initializer
};
t4_s : newtype struct of {
    a: integer = 106;   // initializer
    b: string = "boo"; // initializer
};
```

The `struct of...` type can be assigned from a C-style initializer or from a list of name-value pairs. The following example shows assignment using the types defined in the previous example.

```
s3 : t3_s = { 10, z = "bye" }; // note that 'y' defaults
                                // to the value 76
```

```
// the defaults cause s4 to have a=106, b="boo"
s4 : t4_s = { };
```

Here is another example.

```
shoebox : newtype struct of {
    style : string = "Tennis Shoe";
    shoe_size : double = 10.5;
};

box : shoebox;
shoe = box.style;           // shoe is "Tennis shoe", the default
box.style = "Dress Shoe";   // Reassign the member box.style
```

```
// create a new shoebox with a different kind of shoe
box2 : shoebox = {"Boat Shoe", 9};
```

### function

Allows a function to be passed to another user function. The function that is passed can be used as though it were a global function. For example,

```
// input file
#include <icv.rh>
xyzyy : function (void) returning void { note("abcd"); }
plugh : function ( f : function(void) returning void ) returning void {
f(); }
plugh(xyzyy); // summary file shows "abcd" in output when plugh runs
```

### Nested Types

Types can be combined to form nested types, such as

- list of struct
- list of enum

For example, using the shoebox struct definition in the example in [“struct of...” on page 4-43](#):

```
box : list of shoebox = {
    { "Walking", 12 },
    { "High Heel", 7 }
};

A = box[0].style;           // A is "Walking"
B = box[1].shoe_size;      // B is 7
```

---

## Type Analysis

This section describes how PXL handles data types in special cases, such as when one data type is passed in an argument but another data type is expected.

In cases where PXL encounters one data type when another data type is expected, PXL performs automatic type conversion whenever appropriate. The rules that are used to perform automatic type conversion are discussed in the following sections.

## Standard Arithmetic Conversions

When there is a mismatch between the arithmetic type demanded by an expression and the arithmetic type found, conversions are done as shown in [Table 4-24](#).

*Table 4-24 Arithmetic Conversions*

While expecting ...	Allow integer	Allow constraint of integer	Allow double	Allow constraint of double
integer	Yes <sup>1</sup>	No <sup>2</sup>	No	No
constraint of integer	Yes	Yes	No	No
double	Yes	No	Yes	No
constraint of double	Yes	Yes	Yes	Yes

1. Yes indicates that the conversion is allowed.

2. No indicates that the conversion is not allowed. The operation resulted in an error.

These conversions are only applied when required by the types of the expression. In particular, integer-typed expressions are not converted to double unless required by the context.

When converting an integer or double to a constraint, the resultant constraint appears as if it had been written using the == unary constraint (CONSTRAINT\_EQ).

```
f: function ( c: constraint ) returning void { ... }

f( (1.0, 26.0) );           // constraint is ok
f( 2.0 );                  // single value is ok also
f( 42 );                   // single integer is accepted
```

## Comparing Doubles

Because the type double is an approximation of the real numbers, there are occasionally accumulated errors that cause apparently equivalent operations to produce different results.

Note:

A warning is reported whenever there is a direct comparison (either == or !=) involving one of the double-related types.

## Type Promotion

When a list is expected and an expression of the same type as that contained by the list is found, the expression is promoted from a single item to a list containing that item.

## Parameter Passing

Function parameters that are qualified with either output or input-output must have matching argument values. The rules previously described do not apply. For example, when a function expects a constraint argument that is marked as out, providing the name of an integer-typed variable is not legal because the conversion that is required is not legal.

## String Concatenation

Any expression of the types shown in [Table 4-25](#) can be appended to a string expression using the standard addition operator (+).

*Table 4-25 String Concatenation*

Type	Resultant string appended
integer	Value of the integer expression
double	Value of the double expression
handle	Textual representation of the handle; unique for each distinct object
string	Content of the string
boolean	Either true or false
constraint of integer	Equivalent constraint literal expression
constraint of double	Equivalent constraint literal expression
user-defined enum	Name of the enum literal represented by the expression

When you are working with strings, remember

- When the left side of an addition operator (+) is a string, the result is a string.
- The addition operator (+) is not symmetric for string-valued expressions:  $a + b \neq b + a$
- There are no spaces added between subsequent items appended to a string, so you might see unexpected results, such as in the following examples:

```
s : string;
s = "hello " + 10 + " " + (1 == 0);
// s now contains the string "hello 10 false"

c : constraint = ( 10, 40 ];
s = "" + c;
// s now contains the string "(10.,40.]", because the
```

```
// type of "c" is constraint of double

mye : newtype enum of { A, B, C, D, X };
t : mye = X;
s = "" + t + A + B;
// s now contains the string "XAB"

s = "" + 11 + 3.14 + (1 == 1);
// s now contains the string "113.14true"

s = "" + (11 + 3.14) + (1 == 1);
// s now contains the string "14.14true"
```

## Default Injection

When processing a function call, parameters for which no value is given assume the defaults, if any, from the function prototype. A parameter with no default or specified value results in an error.

Similarly, when processing a literal structure, members for which no value is given assume the defaults, if any, from the newtype structure definition. A member with no default or specified value results in an error.

---

## Nested Containers

Lists, hashes, and structures are also known as containers. Containers can be nested in many different combinations. An example of a nested container is a list of structures.

Here is an example of a hash of a hash. The runset code is

```
hash1_h : newtype hash of string to integer;
hash2_h : newtype hash of string to hash1_h;

multi_hash : hash2_h = {};

multi_hash["A"] = {};           // Initialize hash of "A" to empty
multi_hash["A"]["ONE"] = 1;
multi_hash["A"]["TWO"] = 2;
multi_hash["B"] = {
  "THREE" => 3,
  "FOUR" => 4
};

foreach(key1 in multi_hash.keys() ){
  foreach(key2 in multi_hash[key1].keys()){
    note( "multi_hash[" + key1 + "][" + key2 + "] = "
      + multi_hash[key1][key2] + "\n");
  }
}
```

The output is

```
runset.rs:123 multi_hash[A][ONE] = 1
runset.rs:123 multi_hash[A][TWO] = 2
runset.rs:123 multi_hash[B][FOUR] = 4
runset.rs:123 multi_hash[B][THREE] = 3
```

Here is an example of a list of a list. The runset code is

```
list1_l : newtype list of integer;
list2_l : newtype list of list1_l;

list2 : list2_l = {
  { 0, 1, 2, 3 },
  { 10, 11, 12, 13 }
};

list2.push_back({ 20, 21, 22, 23 });
list2[1].push_back( 14 );

for( j=0 to (list2.size() - 1) ){
  for( i=0 to (list2[j].size() - 1) ){
    note( "list2[" + j + "][" + i + "] = " + list2[j][i] + "\n");
  }
}
```

The output is

```
runset.rs:143 list2[0][0] = 0
runset.rs:143 list2[0][1] = 1
runset.rs:143 list2[0][2] = 2
runset.rs:143 list2[0][3] = 3
runset.rs:143 list2[1][0] = 10
runset.rs:143 list2[1][1] = 11
runset.rs:143 list2[1][2] = 12
runset.rs:143 list2[1][3] = 13
runset.rs:143 list2[1][4] = 14
runset.rs:143 list2[2][0] = 20
runset.rs:143 list2[2][1] = 21
runset.rs:143 list2[2][2] = 22
runset.rs:143 list2[2][3] = 23
```

Here is another example of a list of a list. The runset code is

```
list3 : list of list of integer = {};

list3.push_back( { 0, 1, 2, 3 } );
list3.push_back( { 4, 5, 6 } );
list3.push_back( { 7, 8 } );
list3.push_back( { 9 } );

for( j=0 to (list3.size() - 1) ){
  for( i=0 to (list3[j].size() - 1) ){
    note( "list3[" + j + "][" + i + "] = " + list3[j][i] + "\n");
  }
}
```

```
}  
}
```

The output is

```
runset.rs:159 list3[0][0] = 0  
runset.rs:159 list3[0][1] = 1  
runset.rs:159 list3[0][2] = 2  
runset.rs:159 list3[0][3] = 3  
runset.rs:159 list3[1][0] = 4  
runset.rs:159 list3[1][1] = 5  
runset.rs:159 list3[1][2] = 6  
runset.rs:159 list3[2][0] = 7  
runset.rs:159 list3[2][1] = 8  
runset.rs:159 list3[3][0] = 9
```

---

## Flow Control

Flow can be controlled using

- [Conditional Statements](#)
- [Loops](#)

---

### Conditional Statements

Conditional statements in PXL are constructs that allow you to perform different actions depending on whether a specified condition evaluates to true or false.

The `if` statement is used in a program to perform an action only when a predefined condition is fulfilled.

The `if` statement is used in conjunction with at least two other components:

- An expression that can be evaluated to a Boolean value, true or false.
- A component that is executed if the evaluated expression returns a true value.

Optionally, along with the required components, you can build a more complex conditional statement by including the following components:

- An `else` construct followed by a component that is executed if the first expression is evaluated to false.
- An `elseif` construct that indicates that a looped `if` condition or one or more `if` conditions are nested within a parent if conditional statement. PXL also supports the `elif` construct.
- The `if` statement uses the following syntax:

```

if (e1) s1;
if (e1) s1 elif (e2) s2;
if (e1) s1 elseif (e2) s2;
if (e1) s1 else s2;

```

The `if` statement is evaluated, as follows:

- The first Boolean expression (`e1`) is always evaluated first.
- The first component (`s1`) is processed if `e1` evaluates to true.
- If the `else` statement is encountered, the statement that follows (`s2`) is executed.
- If `elif` or `elseif` statements are encountered, the statements following those are executed.

---

## Loops

As in other programming languages, loops can be used in PXL to repeat an action a number of times.

PXL provides the following methods to create looping constructs in programs:

- [for Loop](#)
- [foreach Loop](#)
- [while Loop](#)

### for Loop

The `for` loop is used to repeat a particular action a specified number of times. It consists of at least one integer (`e1`) and an action (`s1`) that must be repeated. It could also contain additional integers (`e2`, `e3`).

It uses the syntax

```

for (var = e1 to e2) s1;           //s1 is a statement; it can be compound
for (var = e1 to e2 step e3) s1;

```

- Requires `e1`, `e2`, and `e3` to be integer-valued expressions.
- Defines `var` as an integer variable visible only within statement `s1`.
- Evaluates `e1`, `e2`, and `e3` one time at the beginning of the loop; subsequent assignments that affect the values of these expressions have no effect on looping.

For example,

```

for(i=0 to 10 step 2){           // Increment by 2: 0, 2, 4, 6, . . .

```



```

    if( i < 6) {
        a[i] = b[i] + c[i];
    } elif( ( i >=6 ) && ( i < 10)){
        a[i] = b[i] + c[i] + 1;
    } else {
        a[i] = b[i] - c[i];
    }
};

```

## foreach Loop

The `foreach` loop is similar to a `for` loop but it is used to perform a set of actions with reference to the elements of a container, such as a list or an enum. It does not use a variable counter to determine the number of iterations of the action.

It consists of a list of type (e1) and a component (s1) that is evaluated for each element in e1.

It uses the syntax

```
foreach (var in e1) s1; //e1 is of the type "list of"
```

- s1 is evaluated one time for each element of e1, with var copied from the list.
- s1 is bypassed if the input list e1 is empty.
- var is a constant variable of the correct type visible only within statement s1.

## while Loop

The `while` loop is used to repeat a particular action until a condition is met. It consists of a Boolean expression (e1) and an action (s1) that must be repeated. The while loop always evaluates the Boolean expression e1 at least one time.

It uses the syntax

```
while (e1) s1; // e1 is evaluated as a Boolean;
              // loop continues until e1 is false
```

The `while` statement is evaluated, as follows:

- Each time e1 evaluates to true, s1 is executed.
- When e1 evaluates to false, the statement ends.

---

## Functions

Functions in PXL are categorized based on their usage. The function types are described in [Table 4-26](#).

*Table 4-26 Function Types*

Function type	Description
Runset function	Standard IC Validator published function.
Remote function	Function written by the user that can be called by runset functions. There are also default remote functions supplied by the IC Validator tool.
User function	Function that is written by a user.
Utility function	Function supplied by the IC Validator tool that provides access to primitive data in a remote function.

---

## Function Definitions

A function definition defines a set of operations associated with a name. A function definition identifies the names, types, and defaults for all parameters and for a returning value.

You can create your own functions and use them along with the provided functions within your runsets.

As is shown in the following example, all function definitions must be at the top level.

```
// a simple function definition

myinversion: function ( arg: double ) returning rv : double
{
    rv = 1.0 / arg;
}

x = myinversion(2.0); // simple function call
```

Function definitions provide a separate scope for all processing. Scoping is determined in the context of where things are written (static scoping). For information about scope see [“Scope of Variables” on page 4-25](#).

Inside of a function definition these scoping rules apply:

1. If a declaration cannot be found at the current scope, higher scopes are searched until the entire function definition has been searched, stopping at a match.

2. Then, if necessary, a global variable that is declared before the function definition and has the same name is used.
3. Lastly, if the variable is being assigned to a function call or is in the position of an output parameter to a function call, it is implicitly declared in the scope of the current statement. However, if the variable is in the position of an input or input-output parameter to a function call even if it is being assigned to a function call, it is not implicitly declared in the scope of the current statement.

Outside of a function definition these scoping rules apply:

1. If a declaration cannot be found at the current scope, higher scopes are searched until the entire runset has been searched, stopping at a match.
2. Then, if necessary, a global variable that is previously declared and has the same name is used.
3. Lastly, if the variable is being assigned to a function call or is in the position of an output parameter to a function call, it is implicitly declared in the scope of the current statement. However, if the variable is in the position of an input or input-output parameter to a function call even if it is being assigned to a function call, it is not implicitly declared in the scope of the current statement.

---

## Function Prototypes

A function prototype defines the parameters to a function and defaults for optional parameters. Function prototypes identify call-by-reference parameters. Function prototypes do not define the operation of the function.

The following example is of a function prototype:

```
my_xor : function (
    a : polygon_layer,
    b : polygon_layer
) returning xorOut: polygon_layer;
```

The following example shows a function declaration.

```
my_xor : function (
    a : polygon_layer,
    b : polygon_layer
) returning xorOut: polygon_layer {
    tmp1 = not(a,b);
    tmp2 = not(b,a);
    xorOut = or(tmp1, tmp2 );
};
```

The following example shows calling the `my_xor()` function, which was defined in the preceding example.

```
a1 : polygon_layer;
b1 : polygon_layer;
xorA1B1 = my_xor(a1, b1);
```

---

## Function Calls

Function calls define how functions operate. A function call consists of the name of the function followed by instructions on how and where the function must be executed through arguments.

Values can be passed to the arguments in a function call in two different styles:

- By argument value alone (pass-by-value). The order of values must match the order of arguments in the function prototype. When you use this style, you must specify values for all optional arguments preceding an optional argument that is of interest to you because values are matched to arguments based on order.
- By name in `argument_name = argument_value` format (pass-by-name). The order of arguments is irrelevant. When you use this style, you do not need to specify values for all optional arguments preceding the optional argument that is of interest to you because values are matched to arguments based on name.

Both styles can be used in a single function call. However, in a function call, all argument values specified with the pass-by-value style must precede all argument values specified with the pass-by-name style. After you use the pass-by-name style in a function call, all following argument calls must be in the same style.

For example, the syntax of the `interacting()` function is:

```
interacting : binary published function(
    layer1 : polygon_layer,
    layer2 : polygon_layer,
    count : count_constraint_t = >0,
    include_touch : more_include_touch_e = EDGE,
    processing_mode : processing_mode_e = HIERARCHICAL
) returning result : polygon_layer;
```

The following is an example of calling the `interacting()` function with the pass-by-name style:

```
y = interacting( layer1 = a, layer2 = b, include_touch = ALL );
// The count argument is skipped, and the value is the default.
// The value of the processing_mode argument is the default.
```

The following is an example of calling the `interacting()` function with the pass-by-value style:

```
y = interacting( a, b, 2 );
// The value of count is 2.
// The values of the include_touch and processing_mode arguments
// are the defaults.
```

Here are two examples of calling the `interacting()` function with mixed styles:

```
y = interacting( a, b, include_touch = ALL );
// The count argument is skipped, and the value is the default.
// The value of the processing_mode argument is the default.

y = interacting( a, b, count = 2 );
// The values of the include_touch and processing_mode arguments
// are the defaults.
```

There are two styles of function calls:

- **Standard Function Call**

```
y = and(a, b);
y = and(a, b, CELL_LEVEL);
y = or(not(a,b),not(b,a));
```

- **Operator-Style Function Call**

```
y = a and b;
y = a and[CELL_LEVEL] b;
y = (a not b) or (b not a);
```

## Standard Function Call

A standard function call is always allowed for any function.

A standard function call consists of the name of the function followed by a parenthesized and comma-separated list of at least one expression, followed by a list of at least one name-value pair.

In the following example, the function has three parameters.

```
g: function ( x : integer, y : integer, z : integer)
    returning void;
```

Of the four function calls listed here, one is not legal:

```
g(x=3, y=10, z=10);    // allowed
g(z=3, y=10, x=4);     // allowed
g(10, 20, 30);         // allowed; x=10, y=20, z=30
```

```
g(z=3, 10, 20);      // ILLEGAL - expressions must not
                     // follow name-value pairs
```

If defaults are present for any parameters, the parameters are not required to be specified.

In the following example, the function has three parameters, two of which have defaults.

```
f: function
  ( x : integer,
    y : integer = 27,
    z : integer = 42
  ) returning void { ... }
...
```

Of the five function calls listed here, two are illegal:

```
f(10, y=2);      // allowed, z is 42
f(10);           // allowed, y is 27, z is 42
f(z=3, x=4);     // allowed, y is 27
f();             // ILLEGAL - x requires a value
f(z=20);         // ILLEGAL - x requires a value
```

## Operator-Style Function Call

An operator-style function call requires arguments to the left and right of the function name for binary functions. It requires other arguments to be specified as name-value pairs in square brackets immediately following the name of the function.

Operator-style function calls are allowed only for functions declared or defined with the binary qualifiers. Using operator-style function calls helps prevent the use of the function name in other contexts, such as variable names and member names.

The following syntax defines how a function can be called by using the binary qualifier:

```
length: binary function (l1: layer, c1: constraint of double) returning
result: layer {...}
...
LONGMET = MET1 length > 5;
```

With careful interface definition, binary allows for easy-to-read programs. For example,

```
// A simple declaration that takes two arguments.
// The LHS is the layer to be selected from; the RHS is the
// constraint defining the criteria area.
...
// ... as is a variable of appropriate type.
c : constraint = (27.0, 42.5);
x : lyr1 area c;
```

## Argument Bindings

Argument bindings determine which arguments at a call site are bound to which parameters in the function prototype.

- Select candidate prototypes based on the name of the function and the type required by the context. If the type cannot be determined, use only the function name. When an operator-style function call is made, the external arguments are bound to parameters in the prototype. For binary functions, the left argument is bound to the first parameter, and the right argument is bound to the second parameter.
- Each name-value argument is bound to the remaining like-named parameter.
- Any arguments that do not have names are bound to the remaining defaulted parameters from left to right.
- If all remaining unbound parameters have defaults, the function is used.
- It is an error if argument binding fails to match exactly one function prototype.

For example, given the function `MyFunc` defined, as follows:

```
MyFunc: binary function (
  fromLayer: lhs layer,
  touching: rhs layer,
  corners: boolean = true,
  whole_polygon: boolean
) returning e: edge_layer;
```

The following function calls are all equivalent:

```
T1 = MyFunc (metalLayer, selectLayer, true);
T2 = MyFunc (metalLayer, selectLayer, whole_polygon=true);
T3 = metalLayer MyFunc[whole_polygon=true] selectLayer;
```

The following code produces a compile-time error because the `whole_polygon` argument does not have a default:

```
/* illegal function call */
Tbroken = metalLayer MyFunc selectLayer;
```

## Defaults

Defaults are specified as right side assignments to option variables in the function prototypes. For example, here is the syntax of the `and()` function:

```
and : function(
  layer1 : polygon_layer,
  layer2 : polygon_layer,
  processing_mode : processing_mode_e = HIERARCHICAL,
  remove_hierarchical_overlap : boolean = true
) returning and_result : polygon_layer;
```

The following call of the `and()` function uses the defaults for the `processing_mode` and `remove_hierarchical_overlap` arguments.

```
c = and( a, b );
```

The following call of the `and()` function uses an optional setting for the `processing_mode` argument and the default for the `remove_hierarchical_overlap` argument.

```
c = and( a, b, CELL_LEVEL );
```

## Resetting Defaults

You can override the default of any function argument. Any argument can have its default changed; it can be numerical, a Boolean value, or an enumerator.

In this example, the default for the argument `x` of the `abc` function is set:

```
abc : literal function ( x : double = 123 ) returning z : double;
A = abc(); // == abc(123)
```

Important:

When you reset defaults, you are redefining the function, and all of its arguments must be listed.

---

## Function Overloading

PXL allows you to have more than one function with the same name in some cases.

Multiple functions can have the same name if PXL can determine which function is to be called by using the types and parameter names. If PXL is not able to determine the function to be called, it results in an error.

You can have two different functions with the same name if one returns a value and the other returns a void.

When a function appears at the statement level, it is assumed to be the void-returning function; when it appears in an expression, it is assumed to be the value-returning function:

```
MyFunction(a,b,c); // use the void-returning function
// "MyFunction"
x = MyFunction(a,b,c); // use the value-returning function
// "MyFunction"
```

---

## Function Recursion

Recursive function calling is not supported in PXL. For example:

```
noCanDo : function( ... ) returning integer {
  ...
```



```

    x = noCanDo(...);
    ...
}

```

---

## Program Structure

A PXL program is a sequence of declarations, function definitions, and statements. A program is executed from top to bottom.

PXL sets these preconditions for a program to be valid:

- All objects must be defined before they are used.
- All statements are evaluated only for side effects. An expression must be part of an assignment.

```
10 + 5; // illegal because the result (15) is not captured
```

---

## Compound Statement

A compound statement defines a nested scope in which local variables can be defined. Compound statements can be used anywhere a single statement is allowed. For information about scope, see [“Scope of Variables” on page 4-25](#).

- Compound statements allow local variables to hide previously defined variables at other scoping levels. The compiler reports a warning. Hiding is also known as shadowing.
- Compound statements use braces ({ }) to identify the compound statement.

```
{ ; } // Smallest legal compound statement
```

- Compound statements follow the following syntax:

```

{
    s1;
    s2;
    ...
    sN;
}

```

---

## Violations

Some IC Validator functions can produce violation data, which is also referred to as error output, instead of or in addition to layer output. These violations are stored in an SQLite database (the error database, PYDB), which is available to VUE for debugging, and written to the LAYOUT\_ERRORS file at the end of the run. There are also some PXL functions that take violation data as input, such as the `write_gds()` and `violation_empty()` functions.

If you want to directly access the error data produced by the IC Validator tool, see [Appendix E, “PYDB Perl API,”](#) for information about using the IC Validator Perl application programming interface (API).

## Violation Comments

All violation output is associated with a comment string. This comment appears as a heading in LAYOUT\_ERRORS file as well as in VUE. Any number of functions can output to the same violation comment.

The violation comment for current and nested PXL scopes is specified by using the @ operator. If no comment is specified in the runset, violations are given a default comment of "Violation".

The following example shows how to program violation comments that are unique for each layer in a set of layers. The violations are named “Metal1 spacing <= 0.1um”, “Metal2 spacing <= 0.1um”, and so forth for each layer.

```
for (i=0 to metal_layers.size()-1) {
    @ "Metal" + (i+1) + " spacing <= 0.1um";
    external1(metal_layers[i], distance<=0.1);
}
```

The following example shows the behavior of violation comments with respect to scope.

Note:

Typically, the violation comment definition (@ operator) should not be separated from the functions to which it applies. The following example highlights the behavior of violation comments with respect to scope.

```
@ "First Error Message";
{ @ "Second Error Message";           // A new scope is created here
  local_var = abc(layer2);
  internal1(local_var, distance < 0.2); // Errors from internal1()
                                         // are stored in the database
                                         // under "Second Error Message"
} // The original scope is restored here

external1(layer1, distance < 0.1); // Errors from external1() are stored
                                   // in the database under "First Error
                                   // Message"
```

The following example shows how to program violation comments that are unique for each layer in a set of layers. The violations are named “Metal1 spacing <= 0.1um”, “Metal2 spacing <= 0.1um”, and so forth for each layer.

```
for (i=0 to metal_layers.size()-1) {
    @ "Metal" + (i+1) + " spacing <= 0.1um";
    external1(metal_layers[i], distance<=0.1);
}
```

The `text_net()`, `texted_with()`, and `not_texted_with()` functions can output several different types of violations. For these functions, there is a way to override the current runset violation comment for individual types of errors. In this example, text shorts are output to the violation comment "Rule 4.5.6: No Shorts", while all other types of text errors are output to "Rule 1.2.3: No text errors".

```
@ "Rule 1.2.3: No text errors";

cdb1 = text_net(cdb1,
               { { M1, M1_text } },
               shorted_violation_comment = "Rule 4.5.6: No shorts");
```

The IC Validator tool processes violation comments using command-line options such as, `-svc`. For example:

```
{@ "A";
  {@ "B";
    internal1(...);
  };
};
```

The tool considers only the lower violation comment for selection. In this case, setting either `-svc B` or `-uvc A` execute the `internal1()` function, however, setting `-svc A` or `-uvc B` does not.

## Violation-Producing Functions

Most DRC and data generation functions are overloaded; that is, they have two versions. The version that returns a polygon layer does not produce error output. The void-returning version produces error output but not a polygon layer. Here are examples of the two versions of the `external1()` function.

```
// This external1 outputs to a polygon layer, no errors are produced
t1 = external1(layer1, distance < 0.1);

// This external1 outputs error data to the comment "Ext1 Check"
@ "Ext1 Check";
external1(layer1, distance < 0.1);
```

There are also some PXL functions that do not return void, but can also produce error output. For example, the `text_net()` function returns a connect database but also produces error output for text shorts, opens, and unused text.

## Violation Variables

The built-in PXL type violation can be used for variables to hold error data if there is a need to use it as input to another PXL function later in the runset. For example, violation variables can be used as input to functions such as `write_gds()` to output error polygons, or `violation_empty()` to determine if the violation contains any errors.

As the following example shows, some PXL functions return violations directly:

```
// These functions return violations for which there is not an associated
// PXL function
v1 = get_layout_grid_violation();
v2 = get_layout_drawn_violation();

// Write errors to GDSII
g = gds_library("err.gds");
write_gds(output_library = g,
          errors = { { v1, { 1 } },
                    { v2, { 2 } } });
```

**Note:**

Error output can be restricted by arguments such as `error_limit_per_check` in the `error_options()` function. Those options can influence how much data is actually stored in the error database. Discarded errors are not available for `write_gds()` and other functions.

## Violation Blocks

Violation variables can also be assigned by using violation blocks. Violation blocks are specified by using the `@=` operator. The left side is a violation variable. A code block enclosed by curly braces follows the `@=` operator. Within this block, all functions which produce error output are associated with the violation variable. For example,

```
// v1 contains the error output from the external1() check
v1 @= {
    @ "Rule 1";
    l1 = a and b not c;
    external1(l1, distance < 2);
}

// v2 contains the error output from both internal1() checks and
// the external1() check
v2 @= {
    @ "Rule 2";
    internal1(l2, distance < 1);
    internal1(l3, distance < 1);
    @ "Rule 3";
    external1(l2, distance < 1);
}

tdb : connect_database; // Declared outside the violation block scope
                        // so it can be used after the violation block

// v3 contains all text errors found by text_net()
v3 @= {
    @ "Rule 4";
    tdb = text_net(cdb, {{ M1, M1_text }});
}
```

```
// These functions can be used to get only certain types of errors
// from a violation. The result is a subset of the input violation.
v4 = get_text_shorted(v3); // v4 contains only text shorts from v3
v5 = get_text_unused(v3);  // v5 contains only unused text from v3

// Take some action if v4 is not empty
if (! violation_empty(v4)) {
    note("Found text shorts!");
}
```

---

## Execution Model

The execution model for PXL demands that resources be available. Any operation in a particular program is available for execution at any time if its preconditions are satisfied. There is a strong bias to execute diagnostics early, meaning that all the diagnostics that are ready are executed immediately during the initial processing of the program if possible. Therefore, you might see out-of-order execution of statements in a PXL program. The final results are unaffected by this reordering; however, intermediate results can appear at different times or simultaneously. This out-of-order execution is also known as parallel execution.

The same program, when executed different times, always generates the same results. These results can appear in a different order, however, based on availability of resources.

Possible runtime errors are

- Attempting to use any variable that has not been assigned a value.
- Indexing beyond the end of a list or with a negative index.
- Reading from a hash with a key value for which there is no element.

---

## Preprocessor

PXL uses a preprocessor to make the development and execution of programs easier and faster. A standard preprocessor module is integrated into the PXL compiler. [Table 4-27](#) shows the preprocessor capabilities.

**Note:**

The line length limit for the preprocessor is 64K characters. Exceeding this limit can usually be worked around by inserting a carriage return in the place of white space or a delimiter, such as a comma (,).

*Table 4-27 Standard Preprocessor Modules*

Typical preprocess capability	Syntax used
Conditional compilation	<code>#if . . . #ifdef, #ifndef, #elif, #else</code>
Error generation	<code>#error</code>
File inclusion	<p><code>#include</code>, including multiple directory search capabilities. The file name must be enclosed in either double quotation mark characters or angle brackets.</p> <ul style="list-style-type: none"> <li><code>#include &lt;icv.rh&gt;</code>. Use this include for IC Validator files. To locate this file, the IC Validator tool searches the <code>\$ICV_HOME_DIR/include</code> directory first. If <code>-I</code> is specified on the command line, the search order is <ol style="list-style-type: none"> <li>1. Directory specified by the <code>-I</code> option.</li> <li>2. <code>\$ICV_HOME_DIR/directory</code>.</li> </ol> </li> <li><code>#include "my_file.rh"</code>. Use this include for including your own files. The file name can include a directory, such as <code>#include "/checkpoint/restart_include.rh"</code>. To locate this file, the IC Validator tool searches the top-level runset directory first, followed by <code>\$ICV_HOME_DIR/include</code>. If <code>-I</code> is specified on the command line, the search order is <ol style="list-style-type: none"> <li>1. Top-level runset directory.</li> <li>2. Directory specified by the <code>-I</code> option.</li> <li>3. <code>\$ICV_HOME_DIR/directory</code>.</li> </ol> </li> </ul> <p>You can also use the <code>-I</code> command-line option. See <a href="#">“Command-Line Options” on page 1-6</a>.</p> <p>See the <i>IC Validator Installation Notes</i>, available on SolvNet, for more information about the IC Validator home directory.</p>
Location tracking	<code>#line</code>
Standard macro support	<p><code>#define . . . #undef</code>, including token construction (macro body # binary operator) and stringification (macro body ## unary operator).</p> <p>You can also use the <code>-D</code> command-line option to define variables. See <a href="#">“Command-Line Options” on page 1-6</a>.</p>

Use preprocessor directives to remove large blocks of code; that is, to enclose the section within a pair of `#if 0` and `#endif` statements. For example,

```
#if 0
// everything through the matching #endif is treated as a comment.
if (something == true)
{
    a line comment.
    do_something(useful = true);
}
#endif
```

Here is an example of using directives to enforce one time inclusion. The first time this file is encountered `MYFILE_RH` is not defined. The included `#define MYFILE_RH` sets this variable to be defined and the code body is included. The code body is never included again because `MYFILE_RH` exists and the `#ifndef` evaluates to false.

```
#ifndef MYFILE_RH
#define MYFILE_RH

// Body of code to include goes here

#endif // end if the #ifndef MYFILE_RH
```

When using `#if` statements,

- Use standard comparison operators:  
`==`, `<=`, `>=`, `&&`, `||`
- Macros that are not defined are interpreted as 0 (zero). For example, if `SOMEMACRO` is *not* defined, then `#if SOMEMACRO == 1` is translated as `#if 0 == 1`.
- Only compare integer values. Strings cannot be compared.

## Predefined Identifiers

During preprocessing, the predefined identifiers shown in [Table 4-28](#) are supported.

*Table 4-28 Predefined Identifiers*

Identifier	Content
<code>__LINE__</code>	Line number in the current source file as an integer
<code>__FILE__</code>	Name of the current source file, as a string constant
<code>__DATE__</code>	Date of compile, as an 11-character string literal, for example, "Apr 25 2006"
<code>__TIME__</code>	Time of compile as a string literal ("hh:mm:ss")

*Table 4-28 Predefined Identifiers (Continued)*

Identifier	Content
<code>__PXL_VERSION_MAJOR__</code>	Major version of the language as an integer
<code>__PXL_VERSION_MINOR__</code>	Minor version of the language as an integer
<code>__PXL_VERSION_PATCH__</code>	Patch level of the language as an integer
<code>__PXL_DATE__</code>	Date of compile, as an eight-digit integer constant as <code>yyyymmdd</code> , for example, 20060425
<code>__PXL_TIME__</code>	Time of compile, as a four-digit integer constant as <code>hhmm</code> , for example, 0014 or 2300

For example, `__PXL_DATE__` and `__PXL_TIME__` can be used to make code that changes behavior based on the current date. Because these variables are preprocessor variables, they can be used in `#if` and `#define` statements as well.

```
/**
 * define a new token, expire_20061231, that is be replaced
 * by the token deprecated before 2006.12.31 but expands
 * to obsolete thereafter.
 */
#if __PXL_DATE__ < 20061231
#define expire_20061231 deprecated
#endif

#ifdef expire_20061231
#define expire_20061231 obsolete
#endif
```

## Release Version Macros

Published macros allow you to compare the current release version to other release versions.

For example, you can determine if the current release version is at least as recent as the release version specified with integer values for year, month, service pack, and hot fix.

```
#if VERSION_GE(2009, 6, 1, 0)
. . .      // code that uses new features introduced in 2009.06.SP01
#else
. . .      // code that uses older method (pre 2006.06.SP01)
#endif
```

The VERSION macros are defined in [Table 4-29](#).



**Note:**

The *year*, *month*, *service\_pack*, and *patch* are integer values.

**Table 4-29** *VERSION* Macros

Macro	Definition
<code>VERSION_GE(year, month, service_pack, patch)</code>	Tests to see if the current release version is greater than or equal to the specified release version.
<code>VERSION_LE(year, month, service_pack, patch)</code>	Tests to see if the current release version is less than or equal to the specified release version.
<code>VERSION_GT(year, month, service_pack, patch)</code>	Tests to see if the current release version is greater than the specified release version.
<code>VERSION_LT(year, month, service_pack, patch)</code>	Tests to see if the current release version is less than the specified release version.
<code>VERSION_EQ(year, month, service_pack, patch)</code>	Tests to see if the current release version is equal to the specified release version.

## Date Comparison Macros

Date comparison macros allow you to change the behavior of a runset based on the current date.

For example, you can use the new macros to enable or disable certain PXL code based on the current date.

```
if (DATE_EQ(2015, 6, 4)) {
    note("Today == 2015-06-04");
} else {
    note("Today != 2015-06-04");
}
```

The DATE macros are defined in [Table 4-30](#).

**Note:**

The *year*, *month*, and *day* are integer values.

Specifying invalid or out-of-range values, such as a day value greater than 31 or a month value greater than 12, results in undefined behavior.

Table 4-30 DATE Macros

Macro	Definition
<code>DATE_EQ(year, month, day)</code>	Tests to see if the current date is equal to the date specified by the <i>year</i> , <i>month</i> , and <i>day</i> integers.
<code>DATE_GE(year, month, day)</code>	Tests to see if the current date is greater than or equal to the date specified by the <i>year</i> , <i>month</i> , and <i>day</i> integers.
<code>DATE_GT(year, month, day)</code>	Tests to see if the current date is greater than the date specified by the <i>year</i> , <i>month</i> , and <i>day</i> integers.
<code>DATE_LE(year, month, day)</code>	Tests to see if the current date is less than or equal to the date specified by the <i>year</i> , <i>month</i> , and <i>day</i> integers.
<code>DATE_LT(year, month, day)</code>	Tests to see if the current date is less than the date specified by the <i>year</i> , <i>month</i> , and <i>day</i> integers.

## Qualifiers

A qualifier affects an item it marks by changing its behavior or the operations that are allowed on it. [Table 4-31](#) defines the qualifiers.

Table 4-31 Qualifiers

Qualifier	Definition
binary	The function can be used as a binary operator as in <code>e1 FUNC e2</code> . The standard function call is of the form <code>FUNC(e1, e2)</code> .
const	The object causes an error each time it is assigned a new value. Initialization does not cause this error. Only attempts at direct assignment or passing through a function parameter qualified with <code>out</code> or <code>in_out</code> .
deprecated	The object causes a warning each time that it is defined or referenced. A function parameter being assigned an initial value in the function prototype or being used within the body of the function of which it is a parameter does not generate this warning.

Table 4-31 Qualifiers (Continued)

Qualifier	Definition
in_out	The function parameter requires an assignable item (an lvalue) in each function call. A nonassignable item or an item of the wrong type causes an error. The value of that item is also changed as a result of the function call. See <a href="#">“Assignment Operators” on page 4-27</a> for more information about lvalue.
inline	Tags a function as requiring expansion in an IC Validator run using the <code>-ndg</code> command-line option. See <a href="#">“Command-Line Options” on page 1-6</a> .
literal	The function parameter requires that the function argument be a literal expression. That is, the compiler must be able to resolve the value of the function argument at parse time.
obsolete	The item causes an error each time it is defined or referenced. A function parameter being assigned an initial value in the function prototype does not generate this error.
out	The function parameter requires an assignable item (an lvalue) in each function call. A nonassignable item or an item of the wrong type causes an error. This parameter is assumed to have no value when the function body is processed. The value of that item is also changed as a result of the function call. See <a href="#">“Assignment Operators” on page 4-27</a> for more information about lvalue.

A qualifier can be used to qualify only relevant items. The qualifiers are shown in [Table 4-32](#).

Table 4-32 Qualifier Items

Qualifier	Variable	Function	Parameter	Struct member
binary		yes		
const	yes		yes	
deprecated	yes	yes	yes	yes
in_out			yes	
inline		yes		
literal			yes	
obsolete	yes	yes	yes	yes
out			yes	

---

## Example of Modularized Code

- **function.rh**

```
#ifndef FUNCTION_RH
#define FUNCTION_RH

#include <icv.rh>

// Function prototypes
fx1 : function (
    lyr : polygon_layer,
    w   : double
) returning out: polygon_layer;

fx2 : function (
    lyr : polygon_layer,
    d   : double
) returning out: polygon_layer;

#include <functions.rs>

#endif
```

- **function.rs**

```
// Function definitions
fx1 : function (
    lyr : polygon_layer,
    w   : double
) returning out: polygon_layer {
    // code body
    out = f( lyr, w);
};

fx2 : function (
    lyr : polygon_layer,
    d   : double
) returning out: polygon_layer {
    // code body
    out = f( lyr, d);
};
```

- **main.rs**

```
#include <icv.rh>
#include <function.rh>

// assign(), library(), etc.

y = fx1( a, 0.1);
z = fx2( b, 0.3);
```

# 5

## Dynamic-Link Library Support

---

*This chapter explains using the dynamic-link feature of the IC Validator tool.*

Use the dynamic-link feature specifically for device extraction. This feature allows you to pass measurement data to C libraries that are external to the IC Validator tool, and then return the results from the C libraries back to the IC Validator tool. The C libraries must be shared (dynamic) libraries.

Dynamic-link library support is described in the following sections:

- [Dynamic-Link Functions](#)
- [Using the Dynamic-Link Functions](#)

---

## Dynamic-Link Functions

The following IC Validator functions support the dynamic-link library feature:

- `dev_dlink_library_close()` function
- `dev_dlink_library_open()` function
- `dev_dlink()` utility function
- `dev_dlink_library()` utility function
- `dlink_libraries` argument of all device configuration functions, such as the `capacitor()` function.

---

## Using the Dynamic-Link Functions

To use dynamic linking:

1. In your IC Validator runset,
  - a. Open the C library using the `dev_dlink_library_open()` function.
  - b. Specify the library handles in the IC Validator runset using the `dlink_libraries` argument in the device configuration functions.
  - c. Close the library using the `dev_dlink_library_close()` function.

For example,

```
dlink_handle = dev_dlink_library_open("library_name");

nmos(
    property_function=my_mos_func,
    dlink_libraries = {dlink_handle}
);

dev_dlink_library_close(dlink_handle);
```

2. Define the calculations for the device measurement data using the utility functions. The dynamic-link remote function is typically organized like this:

```
my_mos_func : function(void) returning void {
    //(1) Get measurement data for the device using the
    //     device utility functions
    //(2) Encode the measurement data into a 1-dimensional array
    //(3) Retrieve dynamic-link library file handle using the
    //     dev_dlink_library() utility function
    //(4) Call user-defined dynamic-link function using the dev_dlink()
    //     utility function, and pass essential information to the
```

```
    //    external shared libraries
        IC Validator data
}
```

3. The user-defined wrapper function, written in the C language, has this typical flow:

```
customer_wrapper_function() {
    //(1) Decode the measurement data into customers' format
    //(2) Decide function pointer according to string parameter
    //(3) Call internal function
    //(4) Return computed results
}
```





# 6

## Unified Fill

---

*Using the functions and PXL, the programming language of the IC Validator tool, you can create both simple and complex procedures to meet your fill requirements. This chapter describes the basics of how to use the `unified_fill()` function, an application-oriented, fill insertion function. This chapter also shows several usage models for runsets using the unified fill feature.*

Note:

The `unified _fill()` function is described in the *IC Validator Reference Manual*.

A fill pattern defined in the `unified_fill()` function can have either a single fill layer or multiple fill layers. That is, the function can output more than one polygon layer. Each fill-layer definition can have a single rectangle or any number of nonrectangular polygons. Additionally, every pattern can be viewed as a structure that is repeated inside the target region as defined by the spacing constraints.

The unified fill feature is described in the following sections:

- [Overview](#)
- [Types of Fill Patterns](#)
- [Fill-to-Fill Spacing](#)
- [Target Region and Fill-to-Signal Spacing](#)

- [Range and Width Dependent Spacing](#)
- [Polygon Grouping](#)
- [Signal Aligned Fills](#)
- [Improving Pattern Insertion](#)
- [Pitch Aligned Fills](#)
- [Criteria Analysis](#)
- [Output Layers](#)
- [Layer Compression](#)

---

## Overview

The `unified_fill()` function provides a high level of automation and flexibility in developing a fill procedure. The fill requirement might be as simple as generating an array of rectangular patterns inside a layer, or as complex as filling a design with a variety of patterns to meet a certain density target.

The `unified_fill()` function supports many options that you can configure in different ways for the desired fill requirement. Some of the high level features of this function are

- Ability to define different types of patterns
- Ability to define fill insertion sequences that use fill cells read from an external database
- Equation-based criteria analysis
- Layer coloring
- Signal-aligned fills
- Automated target layer creation

[Table 6-1](#) defines the terms used for the unified fill feature.

*Table 6-1 Unified Fill Terminology*

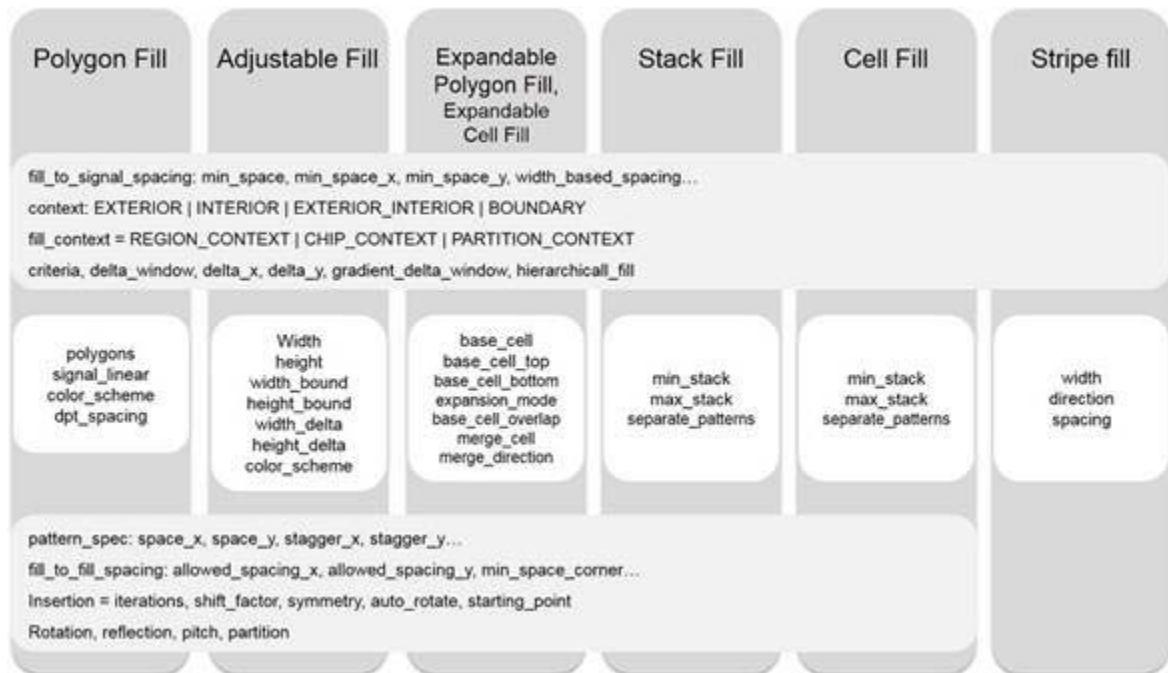
Term	Definition
Design layer and Signal layer	Existing layer in the design
Fill cell	Fill pattern definition that is imported to an IC Validator runset from an external layout database
Fill layer	Polygon layer output generated by the <code>unified_fill()</code> function
Target region and Fillable region	Region where fill can be added
Keepout region and Exclusion region	Region where fill must not be added
Fill pattern and Fill structure	Fill configuration composed of one or more fill layers. A pattern can be defined as either <ul style="list-style-type: none"> <li>• A single fill layer</li> <li>• Multiple fill layers</li> </ul>

Table 6-1 Unified Fill Terminology (Continued)

Term	Definition
Fill-to-signal spacing	Spacing from the fill pattern, or from the fill layer, to the design layer
Fill-to-fill spacing	Spacing between fill patterns or between fill layers

Figure 6-1 summarizes the features that are available for each type of fill pattern.

Figure 6-1 Fill Pattern Overview



## Types of Fill Patterns

The `unified_fill()` function supports the following types of fill pattern definitions:

- [Polygon Fill Patterns](#)
- [Adjustable Rectangle Fill Patterns](#)
- [Stack Fill Patterns](#)

- [Cell Fill Patterns](#)
- [Expandable Fill Cell Patterns](#)
- [Stripe Fill Patterns](#)

---

## Polygon Fill Patterns

A polygon fill pattern is a coordinate based definition of a fill pattern. For this type of fill pattern, set the `type` option to `UF_POLYGON`.

The fill pattern can have any number of fill layers. For a multilayer pattern definition, each fill layer has its own layer definition. As shown in [Example 6-1](#), a fill layer is identified with the name specified in the `output_layer_key` option of the `layers` option.

### Example 6-1 Single Layer Pattern Definition With Single Rectangle

```
DATA_M1_fill : list of list of coordinate_s = {
    { {0.0, 0.0}, {1.0, 0.0}, {1.0, 1.0}, {0.0, 1.0} }
};
SHAPE_M1_fill : polygon_layer_s = {
    layer_spec = {output_layer_key = "M1_fill"}
    polygons = DATA_M1_fill,
};

fill_patterns = {{
    type = UF_POLYGON,
    polygon_fill = {
        layers = {SHAPE_M1_fill}
    }
}}
```

## Single Layer Pattern Definition With Multiple Rectangles

The pattern definition shown in [Example 6-2](#) and [Figure 6-2](#) has one fill layer, `M1_fill`, that is defined by two rectangles. The dashed line indicates the bounding box of the layer definition. This pattern is repeated inside the target region defined by the spacing constraints. The bounding box of the pattern is the same as that of the layer definition because the pattern consists of only one layer.

### Example 6-2 One Fill-Layer Example Using a Fill Pattern of Type `UF_POLYGON`

```
DATA_M1_fill : list of list of coordinate_s = {
    { {0.0, 0.0}, {1.0, 0.0}, {1.0, 2.0}, {0.0, 2.0} },
    { {2.0, 0.0}, {3.0, 0.0}, {3.0, 2.0}, {2.0, 2.0} }
};

SHAPE_M1_fill : polygon_layer_s = {
    layer_spec = {output_layer_key = "M1_fill"}
    polygons = DATA_M1_fill,
}
```

```

};

fill_patterns = {{
    type = UF_POLYGON,
    polygon_fill = {
        layers = {SHAPE_M1_fill}
    }
}}

```

**Figure 6-2** One Fill-Layer Example



## Multilayer Pattern Definition

The pattern definition shown in [Example 6-3](#) and [Figure 6-3](#) has two fill layers, M1\_fill and M2\_fill. The dashed line indicates the bounding box of the pattern definition. This pattern is repeated inside the target region as defined by the spacing constraints. In this example, bounding box of the pattern is not same as that of the individual layer definitions. When repeating the pattern, the bounding box of the pattern as a whole is considered.

**Example 6-3** Two Fill-Layer Example Using a Fill Pattern of Type UF\_POLYGON

```

DATA_M1_fill : list of list of coordinate_s = {
    { {1.0, 0.0}, {2.0, 0.0}, {2.0, 3.0}, {1.0, 3.0} }
};

SHAPE_M1_fill : polygon_layer_s = {
    layer_spec = {output_layer_key = "M1_fill"}
    polygons = DATA_M1_fill,
};

DATA_M2_fill : list of list of coordinate_s = {
    { {0.0, 1.0}, {3.0, 1.0}, {3.0, 2.0}, {0.0, 2.0} }
};

SHAPE_M2_fill : polygon_layer_s = {
    layer_spec = {output_layer_key = "M2_fill"}
    polygons = DATA_M2_fill,
};

fill_patterns = {{
    type = UF_POLYGON,
    polygon_fill = {
        layers = {SHAPE_M1_fill,
                  SHAPE_M2_fill}
    }
}}

```

Figure 6-3 Two Fill-Layer Example



## Multiple Fill Pattern Definition

You can specify multiple pattern definitions in the `unified_fill()` function. The input target region is processed for each fill pattern definition in the order specified. [Example 6-5](#) and [Figure 6-4](#) show a fill procedure that first inserts the multilayer pattern, M1 and M2, followed by the single layer pattern, M1, in the remaining target region.

**Example 6-4** Multiple Patterns Example Using `fill_patterns` Option of `unified_fill()` Function

```
DATA_M1_fill : list of list of coordinate_s = {
    { {1.0, 0.0}, {2.0, 0.0}, {2.0, 3.0}, {1.0, 3.0} }
};

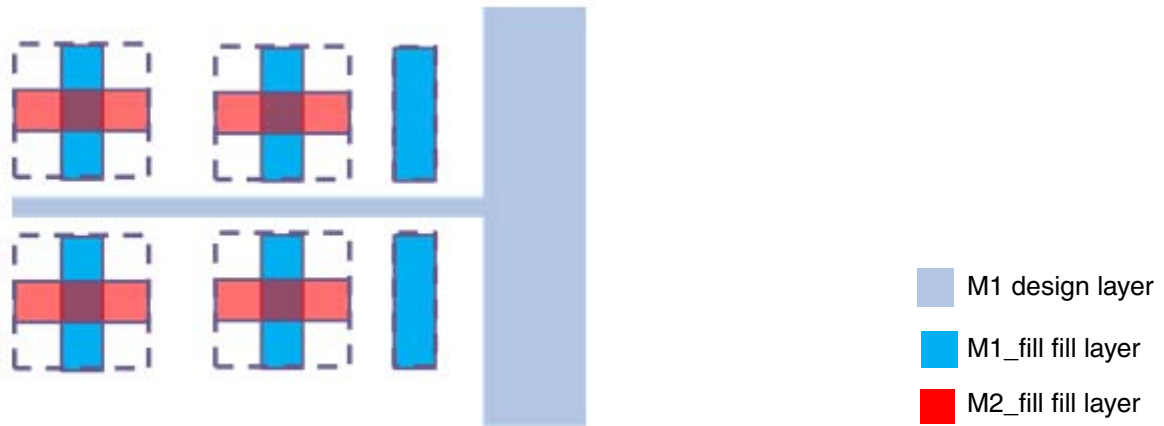
SHAPE_M1_fill : polygon_layer_s = {
    layer_spec = {output_layer_key = "M1_fill"}
    polygons = DATA_M1_fill,
};

DATA_M2_fill : list of list of coordinate_s = {
    { {0.0, 1.0}, {3.0, 1.0}, {3.0, 2.0}, {0.0, 2.0} }
};

SHAPE_M2_fill : polygon_layer_s = {
    layer_spec = {output_layer_key = "M2_fill"}
    polygons = DATA_M2_fill,
};

fill_patterns = {
    {type = UF_POLYGON,
      polygon_fill = {
        layers = {SHAPE_M1_fill,
                  SHAPE_M2_fill}
      }
    },
    {type = UF_POLYGON,
      polygon_fill = {
        layers = {SHAPE_M1_fill}
      }
    }
}
```

Figure 6-4 Multiple Patterns Example



## Adjustable Rectangle Fill Patterns

An adjustable rectangle fill pattern is a single-layer rectangle pattern definition that can automatically stretch or shrink the dimensions of the rectangle to maximize the density of the fill. For this type of fill pattern, set the `type` option to `UF_ADJUSTABLE`.

You can start with a specific rectangle dimension and choose to either grow or shrink all the shapes along the boundary of the target region. Growing and shrinking is done in user-specified increment values. Adjustment of the fill shapes is done only to the shapes along the boundary of the target region.

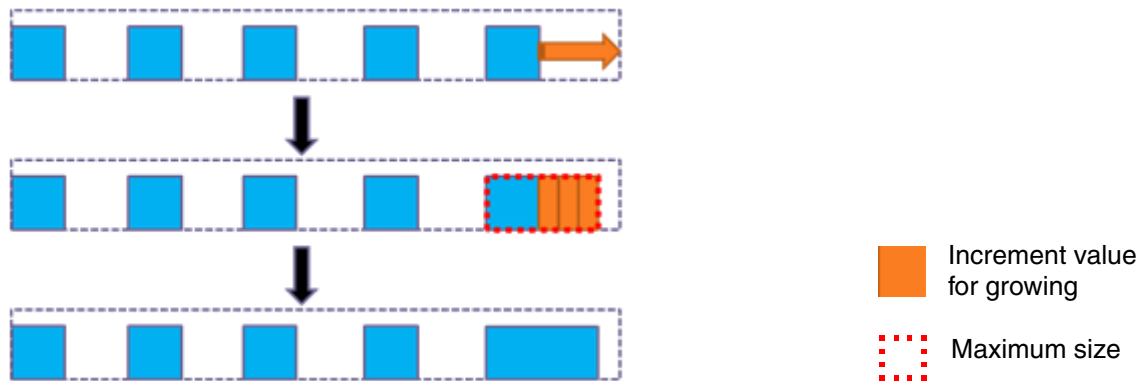
**Example 6-5** defines a minimum rectangle of dimension  $1\ \mu\text{m} \times 1\ \mu\text{m}$ . Rectangles along the boundary of the target region are grown in the x-direction, in increments of  $0.05\ \mu\text{m}$ , to fit the maximum possible shape. As shown in **Figure 6-5**, the maximum possible shape is  $4\ \mu\text{m} \times 1\ \mu\text{m}$ .

**Example 6-5** Adjustable Rectangle Fill Pattern Example Using a Fill Pattern of Type `UF_ADJUSTABLE`

```
fill_patterns = [{
    type = UF_ADJUSTABLE,
    adjustable_fill = {
        layer_spec = {output_layer_key = "M1_Fill"}
        width = 1.0,
        height = 1.0,
        width_bound = 4.0,
        width_delta = 0.05,
    }
}]
```



Figure 6-5 Adjustable Rectangle Fill Pattern Example



## Stack Fill Patterns

A stack fill pattern is similar to the `UF_POLYGON` pattern type but with more automation in developing a fill procedure. For this type of fill pattern, set the `type` option to `UF_STACK`.

You can use the `UF_STACK` type for applications that require multiple variations of a multilayer pattern to be inserted in sequence without having to explicitly define all the different variations. Using the options of the `unified_fill()` function, you provide a single pattern definition, consisting of multiple fill layers. Based on the fillable region, the tool tries to insert different variations of the fill pattern, starting with the original definition that contains all of the layers, and then proceeding to other variations of the pattern that contain a subset of the layers.

The goal is to find a variation of the pattern with the highest number of layers that can fit in a given fillable area. Use the `required_layer_keys` option of the `stack_fill` option to define dependencies between the fill layers, for example, `via1` requires `metal1` and `metal2`. [Example 6-6](#) shows an example.

**Example 6-6** Stack Fill Pattern Example Using `stack_fill` Option of `unified_fill()` Function

```
SHAPE_M1_fill : stack_layer_s = {
    layer_spec = {output_layer_key = "M1_fill"}
    polygons = DATA_M1_fill,
};
SHAPE_M2_fill : stack_layer_s = {
    layer_spec = {output_layer_key = "M2_fill"}
    polygons = DATA_M2_fill,
};
SHAPE_M3_fill : stack_layer_s = {
    layer_spec = {output_layer_key = "M3_fill"}
    polygons = DATA_M3_fill,
};
SHAPE_V1_fill : stack_layer_s = {
    layer_spec = {output_layer_key = "V1_fill"},
```

```

        polygons = DATA_V1_fill,
        required_layer_keys = {"M1_fill", "M2_fill"}
    };
    SHAPE_V2_fill : stack_layer_s = {
        layer_spec = {output_layer_key = "V2_fill"},
        polygons = DATA_V2_fill,
        required_layer_keys = {"M2_fill", "M3_fill"}
    };

    fill_patterns = {{
        type = UF_STACK,
        stack_fill = {
            layers = {SHAPE_M1_fill,
                      SHAPE_V1_fill,
                      SHAPE_M2_fill,
                      SHAPE_V2_fill,
                      SHAPE_M3_fill},
            min_stack = 1,
            max_stack = 5
        }
    }}

```

SHAPE\_M1\_fill, SHAPE\_V1\_fill, SHAPE\_M2\_fill, SHAPE\_V2\_fill, and SHAPE\_M3\_fill are individual layer definitions of a single multilayer pattern. The `unified_fill()` function processes the fillable region with the following patterns in the order shown:

```

M1-V1-M2-V2-M3
M1-V1-M2
M2-V2-M3
M1
M2
M3

```

The previous `UF_STACK` example is equivalent to writing multiple pattern definitions of the `UF_POLYGON` type, as shown in [Example 6-8](#).

#### Example 6-7 Writing Multiple Pattern Definitions Using the `UF_POLYGON` Type

```

fill_patterns = {
    {type = UF_POLYGON,
      polygon_fill = {          // First pattern for insertion
        layers = {SHAPE_M1_fill,
                  SHAPE_V1_fill,
                  SHAPE_M2_fill,
                  SHAPE_V2_fill,
                  SHAPE_M3_fill}
      }
    },
    {type = UF_POLYGON,
      polygon_fill = {          // Second pattern for insertion
        layers = {SHAPE_M1_fill,
                  SHAPE_V1_fill,

```

```

        SHAPE_M2_fill}
    },
    {type = UF_POLYGON,
      polygon_fill = {          // Third pattern for insertion
        layers = {SHAPE_M2_fill,
                  SHAPE_V2_fill,
                  SHAPE_M3_fill}
      }
    },
    {type = UF_POLYGON,
      polygon_fill = {          // Fourth pattern for insertion
        layers = {SHAPE_M1_fill}
      }
    },
    {type = UF_POLYGON,
      polygon_fill =           // Fifth pattern for insertion
        {layers = {SHAPE_M2_fill}
      }
    },
    {type = UF_POLYGON,
      polygon_fill =           // Sixth pattern for insertion
        {layers = {SHAPE_M3_fill}
      }
    }
  }
}

```

The order of the individual layer definitions determines the order of pattern variations that are placed. If the layers are defined as shown in [Example 6-8](#).

**Example 6-8 Order of Individual Layer Definitions Example**

```

fill_patterns = {{
  type = UF_STACK,
  stack_fill = {
    layers = {SHAPE_M3_fill,
              SHAPE_V2_fill,
              SHAPE_M2_fill,
              SHAPE_V1_fill,
              SHAPE_M1_fill},
    min_stack = 1,
    max_stack = 5
  }
}}

```

The order of pattern insertion sequence is changed to:

```

M1-V1-M2-V2-M3
M2-V2-M3
M1-V1-M2
M3
M2

```

## M1

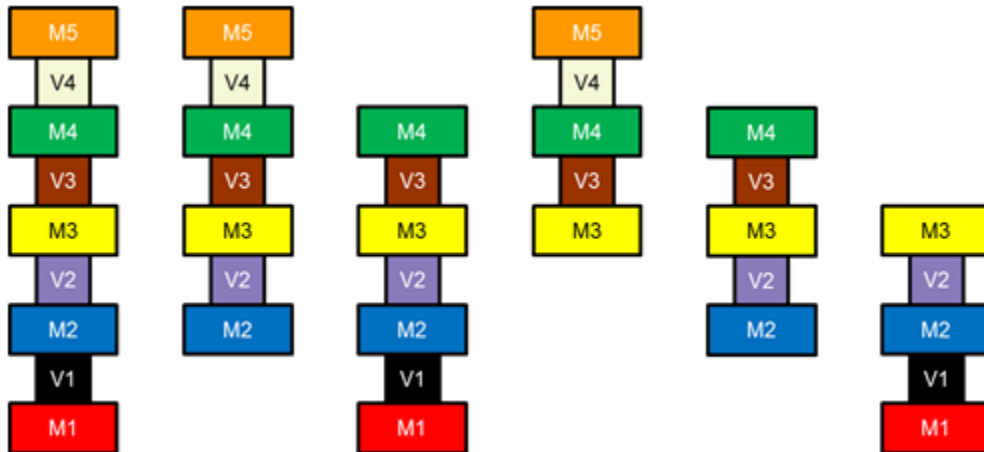
The number of variations of a pattern that can be constructed by the tool depends on the values of the `min_stack` and `max_stack` options.

- The `min_stack` value specifies the minimum number of fill layers required in any variation.
- The `max_stack` value specifies the maximum number of fill layers allowed in a variation.

Figure 6-6 shows an example of all pattern variations for a fill application in which the pattern definition consists of 5 metals and 4 vias as the fill layers. In this example,

`max_stack = 9`, `min_stack = 5`

Figure 6-6 Example of All Pattern Variations for a Fill Application

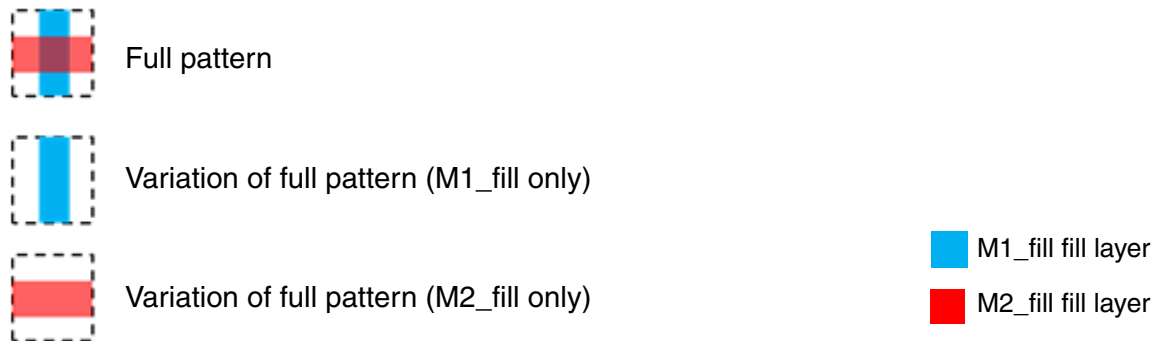


Depending on the layer definitions in the pattern, the bounding box of the pattern might be larger than the individual bounding box of each layer definition. In such cases, the bounding box of the smaller variations of the pattern might be different from that of the complete pattern definition, which consists of all the layers.

By default, the `unified_fill()` function retains the bounding box of the complete pattern for all of its variations. All automatic variations of the pattern by default align with the bounding box of the complete pattern, as shown in Figure 6-6.

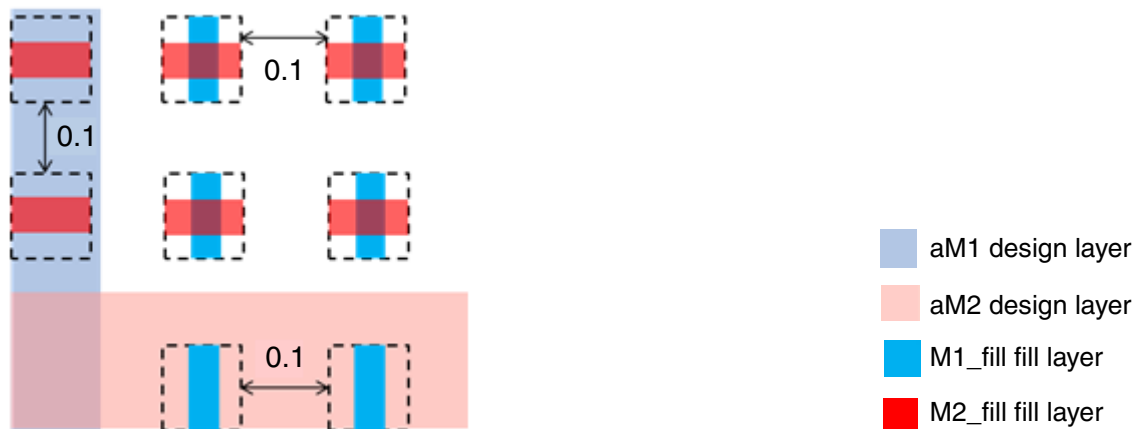
Example 6-9 and Figure 6-8 show an example where the full pattern is a structure with two fill layers, `M1_fill` and `M2_fill`, which are shown in Figure 6-7. There are two variations of this pattern, a pattern with only `M1_fill` and a pattern with only `M2_fill`. However, all patterns have the same bounding box as that of the full pattern. As a result, it might appear that the individual rectangles on `M1_fill` are spaced farther apart than the specified spacing value. This is because the spacing is based on the bounding box.

Figure 6-7 Two Fill Layers Example

Example 6-9 Pattern Structure With Two Fill Layers Using `stack_fill` Option of `unified_fill()` Function

```
fill_patterns = [{
    type = UF_STACK,
    stack_fill = {
        pattern_spec = {space_x = 0.1,
                        space_y = 0.1},
        layers = {SHAPE_M1_fill,
                  SHAPE_M2_fill},
        min_stack = 1,
        max_stack = 2
    }
}]
```

Figure 6-8 Pattern Structure With Two Fill Layers



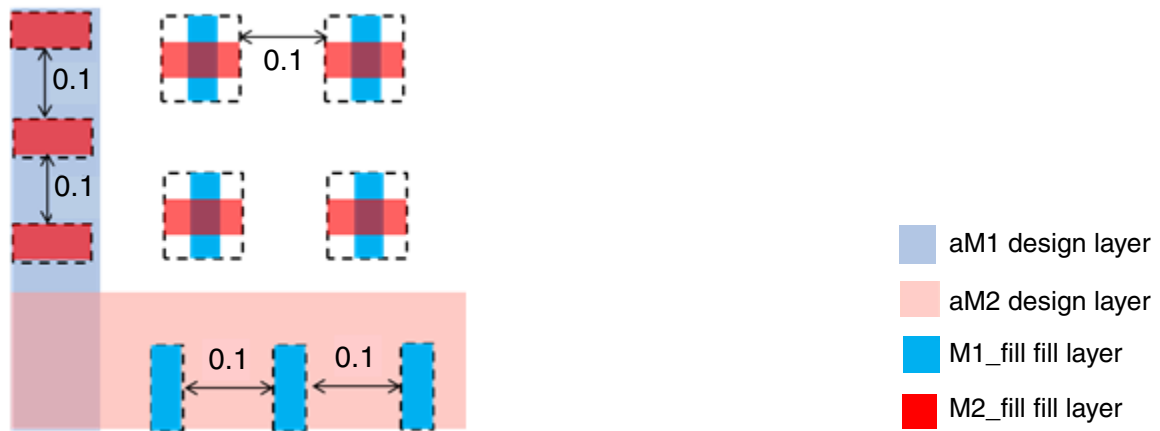
By default, the `unified_fill()` function retains the bounding box of the complete pattern for all of its variations.

You can change this alignment behavior by setting the `separate_patterns` option to `true`. In that case, every pattern variation has its own bounding box based on the extents of the

subset of layers in it. When inserting a pattern variation, the bounding box of that specific pattern variation is used instead of the bounding box of the complete pattern.

In [Figure 6-9](#), the two variations of the full pattern have their own bounding box. As a result, they appear to be placed closer when compared to the behavior when the `separate_patterns` option is false.

*Figure 6-9 Pattern Structure With Two Fill Layers with `separate_patterns` Option True*



When the `separate_patterns` option is false,

- The `fill_context`, `iterations`, and `shift_factor` options are ignored.
- The `min_stack` and `max_stack` values are ignored. In other words, the tool works with all possible variations of the pattern.

## Cell Fill Patterns

A cell fill pattern lets you import patterns from an external database instead of manually defining them in the runset. For this type of fill pattern, set the `type` option to `UF_CELL`.

The `UF_CELL` type provides an additional interface to import a cell, from an external GDSII or OASIS library, into the runset and map the layers in that cell to the fill layers. All other functionality is the same as with the `UF_POLYGON` and `UF_STACK` types. The imported pattern definition can be used either as is, as with the `UF_POLYGON` type, or chosen to be broken down to smaller patterns, as with the `UF_STACK` type.

A fill cell or a library of fill cells can be maintained along with the fill runset. These fill cells contain the fill layers that the runset can import and insert in the target region based on the specified constraints. See the `import_gds_cell()` and `import_oasis_cell()` functions in the *IC Validator Reference Manual* for information about importing a cell into the runset.

In [Example 6-10](#), the `M1_fill`, `V1_fill`, `M2_fill`, `V2_fill` and `M3_fill` layer definitions are imported from the `FillCell` cell in the `Fill_Cell.gds` file. For example, Layer (1;0) in `Fill_Cell.gds` is

designated for the M1\_fill layer. See [“Stack Fill Patterns” on page 6-9](#) for examples of additional functionality that are conceptually equivalent.

#### Example 6-10 Cell Fill Pattern Example

```
Fill_Cell_lib = gds_library("./Fill_Cell.gds");
Fill_Cell = import_gds_cell(Fill_Cell_lib, "FillCell");

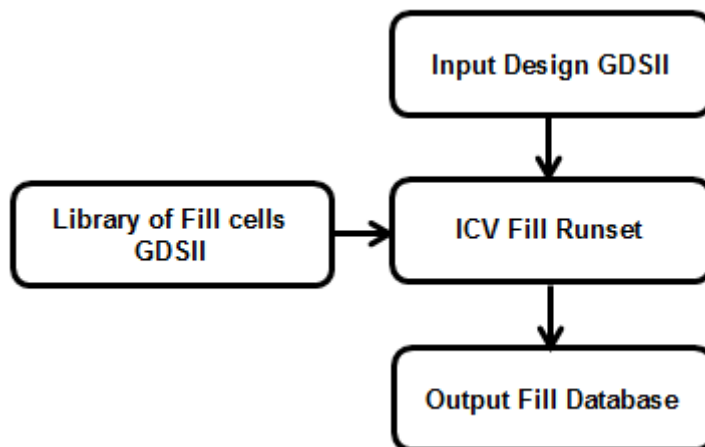
STACK_metal_fill : cell_fill_s =
{
    cell = Fill_Cell
    layers = {
        // Map Individual layers of the cell to Fill layers
        {ldt_list = {{1, 0}}, layer_spec = {output_layer_key = "M1_fill"}},
        {ldt_list = {{2, 0}}, layer_spec = {
            output_layer_key = "V1_fill"},
            required_layer_keys = {"M1", "M2"}},
        {ldt_list = {{3, 0}}, layer_spec = {output_layer_key = "M2_fill"}},
        {ldt_list = {{4, 0}}, layer_spec = {
            output_layer_key = "V2_fill"},
            required_layer_keys = {"M2", "M3"}},
        {ldt_list = {{5, 0}}, layer_spec = {output_layer_key = "M3_fill"}}
    },
    min_stack = 1,
    max_stack = 5,
    rotation = NONE,
    reflection = false
};
```

#### Note:

Depending on your requirements, the UF\_STACK type provides more automation in developing a fill procedure over the UF\_POLYGON type. Similarly, the UF\_CELL type provides more automation over the UF\_STACK and UF\_POLYGON types.

[Figure 6-10](#) shows an example usage model for the UF\_CELL type.

Figure 6-10 Usage Model for the UF\_CELL Type



## Expandable Fill Cell Patterns

An expandable fill-cell pattern lets you insert fill cells that get expanded along the boundaries of a fill target area. For this type of fill pattern, set the `type` option to either `UF_EXPANDABLE` or `UF_EXPANDABLE_CELL`.

The `UF_EXPANDABLE_CELL` type provides an additional interface to import a cell, from an external GDSII or OASIS library, into the runset and map the layers in that cell to the fill layers. All other functionality is the same as with the `UF_EXPANDABLE` type.

To use an expandable fill-cell type pattern, first define a base fill cell with incremental unit cells. For example, as shown in Figure 6-11, a simple base cell is made up of three incremental units. In this example, only the middle unit is repeatable.

Figure 6-11 Repeatable Cell Example

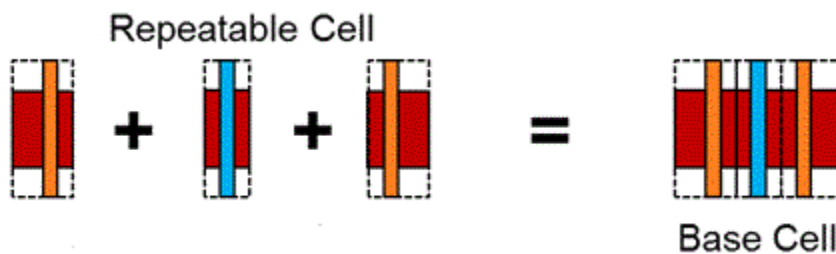
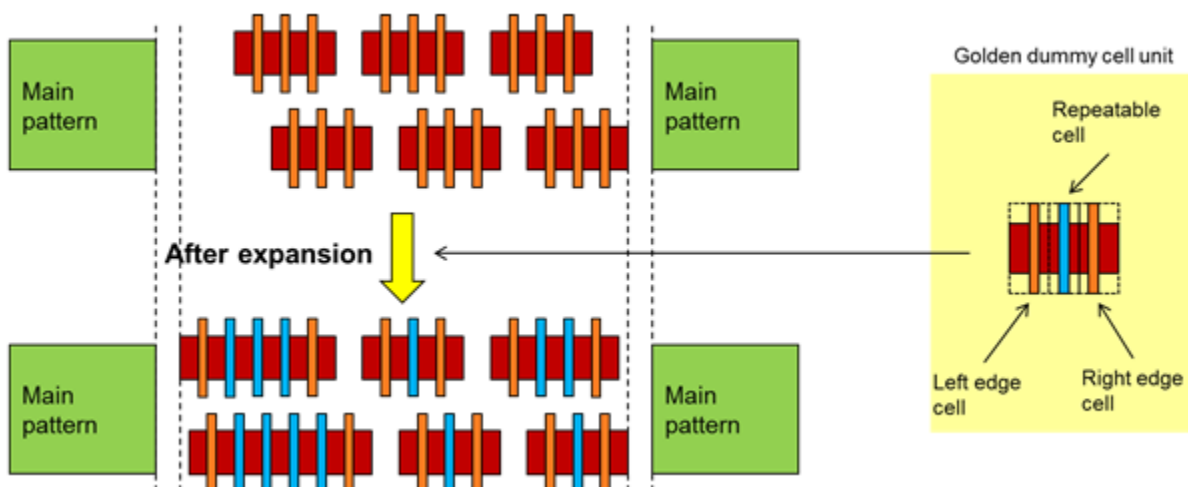


Figure 6-12 shows the expansion of a fill cell. The top half of the figure shows the base cell being inserted before the expansion. The bottom half of the figure shows the base cells that were expanded by repeating the middle unit.

Figure 6-12 Expansion of a Fill Cell



Example 6-11 shows how to program the `UF_EXPANDABLE` type.

1. Define the layers used in each incremental unit.



2. Declare the actual three incremental unit cells that make up the base cell.
3. Call the `unified_fill()` function and retrieve the output layers.

### Example 6-11 UF\_EXPANDABLE Type Example

```
// Define the layers used in each incremental unit
EXPAND_CELL_LEFT_LAYER_1 : polygon_layer_s = {
    layer_spec = {output_layer_key = "OUTPUT_EXPAND_CELL_LEFT_LAYER_1"},
    polygons = {{ {0.045, 0.193}, {0.261, 0.103}, {0.261, 0.287}, {0.058, 0.247} }}
};
EXPAND_CELL_LEFT_LAYER_2 : polygon_layer_s = {
    layer_spec = {output_layer_key = "OUTPUT_EXPAND_CELL_LEFT_LAYER_2"},
    polygons = {{ {0.045, 0.128}, {0.191, 0.128}, {0.191, 0.432}, {0.155, 0.232} }}
};

EXPAND_CELL_MIDDLE_LAYER_1 : polygon_layer_s = {
    layer_spec = {output_layer_key = "OUTPUT_EXPAND_CELL_MIDDLE_LAYER_1"},
    polygons = {{ {0.045, 0.193}, {0.261, 0.103}, {0.261, 0.287}, {0.058, 0.247} }}
};
EXPAND_CELL_MIDDLE_LAYER_2 : polygon_layer_s = {
    layer_spec = {output_layer_key = "OUTPUT_EXPAND_CELL_MIDDLE_LAYER_2"},
    polygons = {{ {0.045, 0.128}, {0.191, 0.128}, {0.191, 0.432}, {0.155, 0.232} }}
};

EXPAND_CELL_RIGHT_LAYER_1 : polygon_layer_s = {
    layer_spec = {output_layer_key = "OUTPUT_EXPAND_CELL_RIGHT_LAYER_1"},
    polygons = {{ {0.045, 0.193}, {0.261, 0.103}, {0.261, 0.287}, {0.058, 0.247} }}
};
EXPAND_CELL_RIGHT_LAYER_2 : polygon_layer_s = {
    layer_spec = {output_layer_key = "OUTPUT_EXPAND_CELL_RIGHT_LAYER_2"},
    polygons = {{ {0.045, 0.128}, {0.191, 0.128}, {0.191, 0.432}, {0.155, 0.232} }}
};

// Declare the actual three incremental unit cells that make up the base cell
cell_A_left : base_cell_s = {
    layers = {EXPAND_CELL_LEFT_LAYER_1, EXPAND_CELL_LEFT_LAYER_2 },
    repeatable = false
};
cell_B_middle : base_cell_s = {
    layers = { EXPAND_CELL_MIDDLE_LAYER_1, EXPAND_CELL_MIDDLE_LAYER_2 },
    repeatable = true,
    maximum = 10
};
cell_C_right : base_cell_s = {
    layers = { EXPAND_CELL_RIGHT_LAYER_1, EXPAND_CELL_RIGHT_LAYER_2 },
    repeatable = false
};

// Call unified_fill()
expand_fill_cell_output = unified_fill(
    fill_patterns = {
        {type = UF_EXPANDABLE,
         expandable_polygon_fill = {
             pattern_spec = {
                 space_x = 0.12,
                 space_y = 0.06,
                 stagger_x = 0.2 },

```

```

        base_cell = {cell_A_left, cell_B_middle, cell_C_right} }
    },
    fill_boundary = { type = LAYER, layer = REGION_CELL2_CORE }
);

// Retrieve the output layers
expand_fill_cell_layer_1 =
    expand_fill_cell_output [ "OUTPUT_EXPAND_CELL_LEFT_LAYER_1" ][0]
    or expand_fill_cell_output [ "OUTPUT_EXPAND_CELL_MIDDLE_LAYER_1" ][0]
    or expand_fill_cell_output [ "OUTPUT_EXPAND_CELL_RIGHT_LAYER_1" ][0]

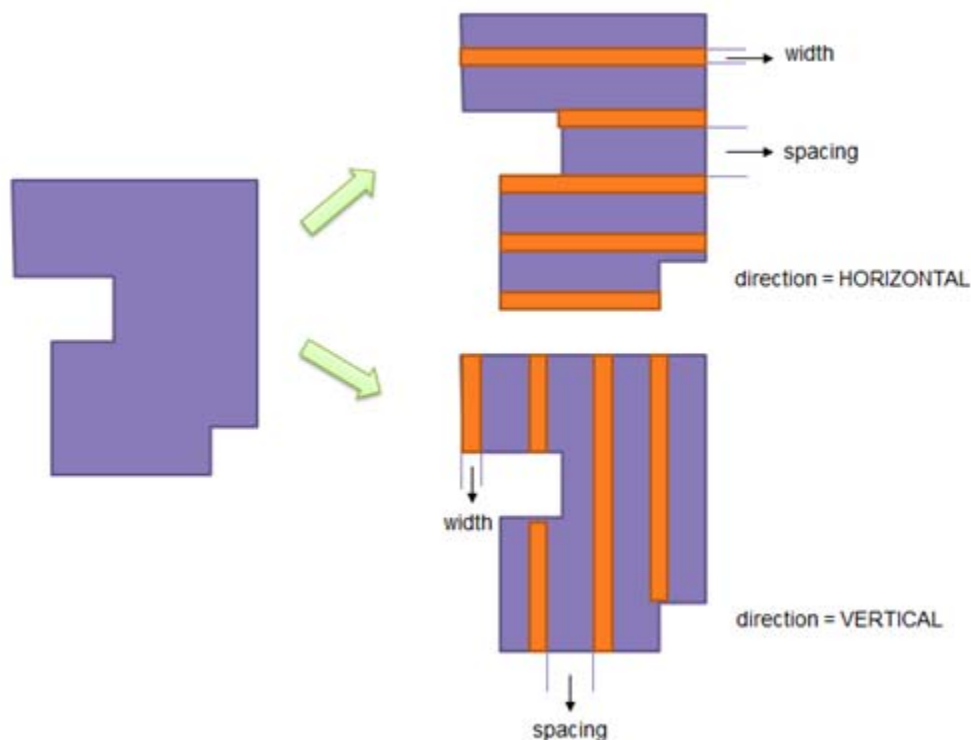
expand_fill_cell_layer_2 =
    expand_fill_cell_output [ "OUTPUT_EXPAND_CELL_LEFT_LAYER_2" ][0]
    or expand_fill_cell_output [ "OUTPUT_EXPAND_CELL_MIDDLE_LAYER_2" ][0]
    or expand_fill_cell_output [ "OUTPUT_EXPAND_CELL_RIGHT_LAYER_2" ][0]

```

## Stripe Fill Patterns

A stripe fill pattern lets you insert a stripe-shape fill across target areas. For this type of fill pattern, set the `type` option to `UF_STRIPE`. You specify the direction, width, and spacing of the stripes, as shown in [Figure 6-12](#).

Figure 6-13 Stripe Fill Patterns

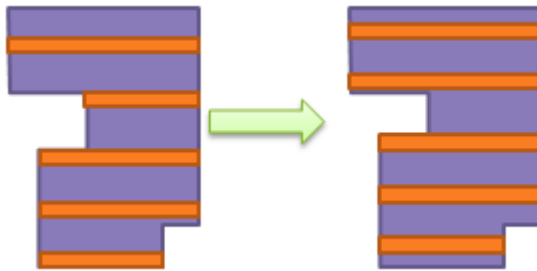


When symmetry is specified for stripe fill, the `unified_fill()` function

1. Gets the bounding box
2. Finds the center of the bounding box
3. Places the first stripe with its center line on the center of the bounding box
4. Places subsequent stripes at equal distances from the center stripe
5. Saves only full width stripes

Figure 6-14 shows the result of symmetry for stripes.

*Figure 6-14 Symmetry of Stripes*



Example 6-12 shows a `unified_fill()` call using the `UF_STRIPE` type.

*Example 6-12 UF\_STRIPE Type Example*

```
result = unified_fill(
    fill_patterns = {
        {type = UF_STRIPE,
         stripe_fill = {
             layer_spec = {
                 output_layer_key = "DM1_FILL",
             },
             direction = HORIZONTAL,
             width = 0.1,
             spacing = 0.2,
         }
    },
    fill_boundary = {
        type = LAYER,
        layer = M1
    },
    extents_output = {
        {output_layer_key = "OUTPUT_DM_EXTENTS"}
    }
);
```

---

## Fill-to-Fill Spacing

As described earlier in this chapter, you can configure the `unified_fill()` function to insert different types of patterns into specified given target regions. By providing a list of pattern definitions, you instruct the tool to fill the target region by repeating each pattern definition in sequential steps.

The fill-to-fill spacing feature lets you define:

- [Spacing Between Instances of the Same Patterns](#)
- [Spacing Between Different Patterns](#)

---

### Spacing Between Instances of the Same Patterns

Spacing between instances of the same pattern is controlled by the `space_x`, `space_y`, `stagger_x` and `stagger_y` options in the `pattern_spec` option.

Note:

These values can be negative in certain conditions to support overlap of pattern placements.

Additionally, for the spacing between fill polygons and patterns that are placed inside different target regions, you can set the spacing between patterns by using the `pattern_spacing` option inside the `pattern_spec` option, or you can set the spacing between individual fill layers by using the `fill_to_fill_spacing` option inside the `layer_spec` option. The options are:

- `allowed_spacing_x`
- `allowed_spacing_y`
- `min_space_corner`
- `extension`

See the `unified_fill()` function for descriptions of these options. Use these options to trigger more complex layer-specific spacing checks during fill treatment. The `space_x`, `space_y`, `stagger_x`, and `stagger_y` options apply to the bounding box of the entire pattern definition in accordance with the `separate_patterns` option.

By design, the `unified_fill()` function makes sure that no two patterns get closer than the `space_x` and `space_y` values. In [Example 6-13](#), `space_x` and `space_y` values are specified for the multilayer pattern definition. If patterns falling in adjacent polygons of the target region come too close, one of them is not inserted to meet the fill-to-fill spacing constraint, as shown in [Figure 6-15](#).

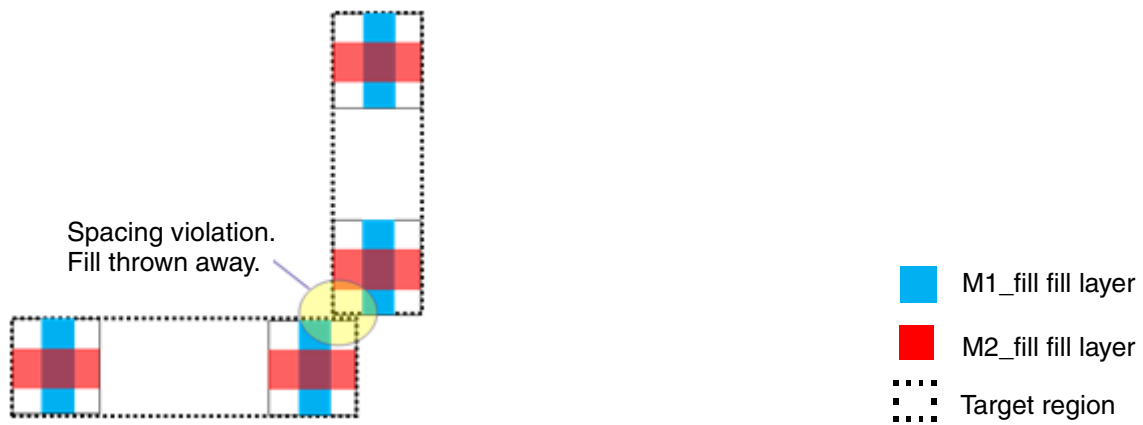
**Example 6-13 Spacing Between Same Patterns Using the space\_x and space\_y Options**

```

SHAPE_M1_fill : polygon_layer_s = {
  layer_spec = {output_layer_key = "M1_fill"}
  polygons = DATA_ M1_fill,
};
SHAPE_M2_fill : polygon_layer_s = {
  layer_spec = {output_layer_key = "M2_fill"}
  polygons = DATA_ M2_fill,
};

fill_patterns = {{
  type = UF_POLYGON,
  polygon_fill = {
    pattern_spec = {space_x = 0.1,
                    space_y = 0.1},
    layers = {SHAPE_M1_fill,
              SHAPE_M2_fill}
  }
}}

```

**Figure 6-15 Spacing Between Same Patterns Example**

In [Example 6-14](#), the `allowed_spacing_x` and `allowed_spacing_y` values are specified in the `pattern_spacing` option for the multilayer pattern definition. This ensures the necessary spacing between pattern instances from different target areas. As shown in [Figure 6-16](#), both pattern instances are placed because the spacing in the x- and y-directions respects the `allowed_spacing_x` and `allowed_spacing_y` values.

**Example 6-14 Spacing Between Multilayer Patterns With Allowed Spacing Values Smaller Than Space Values**

```

fill_patterns = {{
  type = UF_POLYGON,
  polygon_fill = {
    pattern_spec = {space_x = 0.1, space_y = 0.1,
                    pattern_spacing = {
                      allowed_spacing_x = {>= 0.02},
                      allowed_spacing_y = {>= 0.02}
                    }
  }
}}

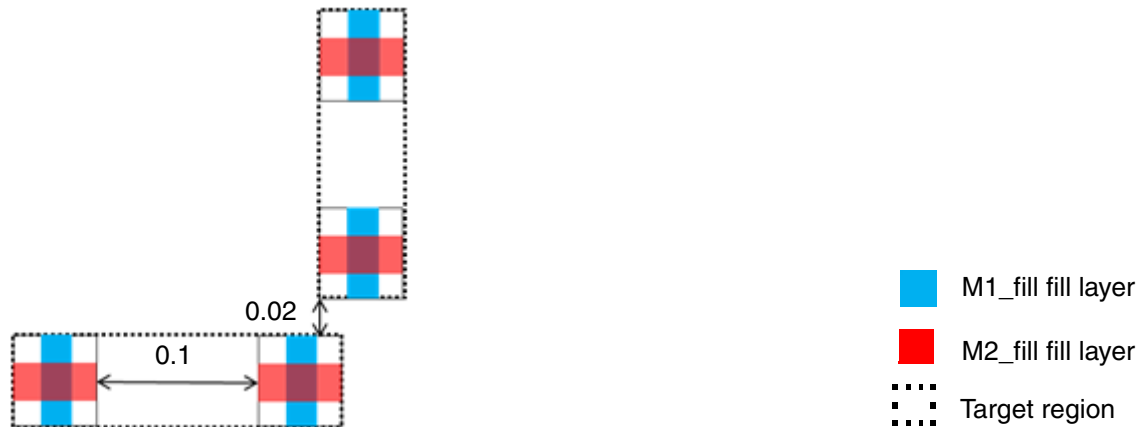
```

```

    },
    layers = {SHAPE_M1_fill,
              SHAPE_M2_fill}
  }
}

```

**Figure 6-16** Spacing Between Multilayer Patterns With Allowed Spacing Values Smaller Than Space Values Example



In [Example 6-15](#), the `allowed_spacing_x` and `allowed_spacing_y` values are specified in the `fill_to_fill_spacing` option for the `M1_fill` fill layer. Even though the space between the bounding boxes violates the `space_x` and `space_y` values, the tool does not remove patterns if the individual layers inside the pattern satisfy the `allowed_spacing_x` and `allowed_spacing_y` values, as shown in [Figure 6-17](#).

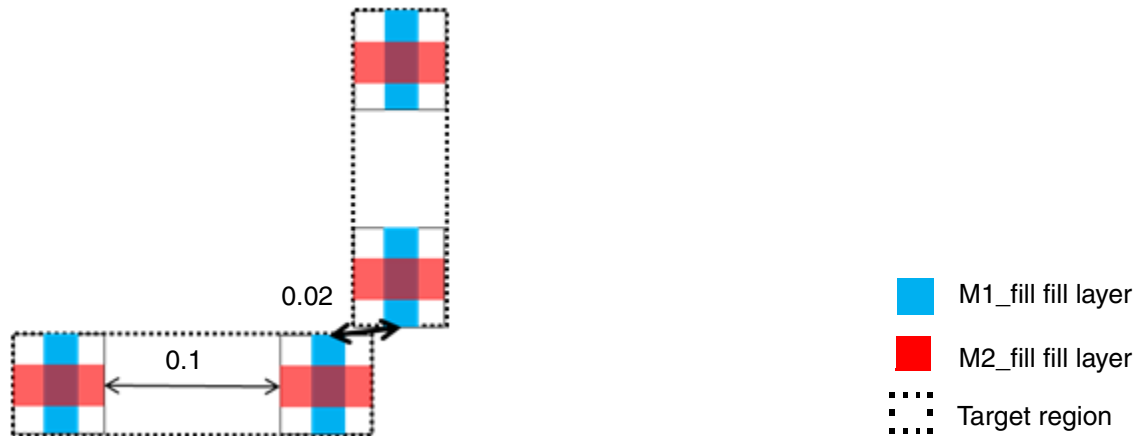
**Example 6-15** Using the Allowed Spacing Values in the `fill_to_fill_spacing` Option

```

SHAPE_M1_fill : polygon_layer_s = {
  layer_spec = {
    output_layer_key = "M1_fill",
    fill_to_fill_spacing = {
      allowed_spacing_x = {>= 0.02},
      allowed_spacing_y = {>= 0.02}
    }
  }
  polygons = DATA_ M1_fill,
};
fill_patterns = {{
  type = UF_POLYGON,
  polygon_fill = {
    pattern_spec = {space_x = 0.1, space_y = 0.1},
    layers = {SHAPE_M1_fill,
              SHAPE_M2_fill}
  }
}}

```

Figure 6-17 Using the Allowed Spacing Values in the `fill_to_fill_spacing` Option Example



## Spacing Between Different Patterns

When processing a list of fill patterns, you can specify spacing values between different patterns using the `other_pattern_spacing` option. This option is a hash of integer to double constraint where the integer represents the order of the pattern within the list of patterns. The integer 0 represents the first pattern. For example, to set the spacing to the third pattern to be 0.5 or greater, write

```
other_pattern_spacing = {2 => >= 0.5}
```

Note:

The spacing is always applied to the bounding box of every pattern.

Figure 6-18 shows an application that uses 3 patterns, A, B, and C. The spacing between the patterns is expected to be as follows:

Pattern A:

- Space to Pattern B = 2
- Space to Pattern C = 1

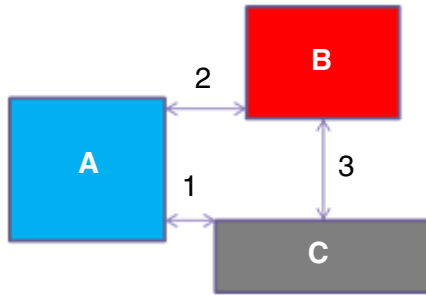
Pattern B:

- Space to Pattern A = 2
- Space to Pattern C = 3

Pattern C:

- Space to Pattern A = 1
- Space to Pattern B = 3

Figure 6-18 Spacing Between Different Patterns Example



Use the appropriate indexes in the `other_pattern_spacing` option, shown in [Example 6-16](#), to set spacing from the current pattern to other patterns.

Example 6-16 Using the `other_pattern_spacing` Option

```
fill_patterns = {
// First Pattern A
{type = UF_POLYGON,
 polygon_fill = {
  pattern_spec = {
    space_x = 0.1, space_y = 0.1,
    other_pattern_spacing = {1 => >= 2, 2 => >= 1}
  },
  layers = {SHAPE_A}
},
},

// Second Pattern B
{type = UF_POLYGON,
 polygon_fill = {
  pattern_spec = {
    space_x = 0.1, space_y = 0.1,
    other_pattern_spacing = {0 => >= 2, 2 => >= 3}
  },
  layers = {SHAPE_B}
},
},

// Third Pattern C
{type = UF_POLYGON,
 polygon_fill = {
  pattern_spec = {
    space_x = 0.1, space_y = 0.1,
    other_pattern_spacing = {0 => >= 1, 1 => >= 3}
  },
  layers = {SHAPE_C}
},
},
}
```



The following rules and restrictions apply for the spacing options:

- The `allowed_spacing_x` and `allowed_spacing_y` values can be set in both the `layer_spec` and `pattern_spacing` options at the same time.
- The `allowed_spacing_x` and `allowed_spacing_y` values in the `layer_spec` option do not support multiple space ranges. That is, the list can only have one value. Only the `>` and `>=` constraints are supported.
- The `allowed_spacing_x` and `allowed_spacing_y` values in the `pattern_spacing` option support multiple space ranges. The `<` and `<=` constraints are not supported.
- The `allowed_spacing_x`, `allowed_spacing_y` and `min_space_corner` options do not support negative values.
- When the `allowed_spacing_x` and `allowed_spacing_y` values are not specified, valid fill-to-fill spacing is any distance greater than or equal to (`>=`) the `space_x` and `space_y` values. The tool makes sure that patterns or fill layers in adjacent target layer polygons have the minimum spacing of the `space_x` and `space_y` values.
- When the `allowed_spacing_x` and `allowed_spacing_y` values are specified, the tool makes sure that patterns or fill layers in adjacent target layer polygons have the minimum spacing of the `allowed_spacing_x` and `allowed_spacing_y` values.
- When the `fill_context` option is `PARTITION_CONTEXT`, patterns or fill layers in adjacent partitions use the `allowed_spacing_x` and `allowed_spacing_y` values, if specified.
- The `other_pattern_spacing` option does not apply to variations of a pattern when the `type` option is `UF_STACK` or `UF_CELL`. In this case, the minimum space between different variations of a given pattern is based on the `space_x` and `space_y` values.
- By default, the `unified_fill()` function performs corner checking between patterns in different polygons of the target layer that do not share any common projection.
- When the `type` option is `UF_STACK`, the `unified_fill()` function does not apply either the `pattern_spacing` or `fill_to_fill_spacing` option between patterns of different permutations.
- The `space_x` and `space_y` values can be less than or equal to 0 (`<= 0`) only if the `type` option is `UF_STACK` or `UF_CELL`, the `separate_patterns` option is `false`, and the `grouping` option is `NONE`.
- Corner checking, using the `min_space_corner` and `extension` options, is disabled if either the `space_x` or `space_y` option is negative. By default, a corner spacing check between fill patterns in different fill regions is based on `max(space_x, space_y)`.

## Target Region and Fill-to-Signal Spacing

The `unified_fill()` function does not require you to explicitly derive the target region in the runset. The target region is automatically constructed by the tool using the various fill-to-signal spacing constraints. Specify these spacing values using the `fill_to_signal_spacing` option either in the definition of an individual fill layer or in the definition of the fill pattern.

In [Example 6-17](#), the M1\_Fill fill layer is kept away from the aM1 design layer by 0.1  $\mu\text{m}$ . Similarly, the

- Minimum space from the M1\_fill fill layer to the aM1\_fill design layer is 0.5  $\mu\text{m}$
- Minimum space from the M2\_fill fill layer to the aM2 design layer is 0.1  $\mu\text{m}$
- Minimum space from the M2\_fill fill layer to the aM2\_fill design layer is 0.5  $\mu\text{m}$
- Minimum space from the (M1\_fill, M2\_fill) multilayer pattern to BLKG\_1 is 0.3  $\mu\text{m}$
- Minimum space from the (M1\_fill, M2\_fill) multilayer pattern to BLKG\_2 is 0.2  $\mu\text{m}$

### Example 6-17 `fill_to_signal_spacing` Option Example

```
DATA_M1_fill : list of list of coordinate_s = {
  { {1.0, 0.0}, {2.0, 0.0}, {2.0, 3.0}, {1.0, 3.0} }
};
SHAPE_M1_fill : polygon_layer_s = {
  layer_spec = {
    output_layer_key = "M1_fill",
    fill_to_signal_spacing = {
      {aM1, min_space = 0.1},
      {aM1_FILL, min_space = 0.5}
    }
  }
  polygons = DATA_M1_fill,
};

DATA_M2_fill : list of list of coordinate_s = {
  { {0.0, 1.0}, {3.0, 1.0}, {3.0, 2.0}, {0.0, 2.0} }
};
SHAPE_M2_fill : polygon_layer_s = {
  layer_spec = {
    output_layer_key = "M2_fill",
    fill_to_signal_spacing = {
      {aM2, min_space = 0.1},
      {aM2_FILL, min_space = 0.5}
    }
  }
  polygons = DATA_M2_fill,
};

fill_patterns = {{
```

```

    type = UF_POLYGON,
    polygon_fill = {
        layers = {SHAPE_M1_fill,
                  SHAPE_M2_fill},
        fill_to_signal_spacing = {
            {aBLKG_1, min_space = 0.3},
            {aBLKG_2, min_space = 0.2}
        }
    }
}
}}

```

For fill patterns of `UF_POLYGON` type, the multilayer pattern is not broken down to smaller variations of the pattern. As shown in [Figure 6-19](#), the pattern is repeated in the target region by keeping a minimum distance from the design layers as specified in the `fill_to_signal_spacing` option.

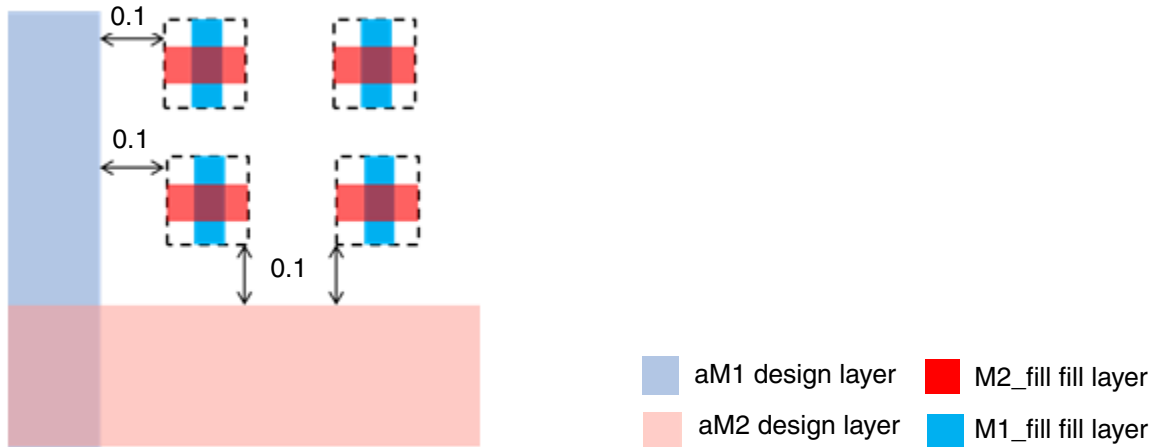
By default, the `unified_fill()` function measures spacing from the edge of the design layer to the bounding box of the pattern. The keepout region for the pattern is constructed using all constraints of the `fill_to_signal_spacing` option defined for this pattern, as well as for every individual layer definition in that pattern.

There are two methods of measuring spacing between fill and signal layers. The `spacing_context` option controls which method of measurement is used for a given fill pattern:

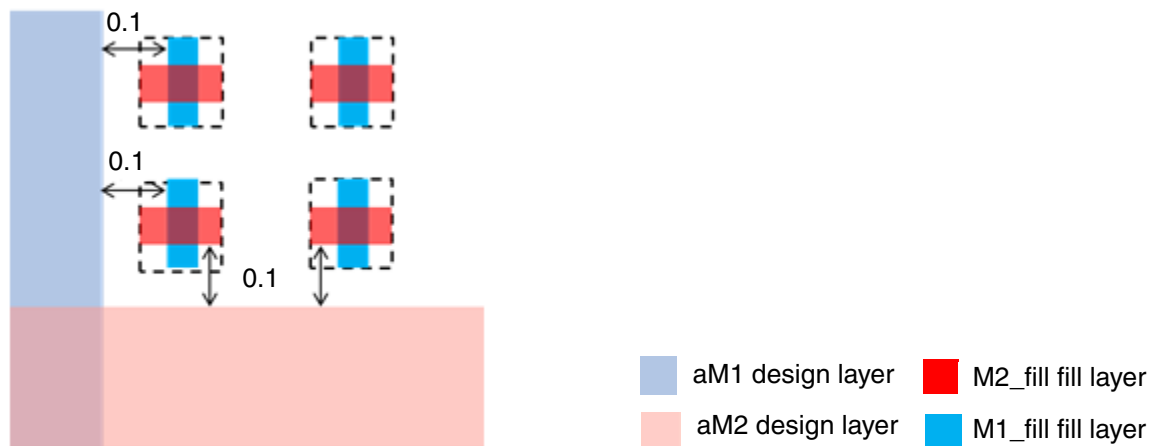
- When set to `UF_PATTERN`, spacing is measured from the signal to the bounding box of a fill pattern.
- When set to `UF_LAYER`, spacing is measured from the signal to individual polygons of a layer inside the pattern.

**Note:**

When the `space_x` and `space_y` values are negative, the `spacing_context` option setting of `UF_PATTERN` is not supported.

Figure 6-19 *spacing\_context = UF\_PATTERN Example*

As shown in [Figure 6-20](#), when the `spacing_context` option is `UF_LAYER`, the `unified_fill()` function measures fill-to-signal spacing from the edge of the design layer to the bounding box of an individual layer definition.

Figure 6-20 *spacing\_context = UF\_LAYER Example*

[Example 6-18](#) and [Figure 6-21](#) illustrate the behavior of the `spacing_context` option.

Example 6-18 *fill\_to\_signal\_spacing in Fill Layer Definition*

```
SHAPE_M1_fill : polygon_layer_s = {
  layer_spec = {
    output_layer_key = "M1_fill",
    fill_to_signal_spacing = {
      {aM1, min_space = 5.0}
    }
  }
  polygons = DATA_ M1_fill,
};
SHAPE_M2_fill : polygon_layer_s = {
```

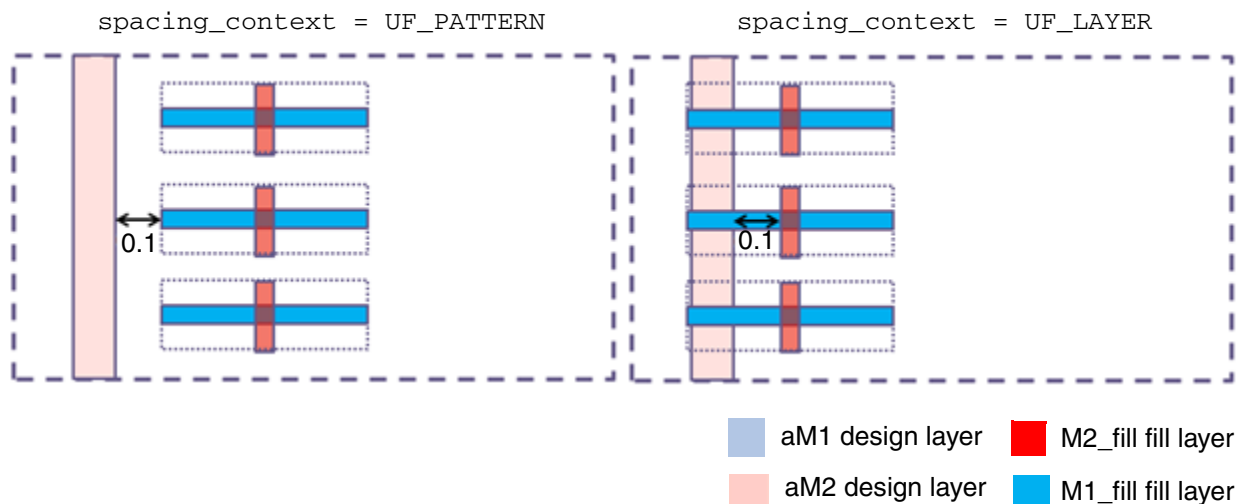
```

layer_spec = {
    output_layer_key = "M2_fill",
    fill_to_signal_spacing = {
        {aM2, min_space = 0.1}
    }
}
polygons = DATA_M2_fill,
};

fill_patterns = {{
    type = UF_POLYGON,
    polygon_fill = {
        layers = {SHAPE_M1_fill,
                  SHAPE_M2_fill}
    }
}}

```

Figure 6-21 *spacing\_context* Behavior for [Example 6-18](#)



For the UF\_POLYGON, UF\_STACK, UF\_CELL, UF\_EXPANDABLE, and UF\_EXPANDABLE\_CELL pattern types, it is possible to specify the fill\_to\_signal\_spacing option at both the layer\_spec level and the pattern level. When the fill\_to\_signal\_spacing option is specified at both levels, the most restrictive value is used. In [Example 6-19](#), the M1\_fill fill layer is kept away from the aM1 design layer by a minimum distance of 0.5  $\mu\text{m}$ .

Example 6-19 *fill\_to\_signal\_spacing* Defined Both in the layer\_spec Option and in the Pattern

```

SHAPE_M1_fill : polygon_layer_s = {
    layer_spec = {
        output_layer_key = "M1_fill",
        fill_to_signal_spacing = {
            {aM1, min_space = 0.5}
        }
    }
}
polygons = DATA_M1_fill,

```

```

};

fill_patterns = {{
    type = UF_POLYGON,
    polygon_fill = {
        layers = {SHAPE_M1_fill},
        fill_to_signal_spacing = {
            {aM1, min_space = 0.1}
        }
    }
}}

```

In [Example 6-20](#), as shown in [Figure 6-22](#), multiple design layers are used with `fill_to_signal_spacing` values at different places in the fill layer definitions and the fill pattern definition.

**Example 6-20** *fill\_to\_signal\_spacing With spacing\_context and Multiple Design Layers*

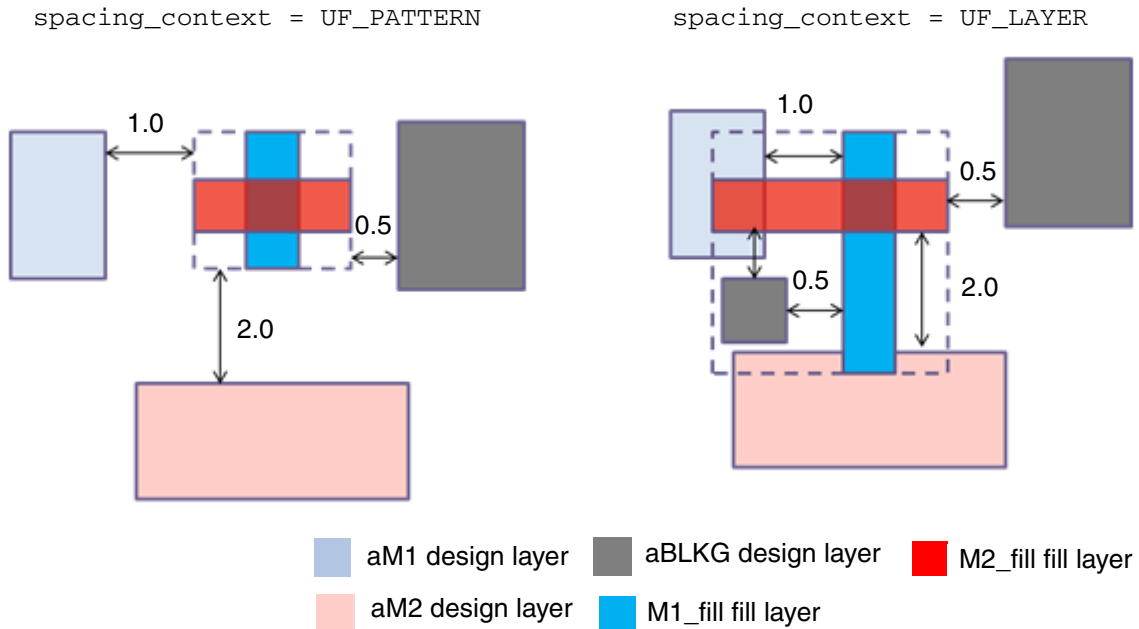
```

SHAPE_M1_fill : polygon_layer_s = {
    layer_spec = {
        output_layer_key = "M1_fill",
        fill_to_signal_spacing = {
            {aM1, min_space = 1.0}
        }
    }
    polygons = DATA_M1_fill,
};

SHAPE_M2_fill : polygon_layer_s = {
    layer_spec = {
        output_layer_key = "M2_fill",
        fill_to_signal_spacing = {
            {aM2, min_space = 2.0}
        }
    }
    polygons = DATA_M2_fill,
};

fill_patterns = {{
    type = UF_POLYGON,
    polygon_fill = {
        layers = {SHAPE_M1_fill,
                  SHAPE_M2_fill},
        fill_to_signal_spacing = {
            {aBLKG, min_space = 5.0}
        }
    }
}}

```

Figure 6-22 *spacing\_context* Behavior for Example 6-20

## Range and Width Dependent Spacing

Apart from a fixed minimum space value in the `fill_to_signal_spacing` option, fill-to-signal spacing ranges from a fill layer to a design layer can also be specified. You can specify a list of lower bound and upper bound constraints to indicate the allowed spacing of the fill layers from the design layer. Fill patterns are placed only within these space constraints. Additionally, a list of spacing constraints can be specified based on the width of the design layer polygons.

In [Example 6-21](#) spacing of the M1\_fill fill layer to the aM1 design layer is

- Minimum space from M1\_fill to aM1:
  - aM1 width > 0.1
  - space = 0.1
- Allowed space from M1\_fill to aM1:
  - aM1 > 0.3
  - $0.2 < \text{space} < 0.5$  or  $\text{space} \geq 1.0$

That is, M1\_fill is not allowed to be placed next to aM1 with  $\text{space} \leq 0.2$  or  $0.5 \leq \text{space} < 1.0$

**Example 6-21 Range and Width Dependent Spacing Example**

```

layer_spec = {
    output_layer_key = "M1_fill",
    fill_to_signal_spacing = {
        {aM1, {
            {width > 0.1, {>= 0.1}},
            {width > 0.3, {(0.2, 0.5), >= 1.0}}
        }},
    }
}
polygons = DATA_M1_fill,
};

fill_patterns = {{
    type = UF_POLYGON,
    polygon_fill = {
        layers = {SHAPE_M1_fill}
    }
}}

```

---

## Polygon Grouping

For `UF_STACK` and `UF_CELL` fill types, the `unified_fill()` function can break the pattern definition into smaller permutations. If a pattern definition has two fill layers, for example, `M1_fill` and `M2_fill`, there can be three permutations:

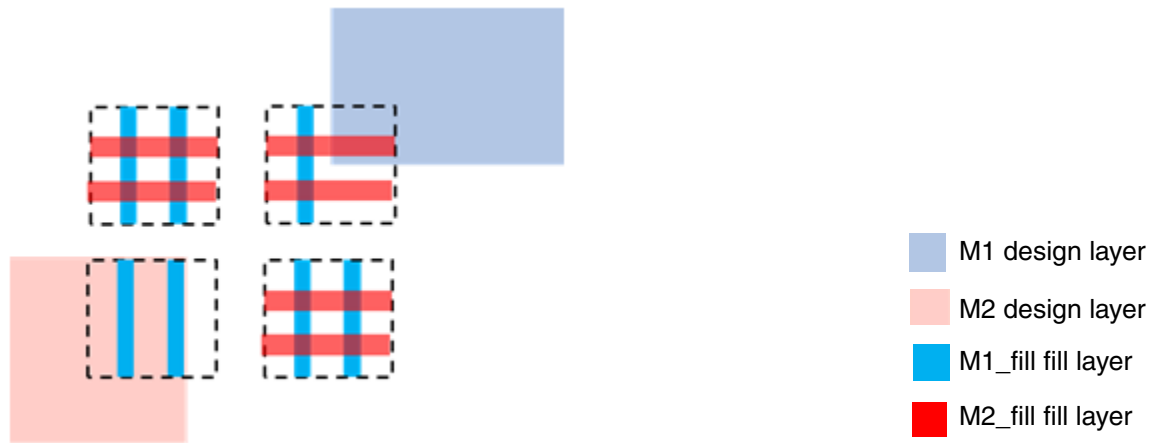
- Complete pattern
- Pattern with only `M1_fill`
- Pattern with only `M2_fill`

When an individual layer definition of a fill layer has multiple polygons, you can configure the `unified_fill()` function to break the layer definition further, so that either all or none of the fill-layer polygons are retained when pattern placements cause spacing violations. When multiple polygons constitute a layer definition in a pattern, the `unified_fill()` function retains, by default, all polygons that meet fill-to-signal spacing in every repetition of the pattern.

As shown in [Figure 6-23](#), a multilayer pattern, indicated by dashed lines, is repeated inside a target region. The `spacing_context` option is `UF_LAYER`. The pattern consists of two fill layers, each composed of two rectangles. Based on fill-to-signal spacing of the fill layer to the design layer, every instance might have either all of the polygons or a partial set of polygons. Polygons causing spacing violations are removed.



Figure 6-23 Multilayer Pattern Repeated Inside a Target Region

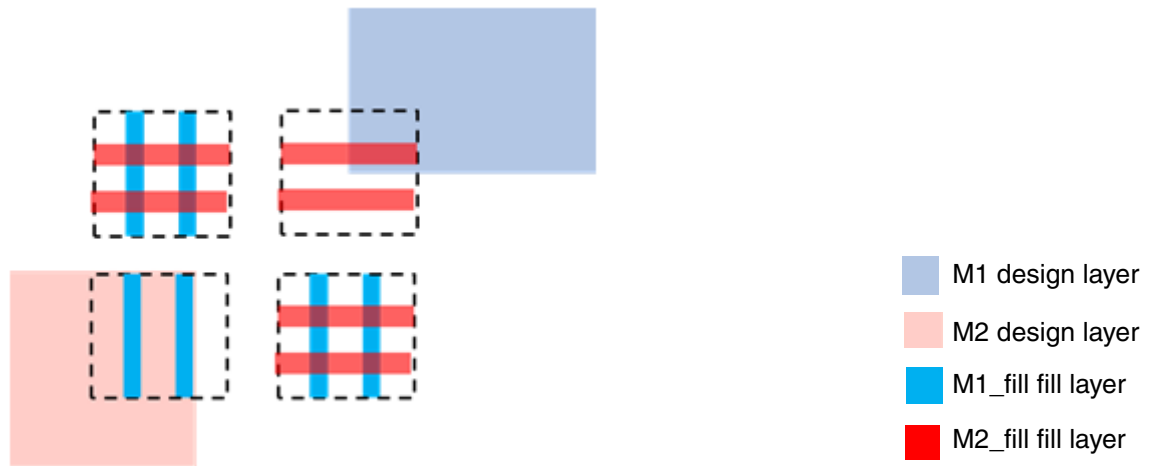


Alternately, you can configure the `unified_fill()` function in such way that if any polygon of a fill layer inside a pattern repetition causes spacing violations, then all polygons of that fill layer are removed from that specific pattern repetition. For this behavior, set the `grouping` option to `ALL` in the definition of the fill layer. For example, as shown in [Example 6-22](#) and [Figure 6-24](#), both polygons of the `M1_fill` fill layer are removed from the top right pattern repetition, even though only one polygon violates the spacing.

**Example 6-22** `grouping = ALL` Example

```
SHAPE_M1_fill : polygon_layer_s = {
  layer_spec = {
    output_layer_key = "M1_fill",
    grouping = ALL,
    fill_to_signal_spacing = {
      {aM1, min_space = 1.0}
    }
  }
  polygons = DATA_ M1_fill,
};
```

Figure 6-24 Layer Removed From Top Right Pattern Repetition Example



The following rules and restrictions apply for polygon grouping:

- If the `spacing_context` option is `UF_PATTERN`, the `grouping` option setting of `NONE` does not apply because fill-to-signal spacing is measured from the bounding box of the pattern but not from the individual polygons. When the `spacing_context` option is `UF_PATTERN`, the `grouping` option is automatically `ALL`.
- When the `space_x` and `space_y` values are negative, the `grouping` option setting of `ALL` is not supported.
- The `grouping` option does not apply to the `UF_POLYGON` and `UF_ADJUSTABLE` types.

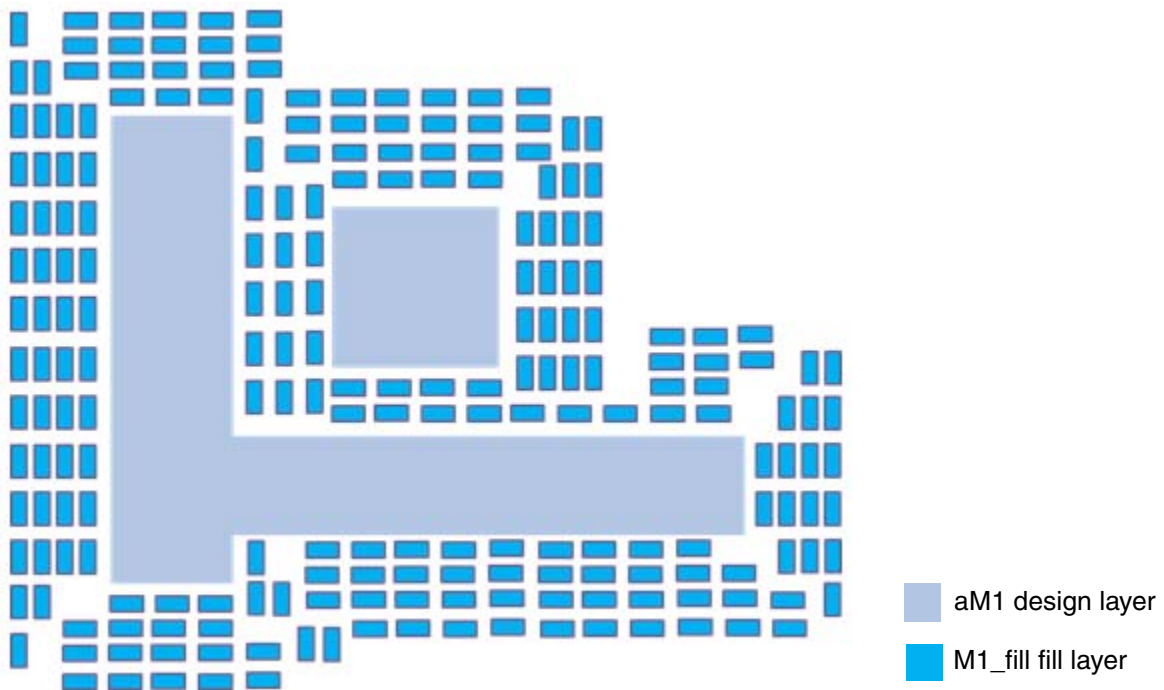
---

## Signal Aligned Fills

For applications that require design layers to be protected by placing fill shapes alongside the design layer polygon, use the `signal` option in the `unified_fill()` function. This feature inserts fill shapes around the design layer shapes in the form of a ring such that the major axis of the fill shape aligns with the edge of the design layer shape. [Figure 6-25](#) shows an example.

*Figure 6-25 Fill with ring\_count = 2*

Use the `signal` option only with the `UF_ADJUSTABLE` and `UF_POLYGON` types. With the `UF_POLYGON` type, the pattern must be a single layer with single shape. You must set the `fill_context` option to `SIGNAL_CONTEXT` in the definition of the `fill_patterns` argument. With the `ring_count` option of the `signal` option, you can control the number of rings of fill pattern shapes that surround each design layer shape. In [Figure 6-25](#), the ring count is 2. [Figure 6-26](#) and [Example 6-23](#) show an example with a ring count of 4.

*Figure 6-26 Fill with ring\_count = 4**Example 6-23 Fill with ring\_count = 4*

```
DATA_M1_fill : list of list of coordinate_s = {
  { {0.0, 0.0}, {2.0, 0.0}, {2.0, 1.0}, {0.0, 1.0} }
};
SHAPE_M1_fill : polygon_layer_s = {
```

```

    layer_spec = {
        output_layer_key = "M1_fill",
        fill_to_signal_spacing = {
            {aM1, min_space = 0.1},
        }
    }
    polygons = DATA_M1_fill,
};

fill_patterns = {{
    type = UF_POLYGON,
    polygon_fill = {
        pattern_spec = {space_x = 0.1,
                        space_y = 0.1},
        layers = {SHAPE_M1_fill},
        fill_context = SIGNAL_CONTEXT,
        signal = {layer = aM1,
                  ring_count = 4}
    }
}}

```

You can use the `signal` option with the width-dependent spacing functionality that is described in the [Range and Width Dependent Spacing](#) section. Use this functionality in applications that require the fill shapes to be kept away from the design layer shape by different minimum space values that depend on the width of the design layer shape. Fill pattern rings are generated using all spacing constraints.

The following rules and restrictions apply for signal aligned fills:

- The `signal` option cannot be used with the `UF_STACK` or `UF_CELL` type.
- The `signal` option cannot be used when pattern definitions have multiple fill layers or if the fill-layer definition has multiple polygons.
- The `stagger_y` option is not supported with the `signal` option. The `stagger_y` option is ignored.
- The `fill_context` option must be `SIGNAL_CONTEXT`. With this setting, the `partition`, `insertion`, and `pitch` options are ignored.
- The `width_based_spacing` option is not supported for the design layer specified in the `signal` option. Use the `min_space` option in `fill_to_signal_spacing` to set the minimum spacing between the fill layer and the design layer specified in the `signal` option.

## Defining the Fill Pattern for Signal Aligned Fills

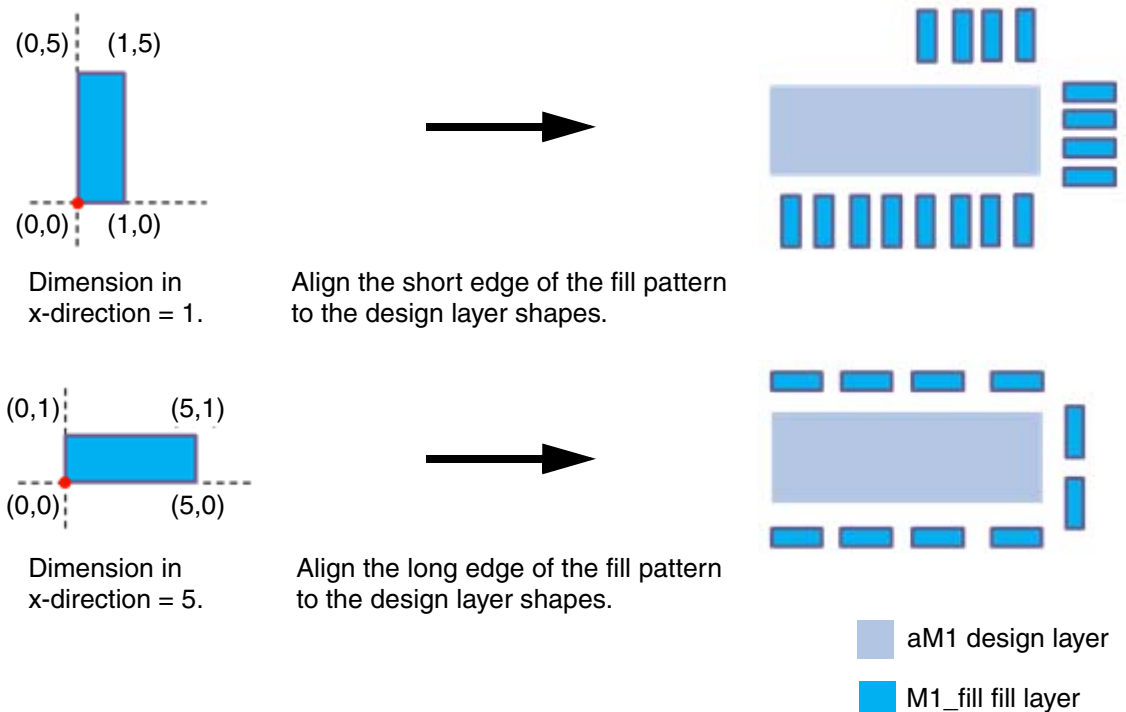
When you define a fill pattern using the coordinate list, make sure that the dimensions of the pattern in the x- and y-directions meet your requirements. The `signal` option in the `unified_fill()` function treats the x-direction as the major axis of the pattern. The tool aligns this edge of the pattern with the shapes on the design layer.

- To align the long edge of the fill pattern rectangle with the design layer shape, make sure that the dimension of the pattern in the x-direction is greater than the dimension in the y-direction.
- To align the short edge of the fill pattern rectangle with the design layer shape, make sure that the dimension of the pattern in the x-direction is less than the dimension in the y-direction.

Figure 6-27 illustrates the two scenarios. For simplicity, not all fill pattern placements are shown. For this example, the coordinate data is:

```
DATA_M1_fill_V : list of list of coordinates_s = {
  { {0.0, 0.0}, {1.0, 0.0}, {1.0, 5.0}, {0.0, 5.0} }
};
```

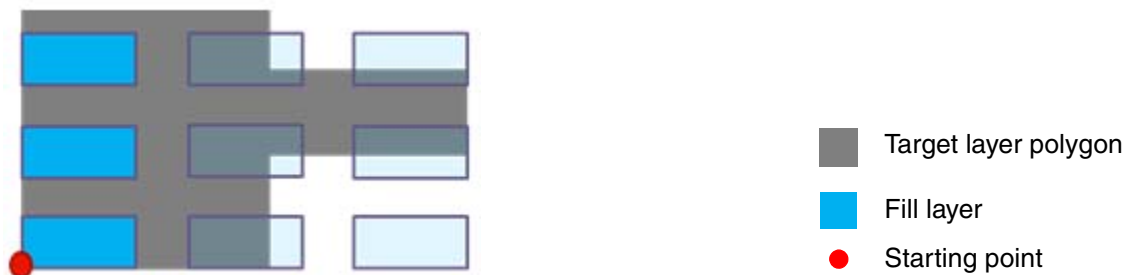
Figure 6-27 Aligning Short or Long Edge of the Fill Pattern



## Improving Pattern Insertion

Similar to the `fill_patterns` argument, the default starting point for pattern placement in the `unified_fill()` function is at the lower-left corner of the bounding box of the polygon on the target layer. When the polygons on the target layer are complex, nonrectangular shapes, the default placement might not always result in maximum insertion. [Figure 6-28](#) shows how several patterns could not be placed in such a situation.

*Figure 6-28 Default Placement That Is Not Maximum Insertion*



The `unified_fill()` function provides several optional configurations to change the default starting point and allow more flexibility in pattern placement. These changes might improve the insertion rate.

[Example 6-24](#) shows how to use the `iteration` option to improve the insertion rate.

*Example 6-24 Using the iteration Option*

```
fill_patterns = {{
    type = UF_POLYGON,
    polygon_fill = {
        pattern_spec = {space_x = 0.1,
                        space_y = 0.1},
        layers = {SHAPE_M1_fill},
        insertion = {iterations = 3}
    }
}}
```

For a detailed description of the `insertion` option, see the `unified_fill()` function in the *IC Validator Reference Manual*.

## Pitch Aligned Fills

In the `fill_pattern` option, you can control the location of the pattern grid with the `grid_mode` option.

- Specify `GLOBAL` to place the fill shapes by using the lower-left of the entire database extents as the reference.

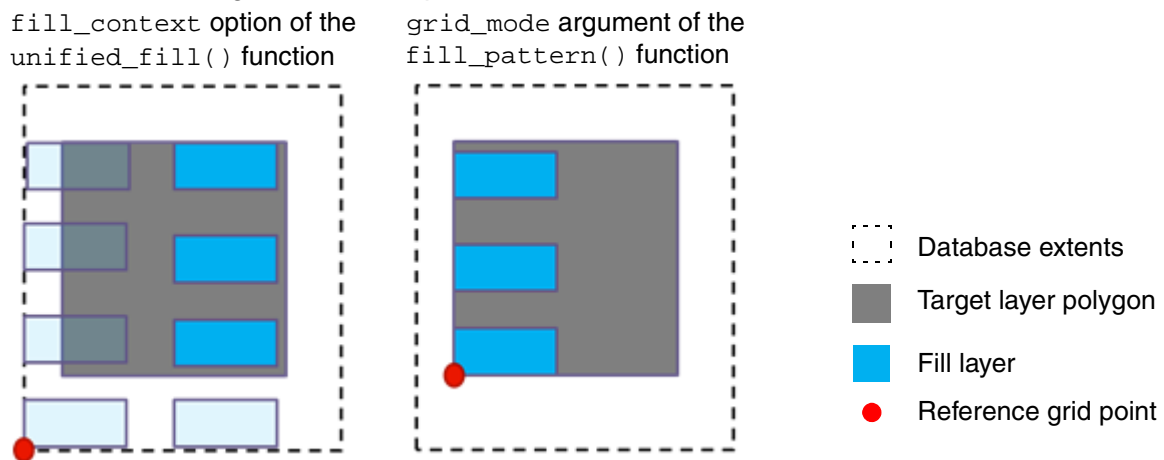
- Specify `LOCAL` to place fill shapes by allowing every polygon on the input layer to have its own unique reference grid point.

The reference grid point is the lower-left of the bounding box of every individual polygon belonging to the input layer.

With the `unified_fill()` function, in addition to the `GLOBAL` and `LOCAL` grid modes, you can enforce more complex control over the location of the pattern grid by referencing another layer. Along with the target layer, you can also specify a context layer. When a context layer is used, fill patterns are placed inside the target layer polygons in such a way that the lower-left vertex of the bounding box of the fill patterns aligns with the grid points originating from the lower-left vertex of the bounding box of each individual polygon on the context layer as the reference grid point. A context layer polygon can encompass multiple polygons on the target layer.

Figure 6-29 shows a comparison of the `grid_mode` argument of the `fill_pattern()` function with the `fill_context` option of the `unified_fill()` function.

Figure 6-29 Pitch Aligned Fills Comparison



Example 6-25 and Figure 6-30 show the behavior of the `pitch` option of the `unified_fill()` function. In this example, the `pitch` option is applied only to the y-direction grid point. The fill patterns in a given target layer polygon are placed starting from the grid point that is closest to the lower-left vertex of the bounding box of the specific target layer polygon, satisfying the specified `pitch` value.

Example 6-25 Applying the Pitch Option Only in the Y-Direction

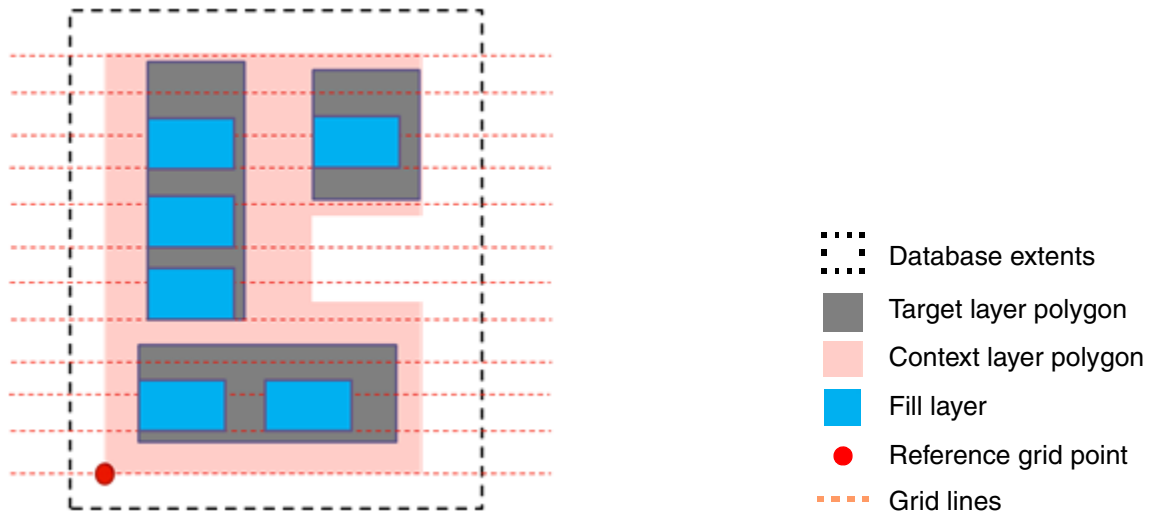
```
fill_patterns = {{
    type = UF_POLYGON,
    polygon_fill = {
        pattern_spec = {space_x = 0.1,
                        space_y = 0.1},
        layers = {SHAPE_M1_fill},
        pitch = {y = 0.1,
```

```

    context_layer = aFILL_CONTEXT_LAYER}
  }
}

```

Figure 6-30 Applying the Pitch Option Only in the Y-Direction



The following rules and restrictions apply for pitch-aligned fills:

- Target layer polygons that are completely outside the context layer are discarded. Fill patterns are not inserted in these polygons.
- If a target layer polygon overhangs context layer polygons, only fill-pattern polygons that are fully enclosed by the context layer are inserted.
- Fill patterns are not inserted in regions of the target layer that interact with multiple polygons on the context layer.
- When the `pitch` option is used, the `fill_context` option is ignored if it is set to `CHIP_CONTEXT` or `REGION_CONTEXT` because these settings are not relevant in this case.
- The `pitch` value must correlate with the pattern dimension and the `space_x` and `space_y` values.

For example, the following value must be a multiple of the `pitch` value; otherwise, the resultant fill might be missing patterns:

$((\text{pattern dimension in x-direction}) + \text{space}_x)$

For a higher insertion rate, you can increase the `shift_factor` value of the `insertion` option. The `unified_fill()` function automatically adjusts the fill patterns inside the target layer polygons such that they align with the grid lines of the context layer polygon. The `shift_factor` option is useful when polygons on the target layer are complex, nonrectangular shapes. [Example 6-26](#) shows how to use the `shift_factor` option.



**Example 6-26 Using the `shift_factor` Option**

```

fill_patterns = {{
    type = UF_POLYGON,
    polygon_fill = {
        pattern_spec = {space_x = 0.1,
                        space_y = 0.1},
        layers = {SHAPE_M1_fill},
        insertion = {shift_factor = 5}
        pitch = {y = 0.1,
                 context_layer = aFILL_CONTEXT_LAYER},
    }
}}

```

---

## Criteria Analysis

With the `unified_fill()` function, you can customize the fill procedure for metrics such as density. The tool supports equation-based criteria analysis to meet requirements such as density ranges, target density, and uniform gradient density across the design. The `unified_fill()` function uses the same algorithms as the `density()` function, ensuring that the fill results meet the postfill design rule requirements.

Use the following arguments of the `unified_fill()` function to configure the fill procedure to meet your requirements:

```

criteria
delta_window
delta_x
delta_y
window_layer
boundary

```

The `criteria` option supports the following options to configure the fill procedure:

```

target
gradient
fill_layer_keys
design_layers
design_layers_hash
window_function

```

The string names specified in the `output_layer_key` option of the pattern definition are used to identify the fill layers used in criteria equations. You can specify any number of criteria equations in the `criteria` argument. If you have multiple criteria equations for the same fill layer, the tool ensures that at least one of the equations is met. If there are multiple criteria equations for different fill layers (for example, `aM1_FILL` and `aM2_FILL` each have a criteria equation defined), then the tool addresses all the equations.

When the `criteria` argument is specified without a window function, the tool evaluates a default equation to check against the criteria value. This default equation is evaluated for every delta window and checked against the `target` and `gradient` values of the `criteria` argument. The fill patterns are evaluated to converge the evaluated value to the specified value. The default equation is:

$$[(\text{Area of fill layers}) + (\text{Area of design layers})] / (\text{Area of delta window})$$

Where,

- *Area of fill layers* is the sum of areas within the current delta window of all polygon data on all the fill layers that are specified in the `fill_layer_keys` option.
- *Area of design layers* is the sum of areas within the current delta window of all polygon data on all the design layers that are specified in the `design_layers` option.

Note:

Pattern definitions must generate fill output that allows the criteria to contribute. For example, the tool cannot meet a higher target density when the provided pattern definitions are composed of narrow, widely-spaced rectangles.

[Example 6-27](#) shows a fill specification for two fill layers, `M1_fill` and `M2_fill`, that have their own criteria equations. The cumulative density of the existing `aM1` design layer and `M1_fill` fill layer should be between 25 percent and 45 percent with a target of 35 percent, when measured in 100  $\mu\text{m}$  x 100  $\mu\text{m}$  windows stepped across the design in increments of 50  $\mu\text{m}$  along the x- and y-directions. The `M2_fill` fill layer has a similar fill specification with the same density values.

#### *Example 6-27 Fill Specification with Two Criteria Equations*

```
DATA_M1_fill : list of list of coordinate_s = {
  { {1.0, 0.0}, {2.0, 0.0}, {2.0, 3.0}, {1.0, 3.0} }
};

SHAPE_M1_fill : polygon_layer_s = {
  layer_spec = {output_layer_key = "M1_fill"}
  polygons = DATA_M1_fill,
};

DATA_M2_fill : list of list of coordinate_s = {
  { {0.0, 1.0}, {3.0, 1.0}, {3.0, 2.0}, {0.0, 2.0} },
};

SHAPE_M2_fill : polygon_layer_s = {
  layer_spec = {output_layer_key = "M2_fill"}
  polygons = DATA_M2_fill,
};

fill_patterns_m1_m2 = {{
  type = UF_POLYGON,
  polygon_fill = {
```

```

        layers = {SHAPE_M1_fill,
                  SHAPE_M2_fill}
    }
}}

M1_density_criteria : fill_criteria_s =
{
    target = {range = [0.25, 0.45], target = 0.35},
    fill_layer_keys = {"M1_fill"},
    design_layers = {aM1}
};

M2_density_criteria : fill_criteria_s =
{
    target = {range = [0.25, 0.45], target = 0.35},
    fill_layer_keys = {"M2_fill"},
    design_layers = {aM2}
};

uf_fill_output_h = unified_fill(
    fill_patterns = fill_patterns_m1_m2,
    criteria = {M1_density_criteria,
               M2_density_criteria},
    delta_window = {100, 100},
    delta_x = 50,
    delta_y = 50,
    fill_boundary = {type = LAYER, layer = data_extent},
    window_layer = aWINDOW,
    grid = 0.001,
    boundary = ALIGN
);

M1_fill_output = uf_fill_output_h["M1_fill"][0];
M2_fill_output = uf_fill_output_h["M2_fill"][0];

```

[Example 6-28](#) is another fill example that uses multiple design layers and fill layers in the same criteria equation. The sum of the densities in every window of M1, M2, and M3 for both design and fill should be greater than or equal to 105 percent.

Note:

For simplicity, [Example 6-28](#) shows only the criteria equations.

#### *Example 6-28 Fill Specification with Multiple Layers in the Same Criteria Equation*

```

M1_M2_density_criteria : fill_criteria_s =
{
    target = {range = >= 1.05},
    fill_layer_keys = {"M1_fill", "M2_fill", "M3_fill"},
    design_layers = {aM1, aM2, aM3}
};

```

For a specific fill layer, when there is more than one fill pattern specified, the `unified_fill()` function sequentially inserts the fill patterns in the order that the patterns are declared in the function. When a criteria is specified, it is possible that the `unified_fill()` function does not make use of all the fill pattern definitions. That is, the

`unified_fill()` function stops inserting fill as soon as the criteria is met regardless of whether all the fill patterns have been used.

In [Example 6-29](#), the procedure for the M1\_fill fill layer is composed of two insertion sequences. After inserting the first pattern, if the criteria is met, the tool does not perform the second sequence.

#### *Example 6-29 Sequence of Pattern Definitions*

```
fill_patterns_m1 = {
{
    type = UF_POLYGON,
    polygon_fill = {layers = {SHAPE_M1_fill_TYPE1}}
},
{
    type = UF_POLYGON,
    polygon_fill = {layers = {SHAPE_M1_fill_TYPE2}}
}}
M1_density_criteria : fill_criteria_s =
{
    target = {range = [0.25, 0.45], target = 0.35},
    fill_layer_keys = {"M1_fill"},
    design_layers = {aM1}
};
uf_fill_output_h = unified_fill(
    fill_patterns = fill_patterns_m1,
    criteria = {M1_density_criteria},
    delta_window = {100, 100},
    delta_x = 50,
    delta_y = 50,
    fill_boundary = {type = LAYER, layer = data_extent},
    window_layer = aWINDOW,
    grid = 0.001,
    boundary = ALIGN
);
M1_fill_output = uf_fill_output_h["M1_fill"][0];
```

Instead of using the default criteria equation, you can specify your own criteria by defining the window function as shown in [Example 6-30](#).

#### *Example 6-30 Customized Criteria Equation*

```
design_layers : layer_groups_h = {
    "aM1" => aM1
};
M1_density_criteria_func : function ( void ) returning void
{
    da_aM1 = uf_area(DSIGN, "aM1");
    fa = uf_area(FILL, "M1_fill");

    check_value : double = (da_aM1 + fa) / uf_window_area();
    uf_save_window(check_value);
};
```

```

Ml_density_criteria : fill_criteria_s =
{
  target = {[0.25, 0.45]},
  fill_layer_keys = {"Ml_fill"},
  design_layers_hash = design_layers,
  window_function = Ml_density_criteria_func
};

```

See the [Unified Fill Utility Functions](#) section in the "Utility Functions" chapter of the *IC Validator Reference Manual* for more information about the utility functions you can use in the window function.

The following rules and restrictions apply:

- Multiple fill layers cannot be used in a single window function
- If target is not specified, the `unified_fill()` function uses the midpoint of the range as its internal target
- A valid constraint that converges must be specified. For example, the following constraint is not allowed

```
target = {range = <0.50}
```

The following examples show the behavior based for several scenarios:

- `target = {range = >0.50}`

The `unified_fill()` function tries to achieve a target density greater than 50 percent.

- `target = {range = [0.45, 0.55]}`

The `unified_fill()` function assumes an internal target density of 50 percent. Fill patterns are not added to those windows for which the initial density is already greater than 50 percent. Otherwise, the tool works toward reaching a target density of 50 percent, but not exceeding 50 percent.

- `target = {range = [0.45, 0.55], target = 0.48}`

The `unified_fill()` function works toward achieving a target density of 48 percent. If the initial density in a window is 46 percent, the tool works toward reaching a density 48 percent, but not exceeding 48 percent.

- `target = {range = >0.30, target = 0.50}`

The allowed range is greater than 30 percent and less than or equal to 100 percent. The tool works toward achieving a target density of 50 percent, but not exceeding 50 percent.

## Output Layers

The `unified_fill()` function returns a hash containing layer lists, with each list having three polygon layers:

Index	Polygon layer
0	Uncolored
1	Color 1 (The <code>color</code> option must be <code>true</code> to get this layer.)
2	Color 2 (The <code>color</code> option must be <code>true</code> to get this layer.)

To retrieve a list of polygon layers, you specify the `output_layer_key` value that was used by the `layer_spec` option of the fill pattern definition. If the `color` option is `false`, only uncolored data is generated on index 0. If the `color` option is `true`, then the two colored output layers are returned on indexes 1 and 2, respectively. For example,

```
M1_fill_ALL = M1_fill_h["M1_fill"][0];
M1_fill_COLOR1 = M1_fill_h["M1_fill"][1];
M1_fill_COLOR2 = M1_fill_h["M1_fill"][2];
```

[Example 6-31](#) is a procedure for two fill layers, `M1_fill` and `M2_fill`, that shows how the fill layers are accessed from the output hash generated by the `unified_fill()` function.

### Example 6-31 Accessing Fill Layers From Hash

```
fill_patterns_m1_m2 = {{
    type = UF_POLYGON,
    polygon_fill = {
        layers = {SHAPE_M1_fill,      //output_layer_key is "M1_fill"
                  SHAPE_M2_fill}     //output_layer_key is "M2_fill"
    }
}}
uf_fill_output_h = unified_fill(
    fill_patterns = fill_patterns_m1_m2
);

M1_fill_output = uf_fill_output_h["M1_fill"][0];
M2_fill_output = uf_fill_output_h["M2_fill"][0];
```

[Example 6-32](#) shows a fill specification for the `M1_fill` fill layer that has two pattern insertion steps. Each pattern definition has a specified color. The `unified_fill()` function creates fill shapes in each pattern insertion step by coloring them as individual sets. Furthermore, both pattern insertion steps use the same string name in the `output_layer_key` option to designate the fill layer. The `unified_fill()` function accumulates all polygon data from different pattern definitions into the same polygon layer list element in the output hash key of that fill layer. In this example, the value of list index 0 of the `M1_fill` output hash key has

complete polygon data from both pattern insertion steps. The value of list index 1 has complete color 1 polygon data from both pattern insertion steps. Similarly, the value of list index 2 has complete color 2 polygon data from both pattern insertion steps.

**Example 6-32 Pattern Insertion Using the color Option**

```
SHAPE_M1_fill_TYPE1 : polygon_pattern_s = {
    layer_spec = {
        output_layer_key = "M1_fill",
    }
    polygons = DATA_M1_fill_TYPE1,
};
SHAPE_M1_fill_TYPE2 : polygon_pattern_s = {
    layer_spec = {
        output_layer_key = "M1_fill",
    }
    polygons = DATA_M1_fill_TYPE2,
};
M1_fill_h = unified_fill(
    fill_patterns = {
        {type = UF_POLYGON,
         polygon_fill = {
             layers = {SHAPE_M1_fill_TYPE1},
             color = true
         }
        },
        {type = UF_POLYGON,
         polygon_fill = {
             layers = {SHAPE_M1_fill_TYPE2},
             color = true
         }
        }
    },
    fill_boundary = {type = LAYER, layer = data_extent}
);

M1_fill_ALL      = M1_fill_h["M1_fill"][0];
M1_fill_COLOR1   = M1_fill_h["M1_fill"][1];
M1_fill_COLOR2   = M1_fill_h["M1_fill"][2];
```

Apart from the fill layers, you can also retrieve the bounding box, that is, the extents, of one or more output fill layers by specifying one or more of the appropriate `output_layer_key` values. [Example 6-33](#) and [Figure 6-31](#) show the usage of the `extents_output` option.

**Example 6-33 Using the extents\_output Option**

```
SHAPE_M1_fill : polygon_pattern_s = {
    layer_spec = {
        output_layer_key = "M1_fill",
    }
    polygons = DATA_M1_fill_TYPE1,
};
```

```

M1_fill_h = unified_fill(
    fill_patterns = [{
        {type = UF_POLYGON,
         polygon_fill = {
             layers = {SHAPE_M1_fill},
             color = true
         }
    }
}],

    extents_output = {
        {output_layer_key = "M1_fill_boundary",
         fill_layer_keys = {"M1_fill"}}
    }
);

M1_fill      = M1_fill_h["M1_fill"][0];
M1_fill_E    = M1_fill_h["M1_fill_boundary"][0];

```

**Figure 6-31** Using the `extents_output` Option



When the `output_layer_key` option is not specified in the `extents_output` option, the bounding box of the pattern is constructed by considering all the fill layers present in the pattern placement. For patterns that are generated by the `UF_STACK` or `UF_CELL` type, the `extents_output` option works in accordance with the `separate_patterns` option.

## Layer Compression

The geometry data on output polygon layers generated by the `unified_fill()` function can be grouped hierarchically to reduce the size of the output file created by functions such as `write_gds()` and `write_milkyway()`. Layer compression can be achieved in two ways in an IC Validator runset:

- Specify the `hierarchical_fill` option inside pattern definitions of the `unified_fill()` function.



- Use the `compress_fill` option in the `write_gds()` and `write_milkyway()` functions.

The `hierarchical_fill` option in the `unified_fill()` function is similar to the `output_aref` option in the `fill_pattern()` function. However, for optimum output file size, use the `compress_fill` argument in the `write_gds()` and `write_milkyway()` functions for all the individual polygon layers.



# 7

## Working With Edges

---

*This chapter gives a brief overview of using edges in IC Validator runsets. The edge functions are described in the [IC Validator Reference Manual](#).*

The IC Validator tool provides for

- [Associating Edge Layers With a Parent Layer](#)
- [Topological Operations](#)
- [Logical Operations](#)
- [Edge Manipulation](#)

---

## Associating Edge Layers With a Parent Layer

Edge layers can have a parent polygon layer.

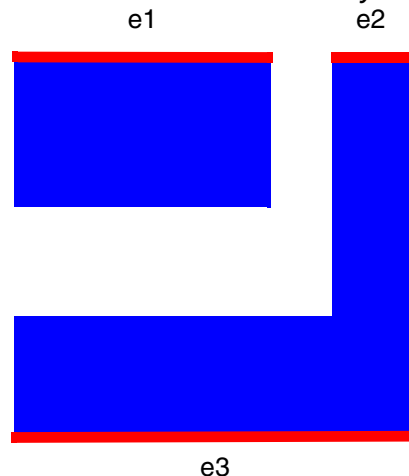
- All edge selection operations maintain the connection between an edge layer and the polygon layer from which it was derived.
- All edge creation operations result in “orphan” edge layers, that is, edge layers without a parent polygon layer.
- The parent polygon layer provides a more intuitive output by influencing dimensional function behavior for edges.

For example, [Figure 7-1](#) shows edge association with a parent layer.

```
red = edge_derived_from_blue;  
@ "red edge min width rule";  
internall(red, spacing < x);
```

The IC Validator tool detects the violation from e2 to e3 and avoids the violation from e1 to e3.

*Figure 7-1 Edge Association With a Parent Layer*



---

## Topological Operations

Topological operations use the complete edges selected and do not add new endpoints. These operations include

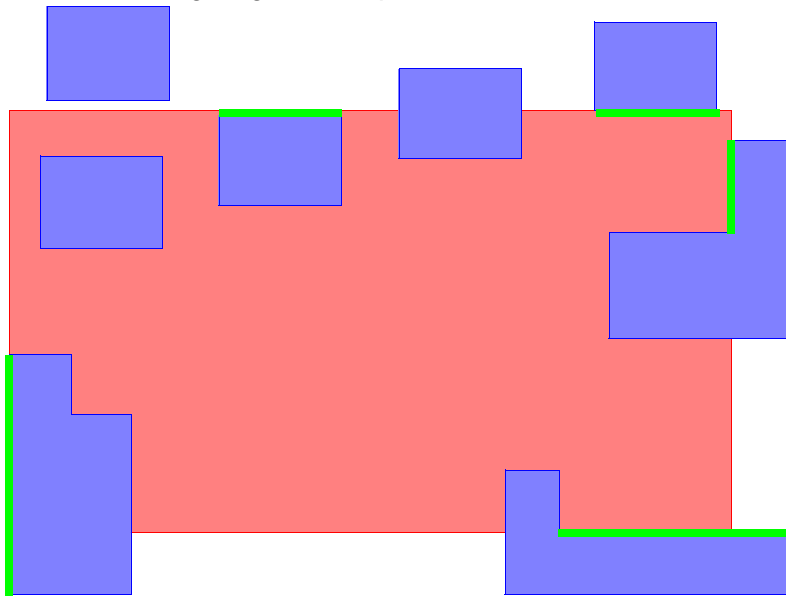
- `adjacent_edge()` and `not_adjacent_edge()`
- `angle_edge()` and `not_angle_edge()`

- `inside_point_touching_edge()` **and** `not_inside_point_touching_edge()`
- `inside_touching_edge()` **and** `not_inside_touching_edge()`
- `interacting_edge()` **and** `not_interacting_edge()`
- `length_edge()` **and** `not_length_edge()`
- `outside_point_touching_edge()` **and** `not_outside_point_touching_edge()`
- `outside_touching_edge()` **and** `not_outside_touching_edge()`
- `point_touching_edge()` **and** `not_point_touching_edge()`
- `touching_edge()` **and** `not_touching_edge()`
- `vertices_edge()` **and** `not_vertices_edge()`

**Figure 7-2** shows an example of touching edges:

```
green = touching_edge (layer1 = blue, layer2 = red);
```

*Figure 7-2 Touching Edges Example*




---

## Logical Operations

Logical operations generate new endpoints. These operations include

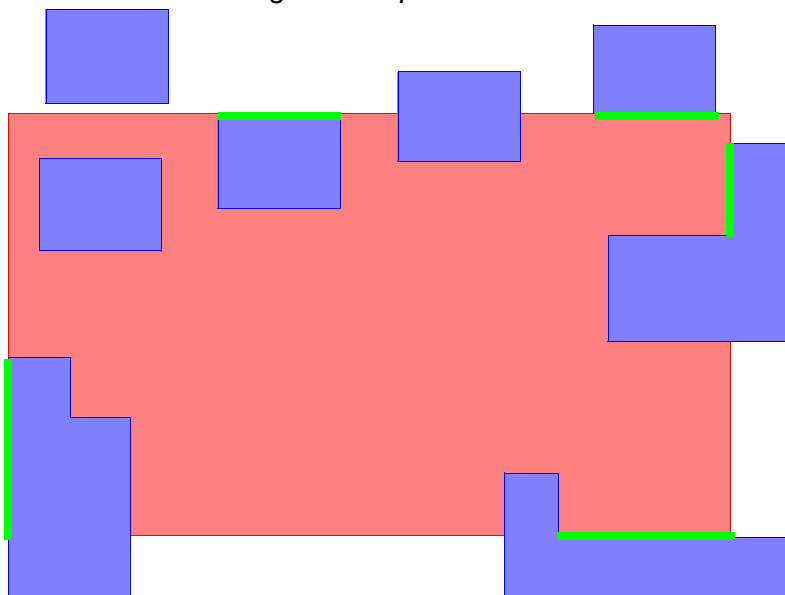
- `coincident_edge()` **and** `not_coincident_edge()`
- `coincident_inside_edge()` **and** `not_coincident_inside_edge()`

- `coincident_outside_edge()` **and** `not_coincident_outside_edge()`
- `and_edge()`
- `not_edge()`
- `or_edge()`
- `xor_edge()`

Figure 7-3 shows an example of coincident edges:

```
green = coincident_edge (layer1 = blue, layer2 = red);
```

*Figure 7-3 Coincident Edges Example*




---

## Edge Manipulation

These functions create edges, which are always orphan edges:

- `move_edge()`
- `snap_edge()`
- `extend_edge()`
- `copy_edge()`

This function creates polygons from edges:

- `edge_size()`

# 8

## DRC Error Classification

---

*This chapter explains the DRC error classification capabilities of the IC Validator tool and how to use this feature.*

The DRC error classification capabilities are described in the following sections:

- [Using DRC Error Classification](#)
- [DRC Error Classification Flow](#)

---

## Using DRC Error Classification

DRC error classification allows you to classify and annotate errors found during an IC Validator run and to export the errors into a database for use in subsequent IC Validator runs. Errors can be classified individually, in groups using VUE, or in bulk using the `pydb_report` utility.

Error classifications and user comments can be retained across multiple IC Validator runs by using a persistent error classification database (cPYDB). Error data from the subsequent run, as well as the classification data, is stored in SQLite databases.

For this flow, use the `error_options()` function. This function has arguments that are specifically for DRC error classification:

- `match_errors`: Specifies the list of error classification databases to import for matching during an IC Validator run. For each error classification database listed, the suppression of matched errors and reporting of unmatched errors can be controlled. (Unmatched errors are errors that exist in an imported error classification database but are not produced by the current IC Validator run.)
- `mustfix_errors`. Designates certain violation comments and cells as must-fix. You cannot classify these comments and cells as Ignore or Waive. This setting applies for the current run and is not retained in the error classification database.
- `comment_match_delimiter`. Specifies a delimiter string for violation comment matching in imported error classification databases.
- `match_errors_processing_mode`. Specifies whether the IC Validator tool uses cell level or hierarchical processing to match classified violations in an error classification database. The default is `CELL_LEVEL.violation_options`. Allows certain global error options to be overridden on a rule by rule basis. For example, if most rules are matched in `HIERARCHICAL` mode, but a few need to run in `CELL_LEVEL` mode, this argument allows for that.

Use the `pydb_report` and `pydb_export` utilities to classify errors, and then reuse the classification data. See “[pydb\\_report Utility](#)” and “[pydb\\_export Utility](#)” for more information. These utilities are in the IC Validator installation bin directory. You can also use VUE to classify and export errors. See the [IC Validator VUE User Guide](#).

---

## Hierarchy Considerations in CELL\_LEVEL Mode

During preprocessing, the IC Validator function makes hierarchy optimizations. As a result, when error matching is done in `CELL_LEVEL` mode, errors could possibly be inconsistent between runs. To help prevent inconsistent hierarchy between runs, when an error classification database is imported, any cells defined within an error classification database



are automatically protected from explosion during preprocessing. However, some optimizations create hierarchy, or explode or flatten individual placements of cells, and therefore, can create consistency problems.

When running the DRC error classification flow in `CELL_LEVEL` mode, always set the `flow` argument in the `hierarchy_auto_options()` function to the `ERROR_CLASSIFICATION` option.

See the [hierarchy\\_auto\\_options\(\)](#) function.

---

## Hierarchical DRC Error Classification

The IC Validator tool provides a `HIERARCHICAL` mode for error classification matching that is more robust and less sensitive to hierarchy changes than `CELL_LEVEL` mode. This mode puts no restrictions on hierarchy optimization. Cells that have classified errors in the error classification database (cPYDB) are allowed to explode.

An error is considered a match for a classified error from the cPYDB if the complete hierarchically formed shape of the errors produced by the IC Validator tool exactly overlaps the complete hierarchically formed shape of errors from the cPYDB with the same classification. If any part of the complete shape is different between the errors found by the IC Validator tool and the errors from the cPYDB, it is not a hierarchical match.

However, in this mode, errors can also be matched individually when possible. That is, when a single error, rather than the complete hierarchically formed shape consisting of multiple errors, exactly overlaps a corresponding single error in the cPYDB it is considered a match, even if it is at a different level of hierarchy.

---

## Error Classification Best Practices

It is important to make the distinction between classified errors in the error classification database (cPYDB) that are referenced by rule, or violation comment, and errors produced by the IC Validator tool that are created by a command. The recommended best practice to ensure proper error matching is to output all of the violations for a rule with a single IC Validator command. This avoids the possibility of errors from different commands within the same rule interacting in the cPYDB and preventing a hierarchical match when they are considered for an individual command in a subsequent run. Violation comments should be made unique to the command to ensure this relationship.

It is also important to clean up invalid cPYDB data. For example if the error shapes change, but the new shapes are considered valid and are to be waived, the cPYDB entries for that violation and cell should be purged and recreated with the new error shapes. Leaving the old, invalid shapes in the cPYDB can prevent the IC Validator tool from finding a hierarchical match.

Currently, error classification is most applicable to DRC violations. Matching of previously classified errors of other types of violations, including ERC, device extraction, and density is not supported.

---

## Database Naming Conventions

Databases are referred to by specifying a database name and database path. Error databases (PYDBs) generated by an IC Validator run are named `PYDB_top-cell` and by default, reside in `run_details/pydb` unless a different path is specified in the `db_path` argument of the `error_options()` function.

For example, you run the IC Validator tool on the design TOP. The directory `run_details/pydb/PYDB_TOP` contains the data files for the error database. The database path is `run_details/pydb`, and the database name is `PYDB_TOP`.

SQLite allows database names to consist of alphanumeric characters and the underscore character (`_`). Database names are transformed, as follows:

- All non alphanumeric characters are replaced with an underscore (`_`). For example, `PYDB_A.1.2` becomes `PYDB_A_1_2`.
- If the database name contains only digits, leading and trailing underscores are added. For example, `123` becomes `_123_`.

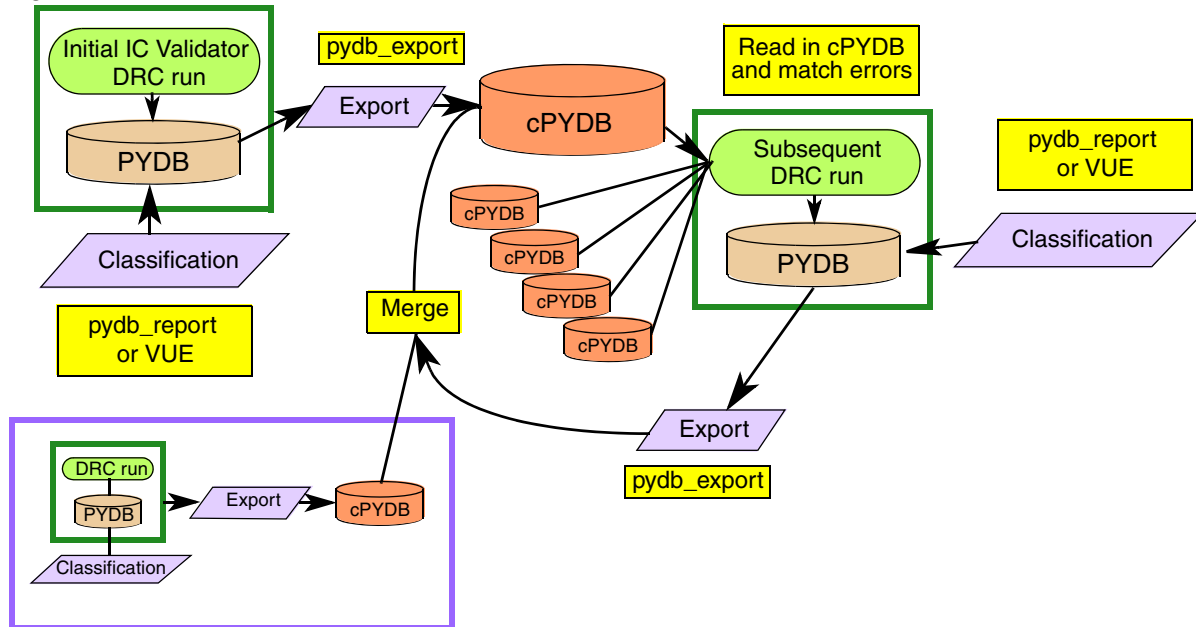
You can continue to use the original database name in runsets, in VUE, and with the `pydb_report` and `pydb_export` utilities, but the database is stored on disk with the modified name.

---

## DRC Error Classification Flow

[Figure 8-1](#) shows the DRC error classification flow.

Figure 8-1 DRC Error Classification Flow



1. Run the IC Validator tool on the design. The resultant DRC errors can then be classified using VUE or the pydb\_report utility.

For reuse of classification data in future runs, export the classified errors to an error classification database using VUE or the pydb\_export utility.

2. When the IC Validator tool is run again, specify the error classification database in the `match_errors` argument of the `error_options()` function. If multiple error classification databases are specified, each one is searched for a matching error in the order specified. The error is pre-classified with the data in the error classification database it matches first, and it is written to the error database for the current run. The error shows as classified in the LAYOUT\_ERRORS file and in VUE.

Using the `match_errors` argument of the `error_options()` function set to the `suppress_matched_errors` option you can specify errors that are not written to the error database.

**Note:**

The `error_limit_per_check` argument of the `error_options()` function only applies to new errors. All errors produced during this run are checked against the imported error classification database list, and all matching errors are written to error database. Errors that do not match an error classification database are written until the specified `error_limit_per_check` limit is reached.

3. Finally, new classifications from the current run or from other runs can be merged into an existing error classification database. Newly classified errors are appended to the database, while existing errors are updated if they have changed.

---

## DRC Error Classification Flow Example

This section walks through an example of the DRC error classification flow using the AD4FUL design. The pydb\_report and pydb\_export utilities are used for classification and database management. (See [“pydb\\_report Utility” on page 8-13](#) and [“pydb\\_export Utility” on page 8-17](#).)

The basic steps are

1. [Running the IC Validator Tool Initially](#)
2. [Creating an Error Classification Database](#)
3. [Running the IC Validator Tool Again](#)
4. [Updating the Existing Error Classification Database](#)

### Running the IC Validator Tool Initially

First, run the IC Validator tool on the design, producing DRC errors. A standard LAYOUT\_ERRORS file is written.

Next, classify

- All errors in cell INVH as Ignore.
- All errors in the violation “Met1 Spacing must be >= 3” as Waive.
- Errors within a coordinate window in the cell TGATE as Watch.

This classification is accomplished by using the pydb\_report utility.

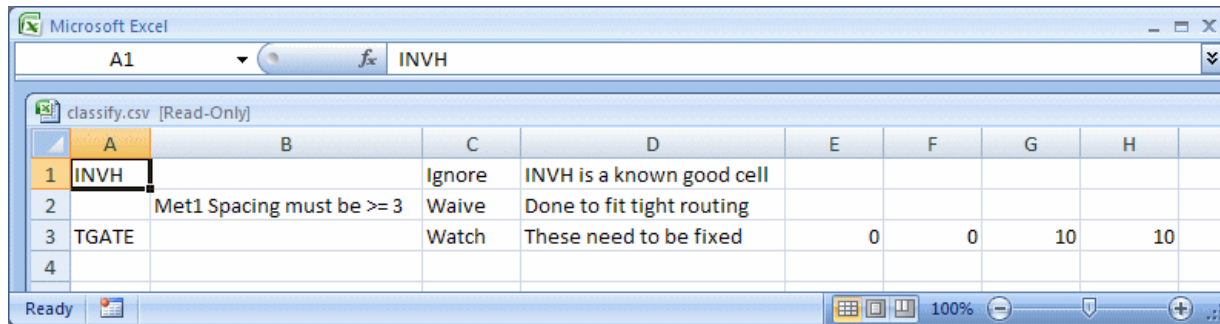
```
pydb_report -pydb_name PYDB_AD4FUL \
  -pydb_path ./run_details/pydb -classify classify.csv
```

The input file to the pydb\_report utility is in a standard comma-delimited format. See [“pydb\\_report Utility” on page 8-13](#) for more information. The file used in this example is

```
INVH,,Ignore,INVH is a known good cell,,,,
,Met1 Spacing must be >= 3,Waive,Done to fit tight routing,,,,
TGATE,,Watch,These need to be fixed,0,0,10,10
```

[Figure 8-2](#) shows the comma-delimited format in Microsoft Excel.

Figure 8-2 Example Classify File in Microsoft Excel



	A	B	C	D	E	F	G	H
1	INVH		Ignore	INVH is a known good cell				
2		Met1 Spacing must be >= 3	Waive	Done to fit tight routing				
3	TGATE		Watch	These need to be fixed	0	0	10	10
4								

The pydb\_report utility writes a log file, pydbreport.log, to the current working directory. The same information is displayed on screen. The output from this example is shown in [Example 8-1](#).

#### Example 8-1 Initial pydb\_report Utility pydbreport.log

```
Connecting to DB: PYDB_AD4FUL
      path: ./run_details/pydb
Connected to DB PYDB_AD4FUL

Error database PYDB_AD4FUL summary
  Number of commands:          12
  Number of commands w/ errors: 12
  Number of dynamic tables:     5
  Total error polygons:        48
  DBU:                          0.00050
  Number of unique cells w/ errors: 6

Classifying Errors from classify.csv...
Classifying all errors in violation/cell...
  Violation: ALL
  Cell:      INVH
  Class:     Ignore
  Comment:   INVH is a known good cell
6 new errors classified.
Classify time = 0:00:00  User=0.00 Sys=0.01 Mem=0.413
Classifying all errors in violation/cell...
  Violation: Met1 Spacing must be >= 3
  Cell:      ALL
  Class:     Waive
  Comment:   Done to fit tight routing
3 new errors classified.
3 existing errors reclassified.
Classify time = 0:00:00  User=0.00 Sys=0.00 Mem=0.491
Classifying all errors in window...
  Violation: ALL
  Cell:      TGATE
  Window:    (0.0000, 0.0000) (10.0000, 10.0000)
  Class:     Watch
```

```

    Comment:   These need to be fixed
    12 new errors classified.
    Classify time = 0:00:00   User=0.01 Sys=0.00 Mem=0.632
    Overall Classify time = 0:00:00   User=0.01 Sys=0.01 Mem=0.632
    Done.

```

You can get a report of the classified errors in the error database for the current run using the `pydb_report` utility:

```

pydb_report -pydb_name PYDB_AD4FUL \
  -pydb_path ./run_details/pydb \
  -show_classified file=classified.txt

```

[Example 8-2](#) shows the first error record from the classified error report (`classified.txt`) for this example.

### Example 8-2 Error Record from Classified Error Report

```

-----
                        Classified Error Report
-----
Report Violation: ALL
Report Cell:      ALL
Report Class:     ALL
=====
Violation: " toxcont must be enclosed by met1 by more than 1.0
  without touching or overlapping"
-----
Cell: TGATE

Class Summary:
      Class      Errors
      -----
      Watch      6

Command: runset.rs:101:enclose
-----
( lower left x, y ) ( upper right x, y) Distance
-----
(3.0000, 4.0000)   (4.0000, 5.1180)   1.0000
* Class:           Watch
* Created by:      juser on 2008-10-21 08:36:19
* Last Modified by: juser on 2008-10-21 08:36:19
* Comment by:      juser on 2008-10-21 08:36:19
      These need to be fixed

```

## Creating an Error Classification Database

To use the classification data in a subsequent run, run the `pydb_export` utility to create an error classification database (cPYDB). For example,

```
pydb_export -pydb_name PYDB_AD4FUL \
-pydb_path ./run_details/pydb \
-cpydb_name CPYDB_AD4FUL -cpydb_path ./cpydb -create
```

This example creates a new error classification database called CPYDB\_AD4FUL in the ./cpydb directory. This database can be used in subsequent IC Validator runs to retain the error classification data. The pydb\_export utility produces a log file, pydbexport.log, in the current directory. The output from this example run is shown in [Example 8-3](#).

### Example 8-3 Initial Error Classification Database

```
Called as: pydb_export -pydb_name PYDB_AD4FUL -pydb_path
./run_details/pydb -cpydb_name CPYDB_AD4FUL -cpydb_path ./cpydb
-create

Operation ..... Create New Database
Input PYDB Path ..... ./run_details/pydb
Input PYDB Name ..... PYDB_AD4FUL
Output cPYDB Path ..... ./cpydb
Output cPYDB Name ..... CPYDB_AD4FUL
Export Matched Errors ..... FALSE

Loading global database info
Loading database time = 0:00:00 User=0.00 Sys=0.01 Mem=0.460
Exporting classified error data
Exporting violation " toxcont must be enclosed by met1 by more
than 1.0 without touching or overlapping"
6 new errors exported
Exporting time = 0:00:00 User=0.00 Sys=0.00 Mem=0.616
Exporting violation " toxcont must be enclosed by tox by more
than 1.5"
6 new errors exported
Exporting time = 0:00:00 User=0.01 Sys=0.00 Mem=0.616
Exporting violation "Met1 Spacing must be >= 3"
6 new errors exported
Exporting time = 0:00:00 User=0.00 Sys=0.00 Mem=0.647
Exporting violation "ngate must be exactly 2.0 by 6.0 rectangles"
2 new errors exported
Exporting time = 0:00:01 User=0.00 Sys=0.00 Mem=0.788
Exporting violation "pgate must be exactly 2.0 by 12.0
rectangles"
1 new error exported
Exporting time = 0:00:00 User=0.00 Sys=0.00 Mem=0.803
Export time = 0:00:01 User=0.01 Sys=0.01 Mem=0.803
Recreating indexes
Recreating indexes time = 0:00:00 User=0.00 Sys=0.00 Mem=0.717
Overall export time = 0:00:01 User=0.01 Sys=0.02 Mem=0.803
Export Successful.
PYDB_Export is done.
```

## Running the IC Validator Tool Again

After fixing some of the errors, run the IC Validator tool again, importing the error classification database created in the last run. Using the error classification database keeps you from having to duplicate work. To use the error classification database, specify the error classification database in the `error_options()` function:

```
error_options(
    match_errors = {
        {db_name = "CPYDB_AD4FUL",
         db_path = "./cpydb"}
    }
);
```

After the IC Validator tool runs, notice in the new `LAYOUT_ERRORS` file that the errors were pre classified using data from the error classification database.

- The error summary section contains error counts broken out by classification tag.
- The error details section groups errors by their classification tag.

Note:

The `pydb_report` utility can be used to regenerate the `LAYOUT_ERRORS` file from the error database at any time. Also, you can use the `pydb_report` utility to generate the classified error report to show detailed classification information, as shown in [Example 8-4](#) and [Example 8-5](#).

[Example 8-4](#) shows the error summary section.

### Example 8-4 Error Summary Section

```

                                ERROR SUMMARY

toxcont must be enclosed by met1 by more than 1.0 without touching or
overlapping
enclose ..... 9 violations found.
  Unclassified Errors ..... 3 violations found.
  Errors classified as Watch ..... 6 violations found.
```

[Example 8-5](#) shows the error details section.

### Example 8-5 Error Details Section

```

                                ERROR DETAILS

-----
toxcont must be enclosed by met1 by more than 1.0 without touching or
overlapping
-----

run2.rs:107:enclose
Structure ( lower left x, y ) ( upper right x, y) Distance
- - - - -
```



```

TGATE      (21.5000, 23.5000)  (22.0000, 24.9140)  0.5000
TGATE      (21.5000, 19.0860)  (22.0000, 20.5000)  0.5000
TGATE      (21.5000, 20.5000)  (22.0000, 23.5000)  0.5000

```

+ Errors classified as Watch

```

-----
TGATE      (3.0000, 4.0000)    (4.0000, 5.1180)    1.0000
TGATE      (3.0000, 0.0000)    (4.0000, 1.0000)    1.0000
TGATE      (7.0000, 0.0000)    (8.1180, 1.0000)    1.0000
TGATE      (3.0000, 0.0000)    (4.0000, 1.0000)    1.0000
TGATE      (3.0000, 1.0000)    (4.0000, 4.0000)    1.0000
TGATE      (4.0000, 0.0000)    (7.0000, 1.0000)    1.0000

```

## Updating the Existing Error Classification Database

After the second run, you can classify more errors and reclassify existing errors:

```

pydb_report -pydb_name PYDB_AD4FUL \
            -pydb_path ./run_details/pydb -classify classify2.csv

```

Figure 8-3 shows the classify file used.

Figure 8-3 Classify File

	A	B	C	D	E	F	G	H
1	TGATE		Fixed	These are fixed in the layout	0	0	10	10
2	ADFULAH	Met2 width must be >=4.0, convex to convex corners must be >= 4.5	Watch	These need to be fixed				
3								

The log file from the pydb\_report utility is shown in Example 8-6.

### Example 8-6 pydb\_report Log File

```

Connecting to DB: PYDB_AD4FUL
      path: ./run_details/pydb
Connected to DB PYDB_AD4FUL

```

```

Error database PYDB_AD4FUL summary
  Number of commands:          12
  Number of commands w/ errors: 12
  Number of dynamic tables:     5

```

```

Total error polygons:          48
DBU:                          0.00050
Number of unique cells w/ errors: 6

Classifying Errors from classify2.csv...
Classifying all errors in window...
Violation: ALL
Cell:      TGATE
Window:    (0.0000, 0.0000) (10.0000, 10.0000)
Class:     Fixed
Comment:   These are fixed in the layout
12 existing errors reclassified.
Classify time = 0:00:00 User=0.01 Sys=0.01 Mem=0.764
Classifying all errors in violation/cell...
Violation: Met2 width must be >=4.0, convex to convex corners
must be >= 4.5
Cell:      ADFULAH
Class:     Watch
Comment:   These need to be fixed
3 new errors classified.
Classify time = 0:00:00 User=0.00 Sys=0.00 Mem=0.624
Overall Classify time = 0:00:00 User=0.01 Sys=0.01 Mem=0.764
Done.

```

Merge the changes into the existing error classification database (cPYDB), using the `pydb_export` utility:

```

pydb_export -pydb_name PYDB_AD4FUL \
  -pydb_path ./run_details/pydb -cpydb_name CPYDB_AD4FUL \
  -cpydb_path ./cpydb -merge

```

The log file from the `pydb_export` utility for the merge step is shown in [Example 8-7](#).

#### Example 8-7 `pydb_export` Log File

```

Called as: pydb_export -pydb_name PYDB_AD4FUL -pydb_path
./run_details/pydb -cpydb_name CPYDB_AD4FUL -cpydb_path ./cpydb
-merge

Operation ..... Merge
Input PYDB Path ..... ./run_details/pydb
Input PYDB Name ..... PYDB_AD4FUL
Output cPYDB Path ..... ./cpydb
Output cPYDB Name ..... CPYDB_AD4FUL
Export Matched Errors ..... FALSE

Loading global database info
Loading database time = 0:00:00 User=0.00 Sys=0.00 Mem=0.460
Merging classified error data
Exporting violation " toxcont must be enclosed by met1 by more
than 1.0 without touching or overlapping"
6 existing errors updated
Exporting time = 0:00:00 User=0.01 Sys=0.00 Mem=0.764

```

```

Exporting violation " toxcont must be enclosed by tox by more
than 1.5"
  6 existing errors updated
Exporting time = 0:00:00  User=0.00 Sys=0.00 Mem=0.796
Exporting violation "Met2 width must be >=4.0, convex to convex
corners must be >= 4.5"
  3 new errors exported
Exporting time = 0:00:01  User=0.00 Sys=0.00 Mem=0.866
Export time = 0:00:01  User=0.01 Sys=0.01 Mem=0.944
Overall export time = 0:00:01  User=0.01 Sys=0.01 Mem=0.944
Export Successful.
PYDB_Export is done.

```

Running the IC Validator tool again, the LAYOUT\_ERRORS file shows that the changes made to the database are retained.

See the detailed information by running the pydb\_report utility:

```

pydb_report -pydb_name PYDB_AD4FUL \
  -pydb_path ./run_details/pydb \
  -show_classified file=classified2.txt

```

Alternately, you can use VUE to generate the report.

The new output from the classified error report for this example is similar to the error report for the initial run.

---

## pydb\_report Utility

Use the pydb\_report utility with an error database to

- Generate a detailed report of classified errors.
- Regenerate the LAYOUT\_ERRORS file.
- Classify errors.

When an error is classified, the classification tag, UNIX user ID, timestamp, and possible comment are stored with the error in the error database for the current run.

Additionally, the pydb\_report utility can be used on an error classification database to generate a detailed report of classified errors.

Note:

See [“Database Naming Conventions” on page 8-4](#).

This utility is in the IC Validator installation bin directory. [Table 8-1](#) defines the command-line options.

*Table 8-1 pydb\_report Utility Command-Line Options*

Command-Line Option	Definition
<code>-64</code>	Runs the utility with 64-bit coordinates support. This command-line option might be necessary if coordinates in the error database exceed the capacity of a 32-bit signed integer, that is, if the IC Validator tool runs with the <code>-64</code> command-line option.
<code>-classify input_file</code>	Classifies errors in an error database based on the given input file. See <a href="#">“Classify File Format”</a> in the following section.
<code>-classify_by_window [interacting   enclosed]</code>	Specifies how errors are selected for classification based on window coordinates in the classify file. The default is <code>interacting</code> . <ul style="list-style-type: none"> <li><code>interacting</code>. Errors having any interaction with the window coordinates are selected for classification.</li> <li><code>enclosed</code>. Errors completely enclosed by the window coordinates are selected for classification. Selected errors can touch the enclosing window.</li> </ul>
<code>-cpydb_name name</code>	Error classification database (cPYDB) instance name. Required if operating on an error classification database.
<code>-cpydb_path path</code>	Error classification database (cPYDB) path. Required if operating on an error classification database.
<code>-help</code>	Displays the usage message.
<code>-le output_file</code>	Generates the LAYOUT_ERRORS file from the current error database.
<code>-no_error_details</code>	Does not report error details (that is, coordinates) in the LAYOUT_ERRORS or TOP_LAYOUT_ERRORS files.
<code>-pydb_name name</code>	Error database (PYDB) instance name; usually <code>PYDB_top_cell</code> . Required if operating on an error database.
<code>-pydb_path path</code>	Error database (PYDB) path; usually <code>./run_details/pydb</code> unless otherwise specified in the <code>db_path</code> argument of the <a href="#">error_options()</a> function. Required if operating on an error database.

Table 8-1 *pydb\_report Utility Command-Line Options (Continued)*

Command-Line Option	Definition
<code>-show_classified</code> <code>[cell=cell]</code> <code>[violation=comment]</code> <code>[class=class1,..., classn]</code> <code>[file=file]</code>	Generates a detailed classified error report for an error database and error classification database. The specified parameters can restrict the report by violation, cell, and class. If the output file is not specified, output goes to the screen.
<code>-suppress_classified_errors</code>	If specified, no classified errors are reported in the <code>LAYOUT_ERRORS</code> or <code>TOP_LAYOUT_ERRORS</code> file. All classified errors are suppressed, whether they come from the error classification database or an IC Validator option that classifies errors programmatically, or have been classified after the IC Validator run with VUE or the error database.
<code>-tle output_file</code>	Generates the <code>TOP_LAYOUT_ERRORS</code> file from the PYDB database.
<code>-v</code>	Prints the utility version.

## Classify File Format

The format of the input file for the `-classify` command-line option is a comma-delimited file with these columns: Cell, Violation, Class, Comment, x1, y1, x2, y2. See [Figure 8-2](#).

- If a cell is given but not a violation, all errors within the cell are classified.
- If a violation is given but not a cell, all errors within the violation are classified.
- If window coordinates are given, a cell is required and errors that interact with that window are classified. The default is any interaction, but you can change this behavior using the `-classify_by_window` command-line option.

The formatting of records in the file should follow the standard RFC4180 recommended guidelines for CSV files with no column headings. This formatting is supported by Microsoft Excel and other spreadsheet tools.

The classifications you can choose the classifications shown in [Table 8-2](#).

Table 8-2 *Classify File Format*

Classification	Description
Error	Unclassified error (default classification)

Table 8-2 Classify File Format (Continued)

Classification	Description
Ignore	Error that the designer does not care about: to be fixed later in the process flow
Waive	Intended design error. It will exist at sign-off.
Watch	Error that needs to be fixed in the next design cycle.
Fixed	Error that has been fixed in the layout
Unmatched	Error that exists in an imported error classification database but was not produced by the current run

## Using Regular Expression Matching

In the input file specified by the `-classify` command-line option you can now use regular expressions for matching violation comments and cell names in the error database. This matching is enabled by using leading and trailing forward slash (/) characters, with an optional modifier character after the trailing slash. The modifier characters are shown in [Table 8-3](#)

Table 8-3 Modify Characters for Use in the Classify File

Modify character	Description
m	Match case. This character is the default.
i	Ignore case.

For example, the following line in a classify file results in all violation comments starting with M1, M2, M3, or M4 in cell Cell1 being classified as ignore:

```
Cell1, "/^M[1-4].*/", Ignore, "comment"
```

The following line in a classify file results in violations in all cells containing "sram", ignoring case, being waived:

```
"/.*sram.*/i", "", Waive, "comment"
```

A violation comment that actually begins with a slash cannot be matched with a literal string, but must use GNU extended regular expression special characters. See the table of these special characters in the `text_options()` function in the *IC Validator Reference Manual* for more information. For example, the following in a classify file would match a violation comment containing `"/M4 rule 5/`:

```
"/\M4 rule 5\/"
```

## Examples

To regenerate TOP.LAYOUT\_ERRORS from the current run, use this syntax:

```
pydb_report -pydb_name PYDB_TOP -pydb_path ./run_details/pydb \
-le TOP.LAYOUT_ERRORS
```

To classify errors in the error database for the current run, use this syntax:

```
pydb_report -pydb_name PYDB_TOP -pydb_path ./run_details/pydb \
-classify classify.csv
```

To show classification details on screen of all ignored and waived errors in the cell TOP, use this syntax:

```
pydb_report -pydb_name PYDB_TOP -pydb_path ./run_details/pydb \
-show_classified cell=TOP class=Ignore,Waive
```

To show classification details of all waived errors in an error classification database, use this syntax:

```
pydb_report -cpydb_name CPYDB_TOP -cpydb_path ./cpydb \
-show_classified class=Waive
```

---

## pydb\_export Utility

Use the pydb\_export utility to export error classification data from an error database to either

- Create a new error classification database (cPYDB). See [“Error Classification Database Format” on page 8-21](#).
- Merge into an existing error classification database by appending new errors and updating existing ones.

Also, the pydb\_export utility can be used to manage error classification databases by merging several error classification databases together or by purging classification data from violations and cells.

For an error to be considered equivalent between runs, the violation comments must match exactly or match up to the first occurrence of a specified comment delimiter, and the error polygon must match at cell level.

Note:

See [“Database Naming Conventions” on page 8-4](#).

This utility is in the IC Validator installation bin directory. [Table 8-4](#) defines the command-line options.

*Table 8-4 pydb\_export Utility Command-line Options*

Command-Line Option	Definition
<code>-64</code>	Runs the utility with 64-bit coordinates support. This command-line option might be necessary if coordinates in the error database exceed the capacity of a 32-bit signed integer, that is, if the IC Validator tool runs with the <code>-64</code> command-line option.
<code>-cell cell</code>	Limit exports to errors of the given cell. This command-line option can be given more than one time to specify multiple classes. It supports regular expressions.
<code>-class class</code>	Limit exports to errors of the given classification. This command-line option can be given more than one time to specify multiple classes. It supports regular expressions.
<code>-comment_match_delimiter delim</code>	<p>Match violation comments up to the specified delimiter.</p> <p>When exporting an error database, the default of the delimiter is whatever is specified in the runset using the <code>comment_match_delimiter</code> argument of the <code>error_options()</code> function.</p> <p>When merging multiple error classification databases, however, this command-line option delimiter should be specified to use partial violation comment matching.</p>
<code>-convert_to_64</code>	Convert a 32-bit or variable width coordinate error classification database (cPYDB) to a 64-bit coordinate error classification database. When used with the <code>-copy_cpydb</code> or <code>-create</code> command-line options, this command-line option converts the database to 64-bits when it is created. Otherwise, it converts an existing database in place.
<code>-convert_to_32</code>	<p>Convert a 64-bit or variable width coordinate error classification database (cPYDB) to a 32-bit coordinate error classification database. When used with the <code>-copy_cpydb</code> or <code>-create</code> command-line options, this command-line option converts the database to 32-bits when it is created. Otherwise, it converts an existing database in place.</p> <p><b>Note:</b></p> <p>This operation fails if the error classification database contains coordinates that are too large to be represented by 32-bit integers.</p>



Table 8-4 *pydb\_export Utility Command-line Options (Continued)*

Command-Line Option	Definition
<code>-convert_to_variable</code>	Convert a 64-bit or 32-bit coordinate error classification database (cPYDB) to a variable width coordinate error classification database. When used with the <code>-copy_cpydb</code> or <code>-create</code> command-line options, this command-line option converts the database to variable width coordinates when it is created. Otherwise, it converts an existing database in place.
<code>-copy_cpydb name=db_name path=db_path</code>	Copy an error classification database to the destination error classification database, creating a new database or overwriting an existing database.
<code>-cpydb_name name</code>	Output error classification database (cPYDB) instance name. Required.
<code>-cpydb_path path</code>	Output error classification database (cPYDB) path. Required.
<code>-create</code>	Export the error database, either creating a new error classification database or overwriting an existing error classification database.  Or, merge multiple error classification databases, creating a new error classification database or overwriting an existing error classification database.
<code>-exclude_cell cell</code>	Excludes errors in the given cell from export. This command-line option can be given more than one time to specify multiple classes. It supports regular expressions. The <code>-exclude_cell</code> command-line option takes priority over the <code>-cell</code> command-line option.
<code>-exclude_class class</code>	Excludes errors of the given classification. This command-line option can be given more than one time to specify multiple classes. It supports regular expressions. The <code>-exclude_class</code> command-line option takes priority over the <code>-class</code> command-line option.
<code>-exclude_violation violation</code>	Excludes errors in the given violation comment from export. This command-line option can be given more than one time to specify multiple classes. It supports regular expressions. The <code>-exclude_violation</code> command-line option takes priority over the <code>-violation</code> command-line option.

Table 8-4 *pydb\_export Utility Command-line Options (Continued)*

Command-Line Option	Definition
<code>-export_matched</code>	<p>Include matched errors. The default is to export only newly classified errors.</p> <p>When merging into an existing error classification database, this option specifies whether to include matched errors from error classification databases other than the destination.</p>
<code>-help</code>	Displays the usage message.
<code>-merge</code>	Merge the specified error database into the specified error classification database, updating classification of existing errors or adding new errors to the database.
<code>-merge_cpydb</code> <code>name=db_name</code> <code>path=db_path</code>	Merge an error classification database into the destination error classification database. You can use multiple options on a command line to merge multiple error classification databases.
<code>-password password</code>	Password for write access to an existing error classification database. Specify <code>prompt</code> to read the password from the command line.
<code>-purge purge_file</code>	<p>Purge all classification data from violations and cells based on the given input file. See <a href="#">“Purge File Format” on page 8-21</a> for file format information.</p> <p>If any error databases reference the target error classification database and errors matched the violations and cells that are being purged, those errors are no longer viewable.</p>
<code>-pydb_name name</code>	Input error database (PYDB) instance name; usually <code>PYDB_top_cell</code> . Required for export.
<code>-pydb_path path</code>	Input error database (PYDB) path; usually <code>./run_details/pydb</code> unless otherwise specified in the <code>db_path</code> argument of the <code>error_options()</code> function. Required for export.

Table 8-4 *pydb\_export Utility Command-line Options (Continued)*

Command-Line Option	Definition
<code>-read_only</code>	Specifies that the output error classification database (cPYDB) should be read-only. When used with the <code>-create</code> or <code>-copy_cpydb</code> command-line options, the cPYDB is changed to read-only mode after it is created. Otherwise, it changes an existing database to read-only mode. A read-only database may not be modified in any way. Each IC Validator component that connects to a read-only cPYDB has its own database server process. A writable database might not be converted to read-only while other users are connected.
<code>-set_password password</code>	Set the password on the given error classification database. Specify <code>prompt</code> to read the password from the command line.
<code>-v</code>	Prints the utility version.
<code>-violation violation</code>	Limit exports to the given violation comment. This command-line option can be given more than one time to specify multiple classes. It supports regular expressions.

## Error Classification Database Format

The error classification database contains classification tags, user IDs, timestamps, comments, and error polygons for the classified errors. When an error classification database is used in a subsequent run, errors that match are annotated in the database with a reference to the error classification database. The actual classification data for these errors remains in the error classification database. Therefore, to view errors for the current run, the error classification database needs to be present in the location specified when the IC Validator tool was run. Also, if data is purged from the error classification database or is overwritten, you cannot view the matched errors from the current run.

## Purge File Format

The format of the input file for the `-purge` command-line option is a comma-delimited file with two columns: Cell and Violation.

- If a cell is specified but not a violation, all errors within the cell are purged.
- If a violation is specified but not a cell, all errors within the violation are purged.

Format the records in the file to follow the standard RFC4180 recommended guidelines for CSV files with no column headings. This formatting is supported by Microsoft Excel and other spreadsheet tools.

## Examples

To create a new error classification database (cPYDB) containing classification data from the current run, use this syntax:

```
pydb_export -pydb_name PYDB_AD4FUL \
  -pydb_path ./run_details/pydb \
  -cpydb_name CPYDB_AD4FUL -cpydb_path ./cpydb \
  -create
```

To set the password on an error classification database, use this syntax:

```
pydb_export -cpydb_name CPYDB_AD4FUL -cpydb_path ./cpydb \
  -password oldpassword -set_password newpassword
```

To merge classification data into an existing error classification database that is password protected, use this syntax:

```
pydb_export -pydb_name PYDB_AD4FUL \
  -pydb_path ./run_details/pydb \
  -cpydb_name CPYDB_AD4FUL -cpydb_path ./cpydb \
  -merge -password password
```

To merge several error classification databases together, creating a new error classification database, use this syntax:

```
pydb_export -cpydb_name CPYDB_MY_DESIGN -cpydb_path ./cpydb \
  -merge_cpydb name=CPYDB_CELLA path=./cpydb \
  -merge_cpydb name=CPYDB_CELLB path=./cpydb \
  -merge_cpydb name=CPYDB_TOP path=./cpydb \
  -creates
```

To convert an error classification database with 32-bit coordinates to one with 64-bit coordinates in place, use the following syntax. When using the `-convert_to_64` command-line option, you need exclusive access to the database; that is, the database cannot be open by VUE, any other scripts, or IC Validator jobs.

```
pydb_export -cpydb_name db name -cpydb_path db path -convert_to_64
```

To copy an error classification database and convert it to 64-bit without modifying the original database, use this syntax:

```
pydb_export -cpydb_name db name -cpydb_path db path \
  -copy_cpydb name=input db name path=input db path \
  -convert_to_64
```

The `pydb_report` utility can provide information about the coordinate width of an existing error classification database:

```
pydb_report -cpydb_name CPYDB_AD4FUL64 -cpydb_path ./cpydb
...
```

(C) Copyright 2008-2014.

Synopsys, Inc. All rights reserved.

```
cPYDB Name ..... CPYDB_AD4FUL64  
cPYDB Path ..... ./cpydb  
Coordinate Width ..... 64 bit
```

To export only the classes starting with "W" in all cells that do not start with "W", use this syntax:

```
pydb_export -class "/^W.*/" -exclude_cell "/^W.*/" ...
```

To export all classes except "Ignore" and "Waive", use this syntax:

```
pydb_export -exclude_class Ignore -exclude_class Waive ...
```

To export only the top cell, use this syntax:

```
pydb_export -cell TOP ...
```



# 9

## Pattern Matching

---

*This chapter provides an introduction to IC Validator pattern matching. The pattern matching functions are described in the [IC Validator Reference Manual](#).*

The IC Validator pattern matching feature is described in the following sections:

- [Overview](#)
- [Using the Pattern Matching Flow](#)
- [Creating the Pattern Library](#)
- [Viewing and Locking a Pattern Library](#)
- [Checking a Layout](#)
- [Detecting and Repairing Hotspots Within the IC Compiler Tool](#)

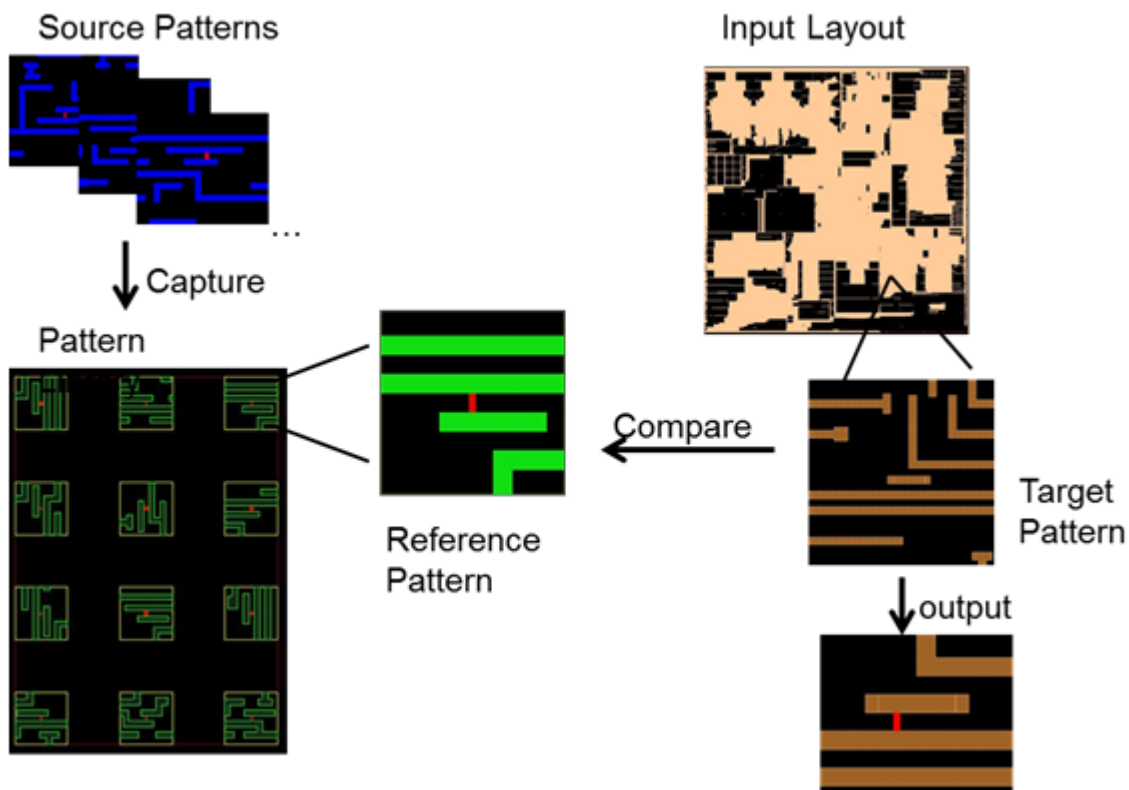
## Overview

IC Validator pattern matching compares two geometrical configurations (patterns) to determine if they match. In pattern matching process, you match a target pattern from a design layout with a reference pattern from an existing pattern library to determine if the patterns match. [Figure 9-1](#) illustrates this step.

The reference pattern represents any combination of geometric shapes that cannot be manufactured, such as a lithography hotspot, or are difficult to be interpreted accurately with standard DRC rules, such as a complex 2-dimensional geometric measurements. Reference patterns are pre-identified through robust verification and validation techniques, and they are precisely specified and added to a pattern library before matching.

An advantage of pattern matching as a DRC-based flow is that it is much faster than the simulation-based lithography detection, which is extremely computationally expensive and time consuming. Therefore, pattern matching offers a cost-effective approach to identify lithography-induced or process-induced hotspots earlier in the design flow, and it makes real-time verification and repairing possible.

*Figure 9-1 Pattern Matching Flow*

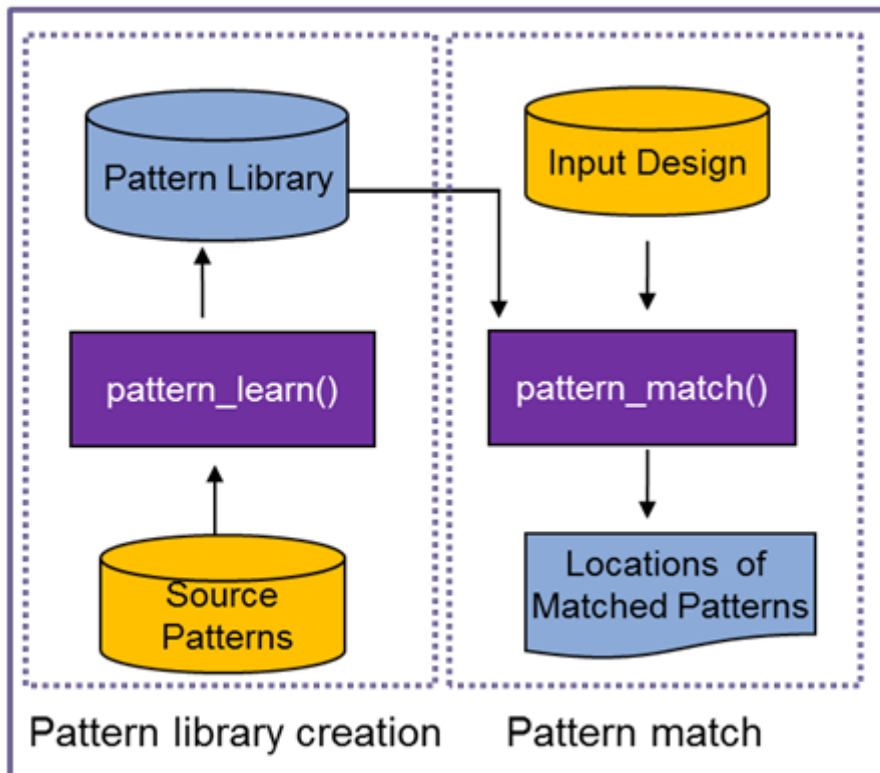




## Using the Pattern Matching Flow

To use the patching matching flow, you provide a set of source patterns in the form of design pattern layers and location marker layers. These layers are identified either at a manufacturing site known as yield detractors or at a fabless company for methodology checking. [Figure 9-2](#) shows a top-level IC Validator pattern-matching flow for pattern library creation and pattern matching.

Figure 9-2 Pattern Matching Flowchart



During pattern library creation, pattern information is extracted from the source pattern and written to a pattern library. The pattern library is called during the pattern matching step as a reference for identifying the same or similar patterns in the input design layout.

The `pattern_learn()` function creates a pattern library. The `pattern_match()` function performs pattern matching. See Chapter 1, "Tables of Runset Functions" in the *IC Validator Reference Manual* for a summary of all of the pattern matching functions.

---

## Creating the Pattern Library

This section shows an example of the pattern library creation flow using the `pattern_learn()` function.

## Preparing the Input Layout

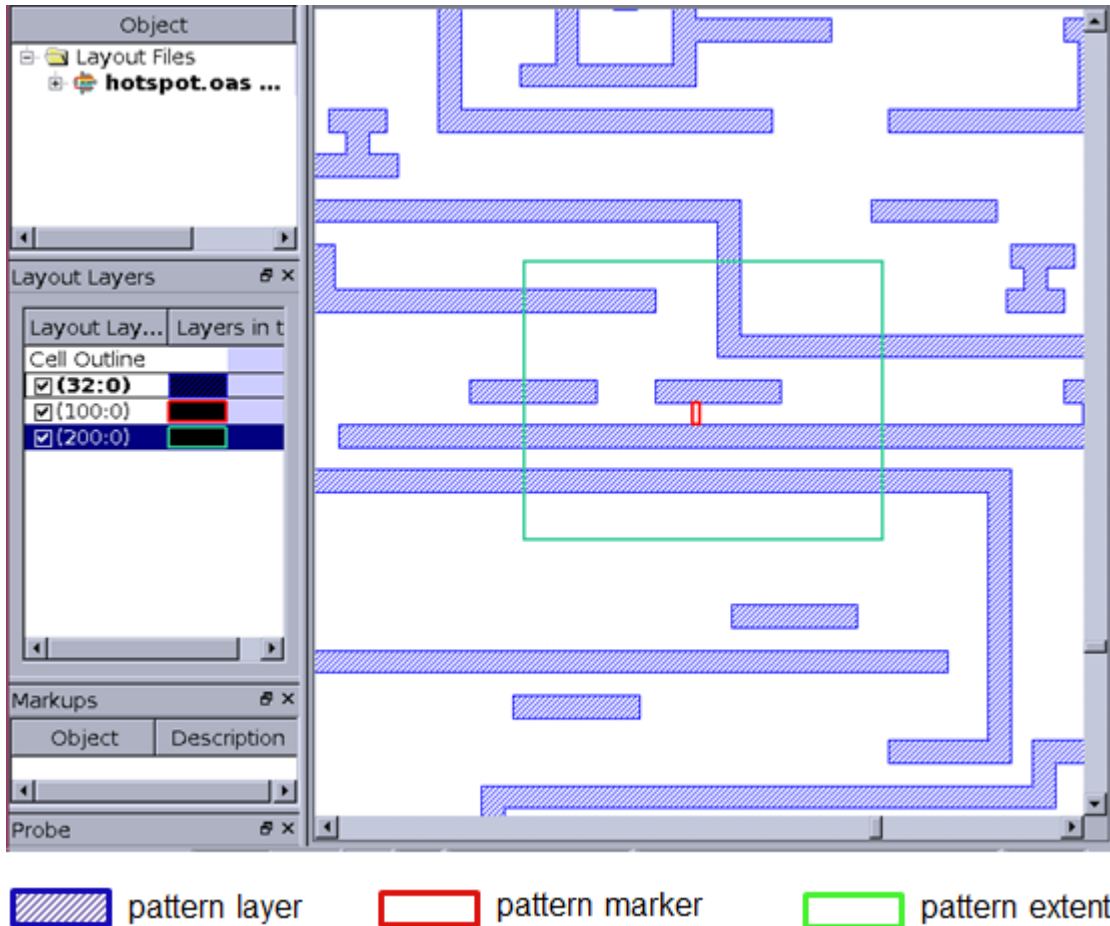
Source patterns in the GDSII or OASIS formats are captured and registered to a pattern library through a pattern definition process, which contains specifications to precisely describe a pattern.

## Required Input

Three user-defined inputs are required for creating a pattern library. These inputs are part of the input layout, or they are generated through runset operations before the `pattern_learn()` function call. [Figure 9-3](#) shows an example of required input.

- Pattern layers. Specifies that a pattern contains single or multiple pattern layers, such as a pattern consisting of metal and via layers.
- Pattern marker. Specifies a rectangular polygon that is placed on each source pattern to mark the location of interest, such as the pinch or bridge location, or a DRC violation. The pattern marker is output during pattern matching to report the match between the pattern library and the design.
- Pattern extent. Specifies the bounding region of a pattern. Pattern layers in this region are processed and registered to the pattern library. The pattern extent can also be generated through the `ambit_mode` and `match_ambit` arguments of the `pattern_learn()` function.

Figure 9-3 Required Input for Creating a Pattern Library



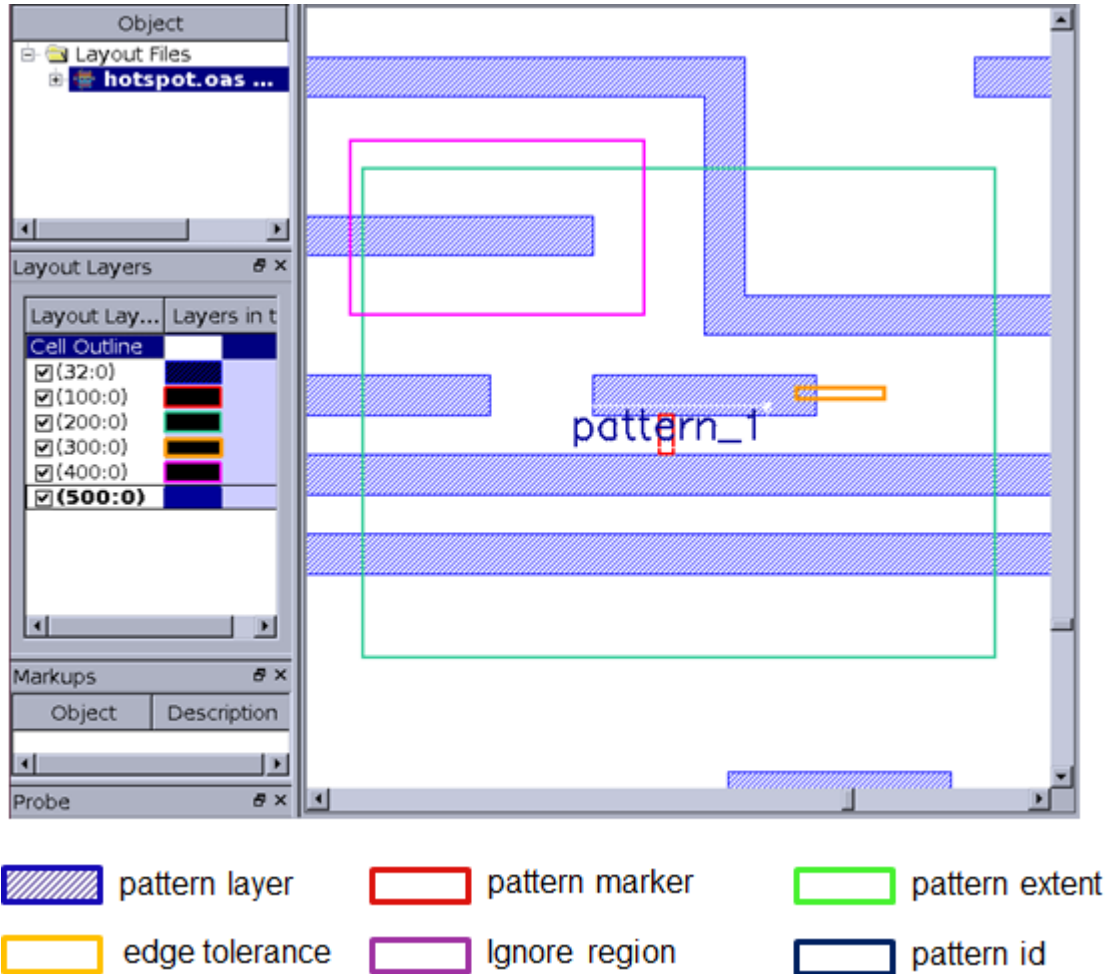
## Optional Input

You might also need to provide the following specifications for your pattern definition. [Figure 9-4](#) shows an example of optional input.

- Edge tolerance layers. Specifies a set of polygon layers that define the allowed edge placement variation for each pattern layer edge.
- Ignore region layers. Contains one or more rectilinear polygons defining areas on the source pattern that are excluded from the matching process.
- Optional pattern marker layers. Contains polygons that can be attached to a pattern and retrieved for certain post-pattern matching operations. For example, it can be user-defined fix guidance for a particular pattern.
- Pattern text ID layer. Contains the text ID of the source patterns. If the text ID layer is not provided, a tool-generated text ID is used.

- Pattern text property layers. Contains the text information defined for a source pattern, for example, the pattern ID, pattern type, or minimum CD. You attach these properties to the pattern during pattern library creation and retrieve them by using the `select_marker_by_double_property()` and `select_marker_by_string_property()` functions for post-pattern matching operations.

Figure 9-4 Optional Input for Creating a Pattern Library



## Additional Pattern Specifications

You might need to set additional pattern specifications during pattern library creation, such as:

- Pattern fuzziness. Specifies the pattern matching mode. You can choose the exact or fuzzy matching mode. The pattern fuzziness is edge-based and is specified individually and globally.

- Pattern reflection and rotation. Specifies whether to report a pattern as a match when it has a different orientation from the original shape.
- Pattern type. Specifies whether the source patterns are 1-dimensional or 2-dimensional patterns. Mixed pattern types are not supported.

See the *IC Validator Reference Manual* for more information about the `pattern_learn()` function.

## Example of Creating a Pattern Library

You create a standard DRC runset file to create a pattern library. The runset must include the following sections:

- Layout database specifics
- General options
- Pattern library path definition using the `pattern_options()` function
- Layer assignments
- Pattern library creation using the `pattern_learn()` function

A pattern matching runset that is used to create a pattern library is described in the following steps:

### 1. Beginning of the runset

```
#include <icv.rh>
```

### 2. User-defined environment variables

Environment variables in the runset are optional and can be customized.

Two environment variables are defined for pattern library creation. The `PATTERN_LIB_PATH` environment variable specifies the location where the pattern library is saved. The `PM_LIB_NAME` environment variable specifies the name of the pattern library. For example,

```
#pragma envvar default PM_LIB_PATH "/pattern_library"
#pragma envvar default PM_LIB_NAME "MET"
```

### 3. Layout database specifications

The `library()` function defines the design library, and it is required in the runset.

```
library (
    format      = GDSII,
    library_name = "TOP.gds",
    cell        = "TOP"
);
```

Use the IC Validator command-line options to override arguments specified in the `library()` function. For example, use the `-i` command-line option to override the library name specified by `library_name` or the `-f` command-line option to override the library format specified by `format`. See the [Command-Line Options](#) section in the “IC Validator Basics” chapter of the *IC Validator User Guide* for more information.

#### 4. General Options

Set `error_limit_per_check` to `ERROR_LIMIT_UNLIMITED` in the `error_options()` function to store all of the errors reported by the `pattern_learn()` and `pattern_match()` functions. The default is 100.

```
error_options (
    error_limit_per_check = ERROR_LIMIT_UNLIMITED
);
```

#### 5. Call the `pattern_options()` function to define the pattern library path.

The `pattern_options()` function defines the path of a pattern library for the `pattern_learn()` and `pattern_match()` functions. The `pattern_options()` function can be called only one time in the runset. See the Runset Functions chapters of *IC Validator Reference Manual* for more information.

```
pattern_options (
    pattern_library_path = $PM_LIB_PATH
);
```

##### Note:

The pattern library path specified by the `pattern_library_path` argument must exist before pattern library creation. Otherwise a runtime error occurs.

#### 6. Layer assignments

```
metal : polygon_layer = assign ( {{3,0}} );
pattern_marker : polygon_layer = assign ( {{4,0}, {5,0}} );
pattern_text_id : text_layer = assign_text( {{6,0}} );
error_list : list of write_error_map_s = {};
```

#### 7. Call the `pattern_learn()` function to create a pattern library.

- Define parameters for pattern library creation.

This optional section of the runset provides access to a list of values for any parameter of the `pattern_learn()` function. See the descriptions of the `pattern_learn()` function in *IC Validator Reference Manual* for more information about the values for parameters in the `pattern_learn()` function.

```
match_ambit : ambit_values_s = {0.2,0.2,0.2,0.2};
ambit_mode : pattern_ambit_mode_e = PM_MARKER_CENTER;
pattern_fuzziness : pattern_fuzzy_e = PM_EDGE_UNIFORM;
uniform_fuzzy_size : double = 0.004;
pattern_reflect : boolean = true;
```

```
pattern_rotate : boolean = true;
```

- Pattern library creation

Call the `pattern_learn()` function to create a pattern library. You can call this function multiple times in a runset. The following example of calling the `pattern_learn()` function to create a single pattern layer library with edge uniform matching mode. The pattern extent is  $0.4 \times 0.4 \mu\text{m}$  and is created by the tool from the center of the pattern marker with  $0.2 \mu\text{m}$  `match_ambit`.

```
tmp_violation @= {
    @"pattern_learn result";
    pattern_learn (
        pattern_library_name = $PM_LIB_NAME,
        pattern_layers = {metal},
        pattern_marker = pattern_marker,
        pattern_text_id = pattern_text_id,
        match_ambit = match_ambit,
        ambit_mode = ambit_mode,
        pattern_fuzziness = pattern_fuzziness,
        uniform_fuzzy_size = uniform_fuzzy_size,
        pattern_reflect = pattern_reflect,
        pattern_rotate = pattern_rotate
    );
}
```

A pattern library directory with a name defined by the `PM_LIB_NAME` environment variable is generated and stored in the path defined by the `PM_LIB_PATH` environment variable.

The error output from the `pattern_learn()` function is stored as the `pattern_learn result`. See [“Examining the Output of Pattern Library Creation”](#) for more information about the error output from the `pattern_learn()` function.

## 8. Output layout database

In this optional section, you write the output from the `pattern_learn()` function to either a GDSII or OASIS file.

```
//DEBUG output only
error_list.push_back({tmp_violation,{0,0}});
output_lib=gds_library("out.gds");
write_gds(
    output_library=output_lib,
    errors=error_list
);
```

## Examining the Output of Pattern Library Creation

The output files produced by the IC Validator runset for pattern library creation include the pattern libraries and the same set of files as you use to perform DRC. See the [Output Files](#) chapter of the *IC Validator User Guide* for more details.

- Error Output

The IC Validator tool creates the `cell.LAYOUT_ERRORS` file when the DRC run completes. For pattern library creation, this file is used as a quick summary of the pattern creation statistics rather than a violation report.

The following `TOP.LAYOUT_ERRORS` file reports an ERROR result:

```

LAYOUT_ERRORS_RESULTS: ERRORS

#####  #####  #####  ###  #####  ###
#      #  #  #  #  #  #  #  #  #
#####  #####  #####  #  #####  ###
#      #  #  #  #  #  #  #  #  #
#####  #  #  #  #  ###  #  #  #####

=====
Library name:      ../../testcase/hotspot_clips.oas
Structure name:    TOP
Generated by:      IC Validator RHEL64 Release I-2013.12.SP1.4010 2014/
01/09

                        ERROR SUMMARY

Runset name: icv runset.rs
User name:  juser
Time started: 2016/03/02 07:35:12AM
Time ended: 2016/03/02 07:35:51AM

Called as: icv runset.rs
pattern_learn result
    pattern_learn ..... 6 violations found.

pattern_learn result/Invalid input: first pattern
layer has no corner within the pattern extent
    pattern_learn ..... 1 violation found.

pattern_learn result:pattern_1
    pattern_learn ..... 4 violations found.

pattern_learn result:pattern_5
    pattern_learn ..... 2 violations found.

```

In this report, six source patterns are processed by the `pattern_learn()` function. Two of the six patterns are identified as `pattern_5` and four are identified as `pattern_1`. Therefore, two unique patterns are captured in the pattern library.

The `pattern_learn()` function also identifies one pattern as an invalid 2-dimensional pattern because it has no corner within the pattern extent. Therefore, it cannot be added



to a 2-dimensional pattern library. Check the `cell.LAYOUT_ERRORS` file after pattern library creation to determine if there are invalid patterns.

- **Pattern Library**

You can store more than one pattern library under the same pattern library path defined by the `pattern_library_path` argument of the `pattern_options()` function. You call one or multiple pattern libraries during pattern matching.

The `pattern_learn()` function writes pattern information to the binary `pattern.dat` file, and it outputs a log file in the same directory defined by `pattern_library_path` argument.

[Example 9-1](#) shows the log file generated for the pattern library, MET.

#### *Example 9-1 Log File*

```
[Tue Nov  5 15:19:22 2013]: Created pattern library.
version= 1.0
dbu=0.000500
learn_version=3
num_pattern_layers=1
num_optional_marker_layers=0
ambit_mode=PM_MARKER_CENTER
match_ambit.left=0.200000
match_ambit.bottom=0.200000
match_ambit.right=0.200000
match_ambit.top=0.200000
pattern_fuzziness=PM_EDGE_UNIFORM
pattern_grouping=PM_NONE
uniform_fuzzy_size=0.005000
edge_jog_size=0.000000
has_user_anchor=false
has_pattern_extent_layer=false
ignore_extra_polygons=false
pattern_type=PM_TWO_DIMENSIONAL
pattern_length=PM_GE
viewable=true
writable=true

[Tue Nov  5 15:19:22 2013]: Learned new pattern(s).
Registered 2 new unique pattern(s); total # of unique pattern(s)=2.
Created reflected/rotated pattern(s); total # of pattern(s)=16.
Derived 16 index pattern(s) using ambit = (0.195000, 0.195000,
    0.195000, 0.195000).
viewable=true; writable=true.
```

At the beginning of the log file, you see the parameters and their corresponding values applied for generating this pattern library for the first time. To add new patterns to an existing pattern library, use the same set of parameters and values. Otherwise an error occurs.

At the end of the log file, you see the number of unique patterns that are registered for each `pattern_learn()` function call.

The last line of the log file indicates that this pattern library is writable, that is, the new patterns can be added to it. This pattern library can be converted to either the GDSII or OASIS format for viewing, if it is viewable.

---

## Viewing and Locking a Pattern Library

You store the pattern library in a binary format. To view its content, convert it to a GDSII or OASIS file.

The `pattern_library_read()` function is required for conversion. See the Runset Functions chapter in *IC Validator Reference Manual* for an example of how to convert a pattern library to a GDSII file.

By default, a pattern library is viewable and writable. The `pattern_library_lock()` function adds read and write protection to a pattern library that, after locked, is neither accessible by the `pattern_learn()` function for adding new patterns, nor by the `pattern_library_read()` function for converting it to a graphical format file. The following line, `viewable=false; writable=false`, is appended to the existing log file of the locked pattern library.

The locked pattern library is then delivered to the end users. See Chapter 3, “Runset Functions: J - Z,” in *IC Validator Reference Manual* for an example of how to lock a pattern library.

Alternatively, you can use the `pdb_utility.pl` script, which is in the IC Validator contrib directory, to perform the functionality of both the `pattern_library_read()` and `pattern_library_lock()` functions without executing a runset:

```
Usage: pdb_utility.pl <pdb_lib_path> <pdb_lib_name> -read/-lock
       [output_name]
```

Options:

```
<pdb_path>.....pattern library path
<pdb_name>.....pattern library name
-read.....converting a pattern library to a graphical
output.
-lock.....adds the write and view protection to a pattern
library.
<output_name>.....optional, specifies the graphical output file
name, [default:pdb_view.gds]
```

This example converts the met pattern library, which is located in the `/test/pattern_lib` directory, to the `hs.gds` output GDSII file.

```
$ICV_HOME_DIR/contrib/pdb_utility.pl /test/pattern_lib met -read hs.gds
```

---

## Checking a Layout

This section shows an example of the pattern matching flow using the `pattern_match()` function.

## Runset Example of Pattern Matching

You create a standard DRC runset file to run the IC Validator tool to perform pattern matching on a design layout. The runset file must include the following sections:

- Layout database specifics
- General options
- Pattern library path definition using the `pattern_options()` function
- Layer assignments
- Performs pattern matching with the `pattern_match()` function

A runset that uses the pattern matching flow for has these components:

### 1. Beginning of the runset

Include `icv.rh` at the beginning of the runset.

```
#include <icv.rh>
```

### 2. User-defined environment variables

Environment variables in the runset are optional and can be customized.

```
#pragma envvar default PM_LIB_PATH "/pattern_library"
#pragma envvar default PM_LIB_NAME "MET"
#pragma envvar default PM_TOP_MET "10"
```

### 3. Layout database specifics

The `library()` function defines the design library used in the pattern matching check.

```
library (
    format      = GDSII,
    library_name = "TOP.gds",
    cell        = "TOP"
);
```

If you use a Milkyway library, set `format` to `MILKYWAY` and specify the `library_path` to the parent directory of the Milkyway library.

### 4. General options

To store and report all of the violations generated by the `pattern_match()` function, set `error_limit_per_check` to `ERROR_LIMIT_UNLIMITED` in the `error_options()` function. The default is 100.

```
error_options (
    error_limit_per_check = ERROR_LIMIT_UNLIMITED
);
```

5. Call the `pattern_options()` function to define the pattern library path.

```
pattern_options (
    pattern_library_path = $PM_LIB_PATH
);
```

The `pattern_options()` function defines the path of a pattern library, and it can be called only one time in the runset. This function specifies the path you need to access a single or multiple pattern libraries.

6. Layer assignments

```
MET1 : polygon_layer = assign ( {{15,0}} );
MET2 : polygon_layer = assign ( {{17,0}} );
MET3 : polygon_layer = assign ( {{19,0}} );
MET4 : polygon_layer = assign ( {{21,0}} );
MET5 : polygon_layer = assign ( {{23,0}} );
MET6 : polygon_layer = assign ( {{25,0}} );
MET7 : polygon_layer = assign ( {{27,0}} );
MET8 : polygon_layer = assign ( {{29,0}} );
MET9 : polygon_layer = assign ( {{31,0}} );
MET10 : polygon_layer = assign ( {{33,0}} );
```

```
metal_layer_list : list of polygon_layer = {MET1, MET2, MET3, MET4,
    MET5, MET6, MET7, MET8, MET9, MET10};
metal_error_list : list of write_error_map_s = {};
```

In this example, MET1 through MET10 are assigned and subjected to a pattern matching check.

7. Call the `pattern_match()` function to perform pattern matching.

You must call the `pattern_match()` function to perform pattern matching. This function can be called multiple times within a runset. The following example shows the `pattern_match()` function executing pattern matching on the metal layers of the TOP.gds file.

```
for(i = 0 to strtol($PM_TOP_MET)-1) {
    layer_index : integer = i + 1;
    tmp_violation @= {
        @"METAL"+layer_index+" PM RESULT: ";
        pattern_match (
            pattern_library_name = $PM_LIB_NAME,
            pattern_layers = {metal_layer_list[i]}
        );
    };
}
```

```
}
    out_layer_index = layer_index;
    metal_error_list.push_back({tmp_violation,
{out_layer_index,0}});
}
```

## 8. Output layout database

Optionally, you can write the output from the `pattern_match()` function to a GDSII or OASIS file.

```
output_lib=gds_library("out.gds");
write_gds(
    output_library=output_lib,
    errors=metal_error_list
);
```

## Examining Pattern Matching Output

The IC Validator tool creates the `cell.LAYOUT_ERRORS` error file when the DRC run completes. The beginning of the file states that the run is either CLEAN or has ERRORS. In pattern matching, the ERRORS result means that one or more matched patterns are identified. The CLEAN result means that no matched patterns are found.

The following TOP.LAYOUT\_ERRORS file reports violations (matches):

```

LAYOUT ERRORS RESULTS: ERRORS
#####  #####  #####  ###  #####  ###
#      #  #  #  #  #  #  #  #  #
#####  #####  #####  #  #  #####  ###
#      #  #  #  #  #  #  #  #  #
#####  #  #  #  #  ###  #  #  #####
=====
Library name:      ./test.gds
Structure name:    TOP
Generated by:      IC Validator RHEL64 Release H-2013.06.SP2.4035 2013/08/
28

                        ERROR SUMMARY
METAL2 PM RESULT:
    pattern_match .....873 violations found.

METAL3 PM RESULT:
    pattern_match .....69 violations found.

METAL4 PM RESULT:
    pattern_match .....2 violations found.

                        ERROR DETAILS
-----
METAL2 PM RESULT:
-----
/match.rs:38:pattern_match
-----
Structure (lower left x, y ) (upper right x, y ) Pattern Orientation
-----
TOP    (8.1600, 528.2500) (8.4000, 528.7200) pattern_78 R90
TOP    (2.1400, 528.6700) (2.3800, 529.1400) pattern_57 R90
TOP    (100.1450, 529.3000) (100.3850, 529.7700) pattern_107 R90FX
TOP    (100.1450, 529.2300) (100.3850, 529.7000) pattern_108 R90
TOP    (107.9800, 531.7500) (108.2200, 532.2200) pattern_80 R0
TOP    (83.7900, 534.7600) (84.0300, 535.2300) pattern_7 R180
TOP    (86.0000, 534.8300) (86.2400, 535.3000) pattern_94 R90FY
TOP    (87.6800, 540.2900) (87.9200, 540.7600) pattern_78 R90
.....

```

The error summary section contains the number of matched patterns on each metal layer produced by the `pattern_match()` function. The error details section lists the coordinates of the pattern marker and pattern ID of each matched pattern.

You can review pattern matching results in VUE, just as you do for other DRC violations.

## Post Pattern-Matching Operations

Use the following functions in the post pattern-matching process to select matched patterns based on their properties or retrieve optional pattern markers of certain patterns for additional DRC operations.

- `marker_merge()`. Merges the marker layer into the polygon layer. The marker layer does not merge; it can be processed only by several IC Validator runset functions. The marker layer must be converted to a polygon layer to be processed by other DRC functions.
- `optional_pattern_markers()`. Retrieves all optional pattern markers attached to a pattern. Optional pattern marker layers are polygon layers.
- `select_marker_by_string_property()`. Retrieves the pattern markers based on the specified string properties.
- `select_marker_by_double_property()`. Retrieves the pattern markers based on the specified double properties.
- `milkyway_route_directives()`. Passes user-defined fix guidance to the router for hotspot repair.
- `drc_features_marker()`. Outputs the marker layer.

See the [IC Validator Reference Manual](#) for examples of each function.

---

## Detecting and Repairing Hotspots Within the IC Compiler Tool

To meet the design-for-manufacturing requirements for advanced nodes, the IC Validator tool provides In-Design DFM technology in the implementation environment. This physical verification approach enables early detection and repairing of DRC violations, with negligible physical or timing impacts within the IC Compiler tool. For more information about the Automatic DRC Repair (ADR) flow, see the Synopsys white paper, “In-Design Physical Verification - Automatic DRC Repair (ADR).”

The ADR flow enables In-Design hotspot detection and correction using IC Validator pattern matching during design implementation. With a usage model similar to DRC, you can use a foundry-certified DFM package (containing pattern library and matching runset) or your own yield detractor pattern library to search a design for potential manufacturing related hotspots to perform automatic fixing and revalidation for all errors. This improves the overall manufacturability of the design.

You run IC Validator pattern matching within the IC Compiler tool using the same set of IC Compiler commands used to perform signoff DRC checking. You use the `signoff_drc` command to do pattern matching and the `signoff_autofix_drc` command to automatically fix the violations (hotspots) detected by the `signoff_drc` command. See the *IC Compiler Implementation User Guide*, which is available on SolvNet, or the IC Compiler man pages for detailed command syntax and usage.

The following sections describe how to detect and correct hotspots using the signoff DRC commands in the IC Compiler tool (not applicable in the IC Compiler II tool).

## Setting Up the Physical Signoff Options

Before you use the `signoff_drc` and `signoff_autofix_drc` commands to run pattern matching, set up the physical signoff options that apply to these commands.

For more information, see the “Performing Signoff Design Rule Checking” section in the *IC Compiler Implementation User Guide*.

## Running Pattern Matching for Hotspot Detection

Use the `signoff_drc` command to perform hotspot detection on the routing layers.

For more information, see the “Performing Signoff Design Rule Checking” section in the *IC Compiler Implementation User Guide*.

## Automatically Fixing Hotspot Patterns

Use the `signoff_drc` command to automatically fix detected hotspots.

For more information, see the “Performing Signoff Design Rule Checking” section in the *IC Compiler Implementation User Guide*.

## Repairing Hotspots With User-Defined Fix Guidance

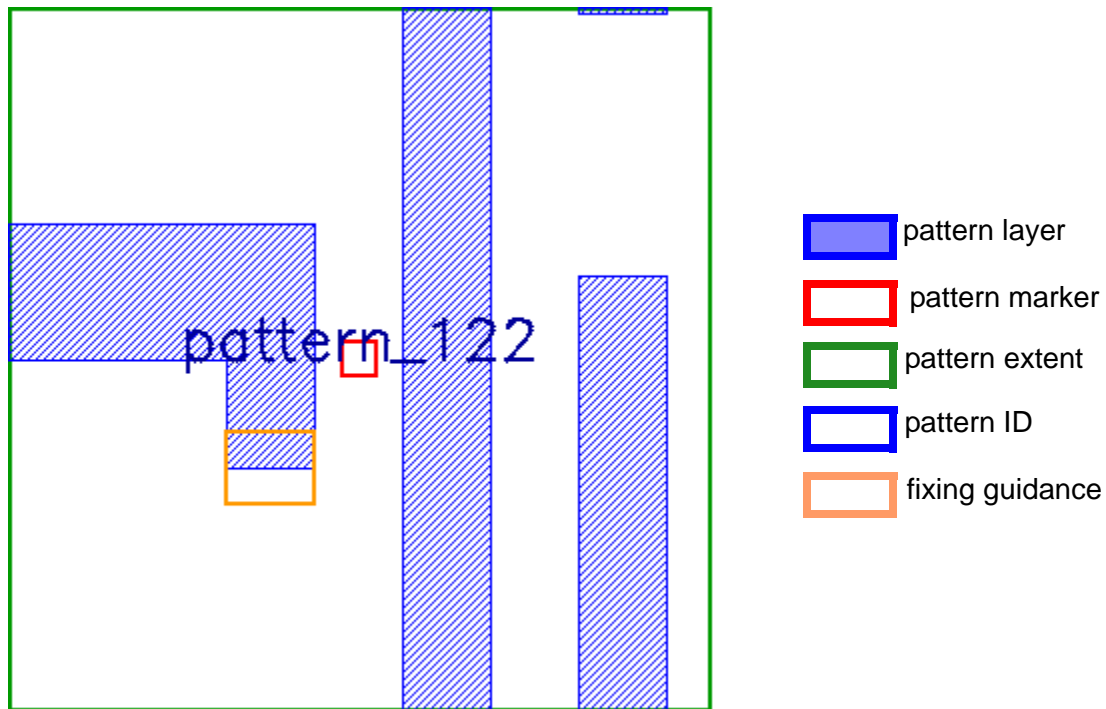
In addition to the default ADR correction, the IC Validator tool also enables you to implement user-defined fix guidance. The steps for implementing fix guidance in the pattern matching flow are:

1. Specify fix guidance as the optional pattern markers during pattern library creation. A single pattern can have multiple guidance.

The pattern in [Figure 9-5](#) shows a single user-defined fix guidance that to add to a rectangle at the end of the L-shape polygon.



Figure 9-5 User-Defined Fix Guidance



Example 9-2 Attaching Fix Guidance During Pattern Library Creation

```
//Layer Assignment
metal : polygon_layer = assign ( {{3,0}} );
pattern_marker : polygon_layer = assign ( {{1,0}} );
pattern_extent : polygon_layer = assign ( {{2,0}} );
pattern_add : polygon_layer = assign ( {{4,0}} );
optional_markers: list of polygon_layer =
    {pattern_extent,pattern_marker,pattern_add};

//Create Pattern Library
pattern_learn (
    pattern_library_name = "met",
    pattern_layers = {metal},
    pattern_marker = marker_layer,
    optional_pattern_markers = optional_markers,
    pattern_text_id = text_layer,
    pattern_extent = pattern_extent,
    pattern_fuzziness = PM_EXACT,
    pattern_reflect = true,
    pattern_rotate = true
);
```

2. Retrieve and pass the fix guidance after pattern matching.

You use the `optional_markers()` function to retrieve the fix guidance and the `milkyway_route_directives()` function to pass the fixing guidance to the router.

### Example 9-3 Retrieving and Passing Fix Guidance

```
//Output hotspot marker layer
hotspot_marker = pattern_match (
    pattern_library_name = "met",
    pattern_layers = {metal_layer_list[i]});

//Retrieve all the optional pattern marker layers defined in step 1
optional_markers = optional_pattern_markers("met", hotspot_marker);

//Retrieve the add shape layer
pattern_add = optional_markers[2];

//Pass fixing guidance to IC Compiler
milkyway_route_directives(
    display_marker = optional_markers[1],
    group_marker = optional_markers[0],
    error_layers = { {optional_markers[1]} },
    route_directives = { {pattern_add, dummy_sub} } );
```

### 3. Run ADR within the IC Compiler tool for hotspot detection and fixing.

Write an IC Compiler TCL script to execute ADR in the `icc_shell`, or use the IC Compiler GUI to execute the commands.

### Example 9-4 IC Compiler TCL Script

```
open_mw_lib ./design
copy_mw_cel -from eco_route_routeopt -to pm_flow_add
open_mw_cel pm_flow_add

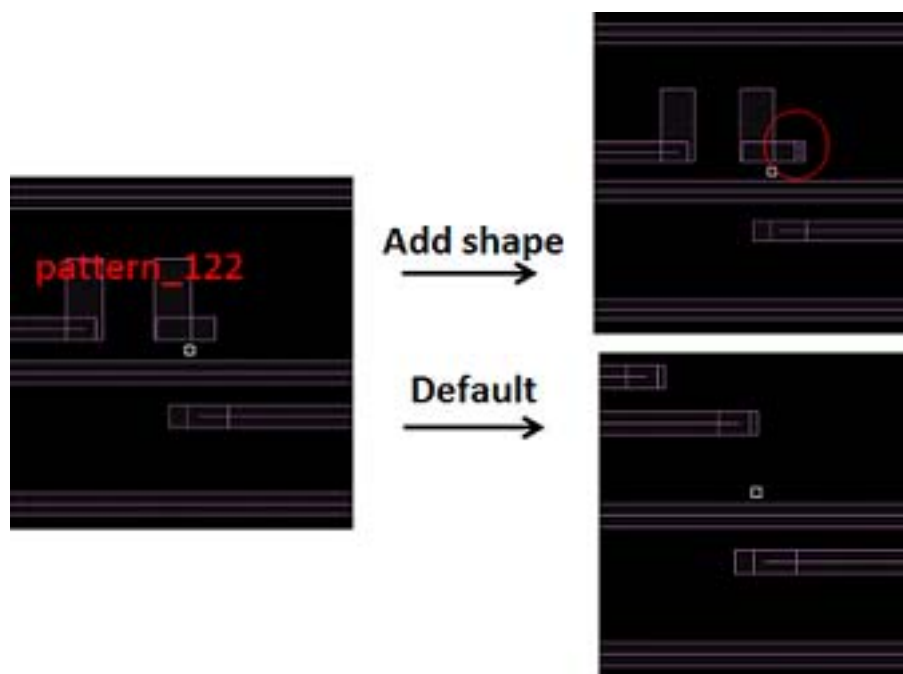
set_physical_signoff_options -exec icv -drc_runset {/runset/ \
match_add.rs}
signoff_drc -run_dir {./signoff_drc_run} -ignore_child_cell_errors

signoff_autofix_drc -run_dir {./signoff_autofix_drc_run} \
-init_drc_error_db signoff_drc_run

save_mw_cel -as pm_flow_add
close_mw_cel
```

The IC Compiler tool starts with user-defined fix guidance based on a defined order. In [Figure 9-6](#), you see the corrected results of pattern\_122 by the user-defined guidance and the default ADR flow, respectively.

Figure 9-6 Example of Hotspot Correction





# A

## IC Validator Architecture

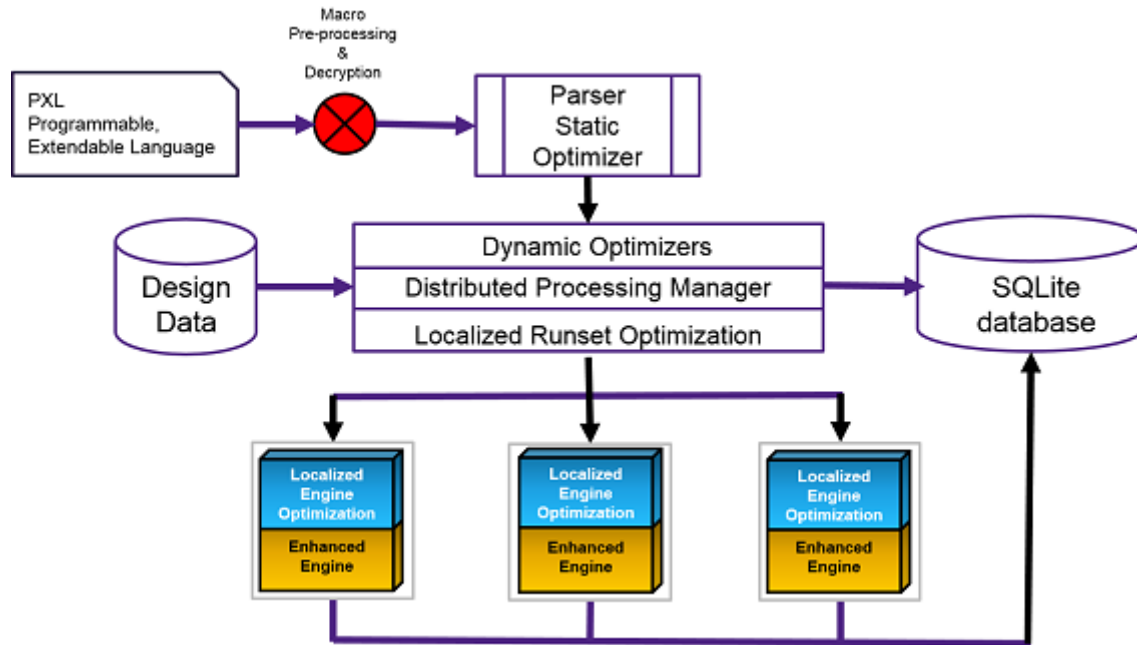
---

*This appendix shows the basic IC Validator architecture.*

## Basic Architecture

Figure A-1 shows the basic IC Validator architecture.

Figure A-1 Basic IC Validator Architecture



# B

## LVL Utility

---

*This appendix discusses the Layout-Versus-Layout (LVL) utility and the QuickLVL utility that are available with the IC Validator tool.*

The LVL utility is described in the following sections:

- [Description](#)
- [Command-Line Options](#)
- [Using the OpenAccess Library Format](#)
- [Layer Files](#)
- [Comparing a Subset of Layers](#)
- [Handling of Text](#)
- [LVL Output](#)
- [Use Models](#)

The QuickLVL utility is described in the following sections:

- [QuickLVL Utility](#)
- [Cell-Level Quick LVL](#)

- [Differences Between LVL and QuickLVL](#)
- [Auto-Incremental DRC Focused on Differing Layers and Regions](#)



---

## Description

The LVL utility allows you to

- Generate an assign file from a single library.
- Compare two layouts of a design using either an XOR-based comparison or a NOT-based comparison. The errors that are found can be viewed in VUE using the *cell1\_vs\_cell2.vue* file. See the [IC Validator VUE User Guide](#) for more information.

Note:

The LVL utility supports text with any text character, except the NULL character.

For the NOT-based comparison,

- The baseline layer is compared using a NOT operation with the check library layer.
- The check library layer is compared using a NOT operation with the baseline layer.

The errors from each NOT operation are shown separately in the *cell1\_vs\_cell2.LVL\_ERRORS* file.

The LVL utility supports input in the following layout formats:

- GDSII
- Milkyway
- OASIS
- OpenAccess
- NDM

Note:

The LVL utility supports text with any text character, except the NULL character.

---

## Command-Line Options

The LVL utility command-line syntax is

```
icv_lv1 library1 library2 -c cell | -c1 cell1 -c2 cell2 [options]
```

[Table B-1](#) describes the basic command-line options. [Table B-2](#) defines the command-line options available for distributed processing.

Table B-1 Syntax for the LVL Utility

Syntax	Description
<code>-64</code>	Runs the utility with 64-bit coordinates support.
<code>-af filename</code>	Name of the file where the assign layers in the library are stored. This file is generated by the LVL utility, and it has the same syntax as an IC Validator runset ASSIGN section. Include a path if you want the file in a directory that is not in the current directory.
<code>-c cell</code>	Cell name of both libraries, <i>library1</i> and <i>library2</i> . <sup>1</sup> The cell name is also needed for generating an assign file with the <code>-af</code> option.
<code>-cl cell1</code>	Cell name of the first library, <i>library1</i> . <sup>1</sup>
<code>-c2 cell2</code>	Cell name of the second library, <i>library2</i> . <sup>1</sup>
<code>-elpc num</code>	Sets the error limit for each type of check. The default is 100.
<code>-error_limit_per_check num</code>	<p>Note:</p> <p>Setting this command-line option to <code>ERROR_LIMIT_MAX</code> causes all errors up to a tool-defined maximum limit to be stored. The limit can be set above the tool-defined maximum limit by giving a specific number. However, setting the limit above the tool-defined maximum might result in decreased performance and increased disk usage for large numbers of errors, and therefore is not recommended. Setting it to 0 suppresses storage of all errors.</p>
<code>-flatten</code>	Flattens all output to the top cell.
<code>-h</code>	Displays the usage message.
<code>-hlic</code>	Forces use of shared licensing scheme. See <a href="#">“Shared Licensing” on page 2-4</a> .
<code>-ignore_text_case</code>	Instructs the utility to ignore case when comparing text. The default behavior is that text comparison considers case.
<code>-lf filename</code> <code>-layer_file filename</code>	Layer file for both libraries that is used for layer mapping, layer merging, and layer naming. It must contain a valid PXL runset ASSIGN section. <sup>2</sup> Include the path if the library is not in the current directory. See <a href="#">“Layer Files” on page B-15</a> for more information.
<code>-lf1 filename</code> <code>-layer_file1 filename</code>	Layer file for the first library. <sup>2</sup> Always use this command-line option along with <code>-lf2</code> . Include the path if the library is not in the current directory. See <a href="#">“Layer Files” on page B-15</a> for more information.

Table B-1 Syntax for the LVL Utility (Continued)

Syntax	Description
<code>-lf2 filename</code> <code>-layer_file2 filename</code>	Layer file for the second library. <sup>2</sup> Always use this command-line option along with <code>-lf1</code> . Include the path if the library is not in the current directory. See <a href="#">“Layer Files” on page B-15</a> for more information.
<code>-oa_color_mapping option</code>	Specifies how to use the locked and unlocked anchor bit for the OpenAccess color layer mapping. The selections for <i>option</i> are <ul style="list-style-type: none"> <li><code>ignore</code>. Uses the color mappings for which the locked or unlocked keyword is not specified, and ignores the color mappings for which locked or unlocked is specified.</li> <li><code>strict</code>. Uses the color mappings for which the locked or unlocked keyword is specified, and ignores the color mappings for which locked or unlocked is not specified.</li> <li><code>full</code>. Sets all shapes that are read with color to locked. The mapping for this setting is the same as the mapping for the <code>compatibility</code> setting.</li> <li><code>compatibility</code>. Follows the industry standard. Ignores color mappings for which the <code>colorAnnotate</code> keyword is located after the color specification. When no color mapping rule matches a color shape, the mapping is used without color.</li> </ul>
<code>-of options_file</code>	Optional file containing an IC Validator runset OPTIONS section. This file can contain any of the OPTIONS section functions, including the <code>gds_options()</code> , <code>milkyway_options()</code> , and <code>oasis_options()</code> functions.
<code>-rsi</code>	Reports input library information. Supports only GDSII and OASIS input formats. Reports <ul style="list-style-type: none"> <li>GDSII last modified and last accessed</li> <li>md5sum computed with <code>md5sum -b file</code></li> <li>format of the input library</li> <li>path to the input library</li> <li>command used for the library input</li> </ul> Enumerators must be in a comma-separated list, and the entire list must be within quotation marks. See the <code>report_streamfile_information</code> argument of the <a href="#">run_options()</a> function. Note: The <code>-rsi</code> command-line option overrides its runset setting in the <code>run_options()</code> function.

Table B-1 Syntax for the LVL Utility (Continued)

Syntax	Description
<code>-ilic</code>	Forces use of native licensing scheme. See <a href="#">“Native Licensing” on page 2-3</a> .
<code>library1</code>	<p>Baseline library name. Include the path if the library is not in the current directory. When one library is specified with the <code>-af</code> option, the LVL utility generates an assign file.</p> <p>You cannot generate an assign file and compare two layouts in one run of the LVL utility.</p> <p>The following examples show how to specify input layouts for various formats:</p> <ul style="list-style-type: none"> <li>• GDSII. <code>/path/to/file.gds</code></li> <li>• OASIS. <code>/path/to/file.oas</code></li> <li>• Milkyway. <code>/library/path/LIBRARY_NAME</code></li> <li>• OpenAccess. <code>/path/to/lib.defs/LIBRARY_NAME</code></li> <li>• NDM. <code>/library/path/LIBRARY_NAME</code></li> </ul>
<code>library2</code>	<p>Check library name. Include the path if the library is not in the current directory. Do not specify this file when using the <code>-af</code> command-line option.</p> <p>You cannot generate an assign file and compare two layouts in one run of the LVL utility.</p> <p>The following examples show how to specify input layouts for various formats:</p> <ul style="list-style-type: none"> <li>• GDSII. <code>/path/to/file.gds</code></li> <li>• OASIS. <code>/path/to/file.oas</code></li> <li>• Milkyway. <code>/library/path/LIBRARY_NAME</code></li> <li>• OpenAccess. <code>/path/to/lib.defs/LIBRARY_NAME</code></li> <li>• NDM. <code>/library/path/LIBRARY_NAME</code></li> </ul>
<code>-mf filename</code> <code>-map_file filename</code>	Specifies the layer mapping file applied to both libraries. See <a href="#">“Mapping Layers” on page B-16</a> for more information.
<code>-mf1 filename</code> <code>-map_file1 filename</code>	Specifies the layer mapping file applied to the first library. See <a href="#">“Mapping Layers” on page B-16</a> for more information.
<code>-mf2 filename</code> <code>-map_file2 filename</code>	Specifies the layer mapping file applied to the second library. See <a href="#">“Mapping Layers” on page B-16</a> for more information.

*Table B-1 Syntax for the LVL Utility (Continued)*

Syntax	Description
<code>-not</code>	Instructs the utility to do a two NOT-based comparison rather than an XOR-based comparison.
<code>-oa_dm4</code>	Uses OpenAccess shared libraries that are compatible with DM4 plug-ins.
<code>-oa_dm5</code>	Uses OpenAccess shared libraries that are compatible with DM5 plug-ins. The default is <code>-oa_dm5</code> .
<code>-outside</code>	Instructs the utility to do a two OUTSIDE-based comparison rather than an XOR-based comparison. When the <code>not_interacting()</code> function is executed, any data that touches is not considered for this operation. Use the <code>-report_touch</code> command-line option along with the <code>-outside</code> option to allow touches to be considered as not interacting.
<code>-rt ALL   POINT</code> <code>-report_touch ALL   POINT</code>	Specifies how point touches are treated. This option is used with the <code>-outside</code> option. <ul style="list-style-type: none"> <li><code>ALL</code>. Treat polygon line touches and edge point touches as outside.</li> <li><code>POINT</code>. Treat point touches as outside.</li> </ul>
<code>-sel "layer1[-layer1max] [ ,dtype1[-dtype1max]] [...]"</code> <code>-select_edge_layers "layer1[-layer1max] [ ,dtype1[-dtype1max]] [...]"</code>	Specifies the edge layers and datatypes that are used in the comparison. Ranges can be used. If a datatype is not specified, it defaults to all datatypes. <sup>3</sup>
<code>-sn layer_name1 ...</code> <code>-select_names layer_name1 ...</code>	Specifies, by name, the layers on which the utility is run.
<code>-spl "layer1[-layer1max] [ ,dtype1[-dtype1max]] [...]"</code> <code>-select_polygon_layers "layer1[-layer1max] [ ,dtype1[-dtype1max]] [...]"</code>	Specifies the polygon layers and datatypes that are used in the comparison. Ranges can be used. If a datatype is not specified, it defaults to all datatypes. <sup>3</sup>

*Table B-1 Syntax for the LVL Utility (Continued)*

Syntax	Description
<code>-suppress_empty_layer_errors</code>	Suppresses the reporting of empty-layer error polygons.
<code>-stl "layer1[-layer1max] [,dtype1[-dtype1max]] [...]"</code>	Specifies the text layers and datatypes that are used in the comparison. Ranges can be used. If a datatype is not specified, it defaults to all datatypes. <sup>3 4</sup>
<code>-select_text_layers "layer1[-layer1max] [,dtype1[-dtype1max]] [...]"</code>	
<code>-text</code>	Instructs the utility to compare text between two layouts. See <a href="#">“Handling of Text” on page B-19</a> for more information. When this command-line option is not used, only geometry is compared, not text.
<code>-uel "layer1[-layer1max] [,dtype1[-dtype1max]] [...]"</code>	Specifies the edge layers and datatypes that are excluded from the comparison. Ranges can be used. If a datatype is not specified, it defaults to all datatypes. <sup>3</sup>
<code>-unselect_edge_layers "layer1[-layer1max] [,dtype1[-dtype1max]] [...]"</code>	
<code>-un layer_name1 ... -unselect_names layer_name1 ...</code>	Specifies by name the layers that are excluded when the utility is run.
<code>-upl "layer1[-layer1max] [,dtype1[-dtype1max]] [...]"</code>	Specifies the polygon layers and datatypes that are excluded from the comparison. Ranges can be used. If a datatype is not specified, it defaults to all datatypes. <sup>3</sup>
<code>-unselect_polygon_layers "layer1[-layer1max] [,dtype1[-dtype1max]] [...]"</code>	
<code>-utl "layer1[-layer1max] [,dtype1[-dtype1max]] [...]"</code>	Specifies the text layers and datatypes that are excluded from the comparison. Ranges can be used. If a datatype is not specified, it defaults to all datatypes. <sup>3 4</sup>
<code>-unselect_text_layers "layer1[-layer1max] [,dtype1[-dtype1max]] [...]"</code>	

Table B-1 Syntax for the LVL Utility (Continued)

Syntax	Description
-V	Prints the utility version.
<ol style="list-style-type: none"> <li>1. If the libraries have the same cell name, provide the cell name with the <code>-c</code> command-line option. If the libraries have different cell names, you must provide both cell names using the <code>-c1</code> and <code>-c2</code> command-line options.</li> <li>2. The layer file can be a full runset. The LVL utility, however, ignores everything but the ASSIGN section.</li> <li>3. If you are comparing two OpenAccess libraries, you can use the layer name and purpose name instead of layer number and datatype number.</li> <li>4. Text matches only when both location and text string match. Text differences appear as an X in VUE. Text comparison is only a cell level operation. For cases of hierarchy mismatch, if a baseline cell is not present in the check layer, then all the baseline text in the cell is reported as an error; if a check layer cell is not present in the baseline, a single-cell mismatch error is reported.</li> </ol>	

The command-line options described in [Table B-2](#) are available for distributed processing with the LVL utility.

**Note:**

Use the `-dp` option for GDSII and OASIS files, and use the `-threads` option with Milkyway, OpenAccess, and NDM and libraries.

Table B-2 Distributed Processing Command-Line Options for the LVL Utility

Option	Definition
-dp	<p>Networks a distributed run that is defined either through the use of a distributed processing (DP) options configuration file or additional command line options. The default distributed processing options configuration file is <code>dp_options.conf</code> in the current working directory.</p> <p>The default distributed processing mode is <code>-dp2</code>, using only as many processors as there are DP clients on a machine for dynamic threading. If you want to use all the processors on the machines for dynamic threading, use the <code>-turbo</code> option.</p> <p>You can use this option with GDSII and OASIS files. You cannot use it with Milkyway, OpenAccess, and NDM libraries.</p> <p>See <a href="#">“Distributed Processing Options Configuration File” on page 1-40</a> and <a href="#">“Using Multiple Distributed Processing Hosts” on page 1-39</a> for more information. This information is applicable to the LVL utility, in addition to the IC Validator tool.</p>

**Table B-2** *Distributed Processing Command-Line Options for the LVL Utility (Continued)*

Option	Definition
<code>-dp#_of_hosts</code>	<p>Networks a distributed run using the specified number of distributed processing hosts, all running on the local host. For example,</p> <pre>% icv_lvl ... -dp4</pre> <p>The default is two hosts, <code>-dp2</code>, using only as many processors as there are DP clients on a machine for dynamic threading. If you want to use all the processors on the machines for dynamic threading, use the <code>-turbo</code> option.</p> <p>You can use this option with GDSII and OASIS files. You cannot use it with Milkyway, OpenAccess, and NDM libraries.</p> <p>See <a href="#">“Distributed Processing Options Configuration File” on page 1-40</a> and <a href="#">“Using Multiple Distributed Processing Hosts” on page 1-39</a> for more information. This information is applicable to the LVL utility, in addition to the IC Validator tool.</p>
<code>-dpcache path</code>	<p>Specifies a local group directory for all distributed processing hosts in a network distributed run. For example,</p> <pre>% icv_lvl ... -dpcache /SCRATCH/user/_group</pre> <p>The <code>-dp</code> option must be specified before the <code>-dpcache</code> option. For example,</p> <pre>% icv ... -dp -dpcache /SCRATCH/user/_group \ -dphosts host1 host2</pre>
<code>-dphosts host1 [local_group_dir] ... hostn [local_group_dir]</code>	<p>Specifies a set of distributed processing hosts to be used in a network distributed run. To run multiple distributed processing hosts on a multicore or multiprocessor system, a host name can be specified multiple times. The <code>-dp</code> option must be specified before the <code>-dphosts</code> option. For example,</p> <pre>% icv_lvl ... -dp -dphosts host1 host1 host2</pre> <p>For each distributed processing host, an optional local group directory can be specified. This directory must be specified as an absolute path accessible on the distributed processing host. For example,</p> <pre>% icv_lvl ... -dp -dphosts host1 \ /SCRATCH/user/_group host1 \ /SCRATCH/user/_group host2 \ /SCRATCH/user/_group</pre> <p>You can use this option with GDSII and OASIS files. You cannot use it with Milkyway, OpenAccess, and NDM libraries.</p>



**Table B-2** *Distributed Processing Command-Line Options for the LVL Utility (Continued)*

Option	Definition
<code>-dpm</code>	Runs the <code>dp_monitor</code> utility. This utility displays the status of an LVL utility distributed processing run.
<code>-dpopts [filename]</code>	<p>Uses <i>filename</i> as the distributed processing options configuration file. Using a file for the distributed processing options allows you to maintain different distributed processing configurations, selecting the desired configuration at the start of the run. The default distributed processing options configuration file is <code>dp_options.conf</code> in the current working directory.</p> <p>The <code>-dp</code> option must be specified before the <code>-dpopts</code> option. For example,</p> <pre>% icv_lvl ... -dp -dpopts my_dp_configuration.txt</pre> <p>You can use this option with GDSII and OASIS files. You cannot use it with Milkyway, OpenAccess, and NDM libraries.</p> <p>See <a href="#">“Distributed Processing Options Configuration File” on page 1-40</a> for more information. This information is applicable to the LVL utility, in addition to the IC Validator tool.</p>
<code>-threads integer</code>	<p>Specifies the number of threads used by the IC Validator tool when it is called by the LVL utility. The default is 2.</p> <p>The <code>-threads</code> option cannot be used with the DP options, such as <code>-dp</code>.</p>
<code>-turbo</code>	<p>Specifies to use all the processors on the machines for dynamic threading. When this option is not used, only as many processors as there are DP clients on a machine are used for dynamic threading.</p> <p>You can use this option with GDSII and OASIS files. You cannot use it with Milkyway, OpenAccess, and NDM libraries.</p>

## Using the OpenAccess Library Format

The LVL utility command-line options are useful when you are comparing layouts that are in the OpenAccess format. The OpenAccess command-line options are:

- `-oalm filename`  
`-oa_layer_map filename`
- `-oalm1 filename`  
`-oa_layer_map1 filename`

- `-oalm2 filename`  
`-oa_layer_map2 filename`
- `-oacm cellmap`  
`-oa_cell_map cellmap`
- `-oacm1 cellmap1`  
`-oa_cell_map1 cellmap1`
- `-oacm2 cellmap2`  
`-oa_cell_map2 cellmap2`
- `-oa_color_mapping ignore | strict | full | compatibility`
- `-oa_color_mapping1 ignore | strict | full | compatibility`
- `-oa_color_mapping2 ignore | strict | full | compatibility`
- `-oav viewname`  
`-oa_view viewname`
- `-oav1 viewname`  
`-oa_view1 viewname`
- `-oav2 viewname`  
`-oa_view2 viewname`

If you are comparing an OpenAccess library to another OpenAccess library, then you do not need to specify a layer mapping file. The LVL utility matches the layer/purpose pairs and runs the comparison. However, if you are comparing an OpenAccess library to a GDSII, OASIS, Milkyway, or NDM library, then a layer mapping file is needed to map the OpenAccess layer/purpose pair to a layer/datatype in the GDSII, Milkyway, Oasis, or NDM library. For example,

```
% icv_lvl lib.defs/OA_LIB gds_lib.gds -c top -oalm layer_map.txt
```

As shown in this example,

- The OpenAccess library (OA\_LIB), which is listed first, is compared to the GDSII library (gds\_lib.gds).
- The lib\_defs file lists all libraries needed for the design including the OpenAccess library. The file name must be lib.defs. If the file is not located in the current directory, specify the path leading to it followed by the filename (libs.def). For example, datapath1/testcase1/lib.defs/OA\_LIB.

The following is an example of the contents of the lib.defs file:

```
DEFINE analogLib ./PDK/lib/analogLib
ASSIGN analogLib libMode shared
DEFINE basic ./PDK/lib/basic
```

```

ASSIGN basic libMode shared
DEFINE device_lib ./PDK/lib/device_lib
ASSIGN device_lib libMode shared
DEFINE OA_LIB ./OA_LIB

```

---

## OpenAccess Layer Mapping File

If you are comparing an OpenAccess database to another OpenAccess database, then you do not need to provide a layer mapping file. The LVL utility matches the layer/purpose pairs and runs a comparison.

However, if you are comparing an OpenAccess database to a GDSII, OASIS, Milkyway, or NDM library, then you need to provide a layer mapping file to map the OpenAccess layer/purpose pairs to a layer/datatype in the GDSII, OASIS, Milkyway, or NDM file.

If both OpenAccess libraries use the same layer mapping file, specify:

```
-oalm filename
```

If each input OA database has a unique layer mapping file, specify the databases individually:

```

-oalm1 filename1
-oalm2 filename2

```

For more information, see the definition of the `layer_mapping_file` argument of the `openaccess_options()` function in the *IC Validator Reference Manual*.

The following is an example of the OpenAccess layer mapping format.

#OA Layer Name	OA Purpose Name	Stream layer	Stream Datatype
Layer1	drawing	1	0
Layer2	drawing	2	1
Layer3	drawing	3	2
Layer4	drawing	4	0
Layer4	boundary	4	1

---

## Cell Mapping

When you are comparing an OpenAccess database to a different format, you might need to provide the cell mapping to prevent flattening. The OpenAccess library specifies unique cells by a library/cell/view triplet. The IC Validator tool and the LVL utility must assign a unique cell name to each unique library/cell/view triplet. The LVL utility flattens hierarchies with mismatched cell names until the two hierarchies match. Using a cell mapping file can ensure that cells are named consistently between the two libraries being compared, and prevent this flattening.

For more information, see the definition of the `cell_mapping_file` argument of the `openaccess_options()` function in the *IC Validator Reference Manual*.

If both OpenAccess libraries use the same cell mapping, specify:

```
-oacm cellmap
```

If each input OpenAccess database has a unique mapping, specify the databases individually:

```
-oacm1 cellmap1
-oacm2 cellmap2
```

The following is an example cell mapping file.

#Library Name	Cell Name	View Name	Stream Name
des_lib	nand2	layout	nand2_1
sdes_lib	nand2	layout	nand2_2
des_lib	cell_nu	layout	cell_nu
sdes_lib	cell_nu	layout	cell_new

---

## Error Output Format for OpenAccess Baseline Library

The error output written to an OpenAccess library in the current working directory named `cell1_vs_cell2`.

The error data in the output library uses layer names from the baseline library, when possible.

The general format for layer name when a layer file is not given is

`LlayerPurpose`

The output purpose name is Xor, BaselineNotCheck, or CheckNotBaseline, depending on the LVL method chosen.

The following is an example output.

```
Checking LxxxPyyy_Baseline XOR LaaaPbbb_Check, output to (LxxxPyyy; Xor)
Checking LxxxPyyy_Baseline NOT LaaaPbbb_Check, output to (LxxxPyyy; BaselineNotCheck)
Checking LaaaPbbb_Check NOT LxxxPyyy_Baseline, output to (LaaaPbbb; CheckNotBaseline)
```

---

## Setting the Cell View Name

When specifying an OpenAccess library on the command line, you might need to set the view name to something other than the default view of “layout” for the top cell.

If both OpenAccess libraries use the same view name, specify

```
-oav viewname
```

If each input OpenAccess database has a different view name for the top cell, specify the databases individually:

```
-oav1 viewname1
-oav2 viewname2
```

---

## Layer Files

By default, all layers of the layout databases are compared. However, by using command-line options, you can run the utility on a subset of these layers. You can also specify mapping files to use instead of mapping files included internally in the Milkyway and NDM libraries.

Layers are mapped before layers are selected or filtered using command-line options. For example, if `-lf layerfile` and `-mf mapfile` are used at the same time, then layer numbers and datatypes are first mapped using the *mapfile* file before interpreting the *layerfile* file.

---

### Selecting a Subset of Layers

When you specify a layer file using the `-lf` command-line option or using both the `-lf1` and `-lf2` command-line options, only those layers that appear in the layer file are compared. No comparison is performed on layer numbers that do not appear in any of the assign statements.

These restrictions apply to the layer files:

- When a layer/datatype pair is used multiple times, any redundant assigns are ignored.

In this example, the third assignment, L3D0, is ignored.

```
L1D0 = assign({{1,0}});
L2D0 = assign({{2,0}});
L3D0 = assign({{1,0}});
```

- The LVL utility stops with an error if a layer file contains multiple definitions of a layer name.

In this example, layer A is defined twice. Therefore, the LVL utility reports an error. You must remove one of the definitions.

```
A = assign({{1,0}});
B = assign({{2,0}});
A = assign({{10,0}});
```

---

## Mapping Layers

The mapping file can be included internally as part of the Milkyway and NDM libraries. Internal mapping files are used by default. Using the `-mf`, `-mf1`, and `-mf2` command-line options, you can map layers using an external file. When you specify a mapping file on the command-line, it is used instead of the internal mapping file.

### Note:

To map layers using the `-mf`, `-mf1`, and `-mf2` command-line options on an OpenAccess library, you must also use the `-oalm`, `-oalm1`, and `-oalm2` command-line options to convert LayerName/Purpose string pairs to LayerNumber/Datatype numbered pairs.

The mapping file can be in two of three formats: the Milkyway format, the NDM format, or the IC Validator mapping file format. See the *Milkyway Database Application Note* and the *Library Data Preparation for IC Compiler User Guide*, which are available on SolvNet, for information about the Milkyway and NDM formats. See [“Layer Mapping” in Chapter 1](#) for information about the IC Validator mapping file format.

The `-lf`, `-lf1`, `-lf2`, `-spl`, `-upl`, `-sel`, `-uel`, `-stl`, `-utl`, `-sn`, and `-un` command-line options operate on layers after they have been mapped.

If do not want to use internal mapping files, you can turn them off using the mapping file command-line options. Use `-mf1 ""` to turn it off for the first library. Use `-mf2 ""` to turn it off for the second library. Use `-mf ""` to turn it off for both libraries.

For example, consider two input libraries:

- `one.gds`. This library contains data on layer 1, datatype 0.
- `two.gds`. This library contains data on layer 2, datatype 0, which has the same data as on layer 1 in the `one.gds` library.

The `mf2` mapping file has

```
D 2:0 1 0
```

The LVL utility command is

```
icv_lvl one.gds two.gds -mf2 mf2 -c top
```

The `-mf2` command-line option allows you to map layer 2 data in the `two.gds` library to layer 1 data. The result is that both libraries end up having their data on the same layer and the LVL comparison is clean.

Alternatively, you could use layer files to accomplish the same thing. For example,

- `file1` contains
 

```
Metal1 = assign({{1, 0}});
```

- file2 contains

```
Metall = assign({{2, 0}});
```

The LVL utility command is

```
icv_lvl one.gds two.gds -lf1 file1 -lf2 file2 -c top
```

The first method, which uses mapping files, is easier if you already have IC Validator layer mapping files for IC Validator runs that can be used with the LVL utility mapping file command-line options.

---

## Comparing a Subset of Layers

You can use filtering command-line options to compare a subset of polygon and edge layers, with or without a layer file. These rules apply:

- You cannot use a select option with the corresponding unselect (exclude) option.
- You cannot use filtering by name with filtering by number.

The filtering options are shown in [Table B-3](#). See the following sections for more information.

*Table B-3 Filtering Options*

Data filtering category	By name	By number
Select	-sn	-spl -sel -stl
Unselect	-un	-upl -uel -utl

**Note:**

You cannot use both the `-sn` and `-un` options in the same run of the utility.

You cannot use both the `-spl` and `-upl` options in the same run of the utility.

You cannot use both the `-sel` and `-uel` options in the same run of the utility.

You cannot use both the `-stl` and `-utl` options in the same run of the utility.

---

## Comparing by Name

You can use command-line options to limit the polygon and edge layers being compared when a layer file is provided using the `-lf` command-line option or using both the `-lf1` and `-lf2` command-line options. This filtering is referred to as filtering by name.

The command-line options used to limit by name the layers being compared are:

- `-sn layer_name1 ...`  
`-select_names layer_name1 ...`
- `-un layer_name1 ...`  
`-unselect_names layer_name1 ...`

In this example, the LVL utility runs on the layers POLY, NWELL, and METAL1 layers:

```
-sn POLY NWELL METAL1
```

In this example, the LVL utility runs on all the layers except METAL2:

```
-un METAL2
```

---

## Comparing by Number

You can use command-line options to limit the layers being compared when no layer file is provided. This filtering is referred to as filtering by number. Polygon, edge, and text layers are filtered separately.

The command-line options used to limit by number the layers being compared are:

- `-sel "layer1[-layer1max] [,dtype1[-dtype1max]] [...]"`  
`-select_edge_layers "layer1[-layer1max] [,dtype1[-dtype1max]] [...]"`
- `-uel "layer1[-layer1max] [,dtype1[-dtype1max]] [...]"`  
`-unselect_edge_layers "layer1[-layer1max] [,dtype1[-dtype1max]] [...]"`
- `-spl "layer1[-layer1max] [,dtype1[-dtype1max]] [...]"`  
`-select_polygon_layers "layer1[-layer1max] [,dtype1[-dtype1max]] [...]"`
- `-upl "layer1[-layer1max] [,dtype1[-dtype1max]] [...]"`  
`-unselect_polygon_layers "layer1[-layer1max] [,dtype1[-dtype1max]] [...]"`
- `-stl "layer1[-layer1max] [,dtype1[-dtype1max]] [...]"`  
`-select_text_layers "layer1[-layer1max] [,dtype1[-dtype1max]] [...]"`
- `-utl "layer1[-layer1max] [,dtype1[-dtype1max]] [...]"`  
`-unselect_text_layers "layer1[-layer1max] [,dtype1[-dtype1max]] [...]"`



In the following example, the LVL utility runs on the following polygon layers:

- 1 through 4, all datatypes
- 5, datatypes 0 through 7
- 6, all datatypes
- 7 and 8, datatypes 0 through 7

```
-spl 1-4 5,0-7 6 7-8,0-7
```

In the next example, the LVL utility runs on all polygon layers and datatypes except for datatypes 5, 6, and 7 in polygon layers 1, 2, and 3.

```
-upl 1-3,5-7
```

In this example, the LVL utility runs on the following edge layers:

- 1 through 5, datatypes 0 through 7
- 6 through 8, datatypes 5 through 7

```
-sel 1-5,0-7 6-8,5-7
```

In the next example, the LVL utility runs on all the edge layers and datatypes except edge layer 2 and datatypes 5 and 6 in edge layer 4:

```
-uel 2 4,5-6
```

---

## Handling of Text

The `-text` command-line option instructs the LVL utility to compare text between two layouts. If one or both databases have multiple text at the same coordinate, (x1,y1), the utility matches as many text strings as possible at (x1,y1) between the two databases. The errors are reported as follows:

- If there is only one unmatched text at (x1, y1) in each database:

```
Violations Found:TEXT_XOR
-----
Structure (Position x, y)  BaselineText  CheckText
-----
A          (x1, y1)        base_text    check_text
```

- If there is one unmatched baseline text and no unmatched checked text, or vice versa:

```
Violations Found:TEXT_XOR
-----
Structure (Position x, y)  BaselineText  CheckText
-----
A          (x1, y1)        base_text
```

- If there are multiple unmatched text at (x1,y1) in the baseline database and at least one unmatched text at (x1,y1) in the check library:

```
Violations Found:TEXT_XOR
-----
Structure (Position x, y)  BaselineText  CheckText
-----
A          (x1, y1)        base_text1    check_text1
A          (x1, y1)        base_text1    check_text2
A          (x1, y1)        base_text2    check_text1
A          (x1, y1)        base_text2    check_text2
A          (x1, y1)        base_text3    check_text1
A          (x1, y1)        base_text3    check_text2
```

In the preceding example, there are three unmatched baseline text and two unmatched checked text. In general, all possible combinations of unmatched text are reported.

---

## LVL Output

The LVL utility errors are saved in the files described in [“Error Files” on page B-21](#)”.

The output library has the same format as the baseline library. The default output cell is from the baseline library. VUE overlays differences to the baseline library.

- If the input design files are in either the GDSII or OASIS format and LVL runs with the layer file, differences are reported on layers that are listed in the LVL\_ERRORS file, which might be different than the baseline layer or check layout.

For example, a difference between baseline layer {31,0} and check layer {131, 0}, which is mapped to the same layer using the LVL command-line options, can be reported in output layer {1, 23}.

- If the input design files are in either the GDSII or OASIS format and LVL runs with the layer mapping file and without a layer file, differences are reported on a layer determined by the name generated based on the mapping file.

For example, if the layer name is L10D15, the difference is reported in layer {10, 15}.

- If the input design files are in either the GDSII or OASIS format and LVL runs without the layer mapping file or layer file, differences between the two layers are reported on the same layer of the output library.

For example, a difference between baseline layer {31, 0} and check layer {31, 0} is reported in output layer {31, 0}.

If one of the input files (for example, OpenAccess or Milkyway) is not in the GDSII or OASIS format, the differences are reported on the layers that are listed in the LVL\_ERRORS file, which might be different from the baseline layer or check layout.

For example, a difference between baseline layer {31,0} and check layer {131, 0}, which is mapped to the same layer using the LVL command-line options, can be reported in output layer {1, 23}.

---

## Exit Status

The exit status values of the `icv_lvl` executable are shown in [Table B-4](#).

*Table B-4 Exit Status Values*

Exit status value	Definition
0	Ran successfully and found no differences.
1	Ran successfully and found differences.
negative number	An error occurred.

In this example, the `icv_lvl` executable ran successfully and found no differences:

```
% icv_lvl lib1 lib2 -c cell
% echo $?
0
```

---

## Error Files

[Table B-5](#) describes the error files that are generated and stored in the current working directory.

Note:

You can specify the maximum number of errors for each type of check using the `-elpc` option. See [Table B-1](#).

The LVL utility also provides screen output that contains real-time results from each XOR or NOT operation.

*Table B-5 Error Files*

File name	Description
<code>cell1_vs_cell2.LVL_ERRORS</code>	Error file containing all comparison errors segregated by layer.

Table B-5 Error Files (Continued)

File name	Description
<code>cell1_vs_cell2.LVL_RESULTS</code>	Results file that gives a brief description of each step of the run and indicates whether there are any errors.  When there is an error, the results file points you to the specific <code>cell.RESULTS</code> file that is located in the run details directory of the failed run. See the <a href="#">run_options()</a> function in the <i>IC Validator Reference Manual</i> for information about specifying the <code>cell.RESULTS</code> file.
<code>cell1_vs_cell2.gds</code> <code>cell1_vs_cell2.oasis</code> <code>cell1_vs_cell2/</code>	Output libraries containing comparison differences in GDSII, OASIS, Milkyway, OpenAccess, and NDM formats.  For OpenAccess libraries, the <code>lib.defs</code> file in the current directory is created or updated with the output library definition.
<code>lvl_details/pydb</code>	SQLite database (the error database, PYDB) containing the comparison errors segregated by each XOR or NOT operation performed by the LVL utility.
<code>lvl_details/</code> <code>cell1_vs_cell2.lvl_sum</code>	Summary file containing the following information for each run of the LVL utility: <ul style="list-style-type: none"> <li>• The <code>icv_lvl</code> executable command line.</li> <li>• Indication of success or failure.</li> <li>• Purpose of the run.</li> <li>• Elapsed time and peak memory of the run.</li> </ul>
<code>cell1_vs_cell2.vue</code>	File with overlay differences to the baseline library. This file allows you to view the errors using VUE.

## Output Directories

[Table B-6](#) describes the directories that are created by the LVL utility. They are in the current working directory.

Table B-6 Output Directories

Directory name	Description
<code>lvl_details/compare_layers/</code>	Directory containing additional diagnostic information about the comparison process between the baseline library and the check library.

Table B-6 Output Directories (Continued)

Directory name	Description
lv_details/extract_layers/	Directory containing summary information regarding preprocessing of the check library.
lv_details/layer_list1/	Automatically generated layer list for the baseline library if a layer list is not present. Summary information of layer list creation is included for diagnostic purposes.
lv_details/layer_list2/	Automatically generated layer list for the check library if a layer list is not present. Summary information of layer list creation is included for diagnostic purposes.

## Use Models

This section shows two simple use models of the LVL utility.

- The following use model compares the baseline library with the check library using two NOT operations. All the layers in *library1* are compared with all the layers in *library2*.

```
icv_lvl library1 library2 -c cell -not
```

- The following use model generates a layer file from the input library using *cell* as the top cell. The output file contains an assign statement for each layer/datatype found in the input library under the *cell* subtree.

```
icv_lvl library -af output_file -c cell
```

---

## QuickLVL Utility

This section describes two use models in the Quick LVL utility allow you to quickly read two layouts and compare the layout elements cell by cell.

- Cell-level Quick LVL: Standalone run of Quick LVL.
- Auto-Incremental DRC: The IC Validator tool calls Quick LVL first to determine differences with respect to the baseline. Results are passed to the tool in an IC Validator readable internal format without user intervention.

The Quick LVL utility supports input in the following layout formats:

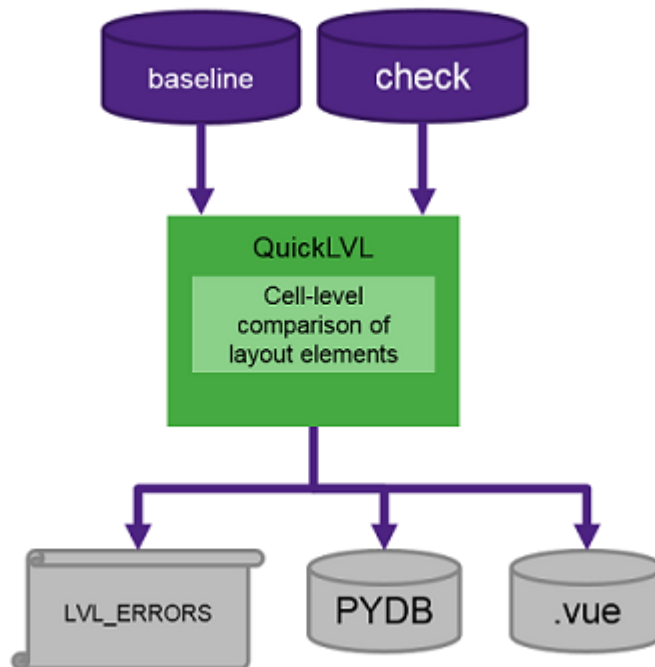
- OASIS
- GDSII

Note:

The Quick LVL utility supports text with any text character, except the NULL character.

[Figure B-1](#) shows the Quick LVL utility.

*Figure B-1 Quick LVL Flowchart*



## Cell-Level Quick LVL

The Quick LVL utility reads two layouts and compares layout elements cell by cell. The results are stored in the error database, PYDB. This mode produces output files in the LVL\_ERRORS and VUE formats.

The command-line syntax is

```
icv_lvl library1 library2
```

[Table B-7](#) describes the command-line options for the QuickLVL utility

**Table B-7** Syntax for the Quick LVL Utility

Syntax	Description
<code>-c cell</code>	Cell name of both libraries, <i>library1</i> and <i>library2</i> . <sup>1</sup> The cell name is also needed for generating an assign file with the <code>-af</code> option.
<code>-c1 cell1</code>	Cell name of the first library, <i>library1</i> . <sup>1</sup>
<code>-c2 cell2</code>	Cell name of the second library, <i>library2</i> . <sup>1</sup>
<code>-quick</code>	Performs cell-level Quick LVL.
<code>-transform [RX,] [0 90 180 270 DX,DY]</code>	Indicates that the top cell of the check layout is transformed as described on the command line. <b>RX</b> : Indicates reflection around X-axis 0 90 180 270; counterclockwise rotation angle <b>dx dy</b> ; translation in X and Y directions, in microns. See the following examples: <ul style="list-style-type: none"> <li><code>-transform 90,0,0</code></li> <li><code>-transform RX,90,1,-1.2</code></li> <li><code>-transform RX</code></li> </ul>
<code>-64</code>	Runs the utility with 64-bit coordinates support.
<code>-threads integer</code>	Specifies the number of threads used by the IC Validator tool when it is called by the LVL utility. The default is 2.  The <code>-threads</code> option cannot be used with the DP options, such as <code>-dp</code> .
<code>-v</code>	Prints the utility version.
<code>-text</code>	Instructs the utility to compare text between two layouts. See <a href="#">“Handling of Text” on page B-19</a> for more information. When this command-line option is not used, only geometry is compared, not text.

*Table B-7 Syntax for the Quick LVL Utility (Continued)*

Syntax	Description
<code>-ignore_text_case</code>	Instructs the utility to ignore case when comparing text. The default behavior is that text comparison considers case.
<code>-mf filename</code> <code>-map_file filename</code>	Specifies the layer mapping file applied to both libraries. See <a href="#">“Mapping Layers” on page B-16</a> for more information.
<code>-mf1 filename</code> <code>-map_file1 filename</code>	Specifies the layer mapping file applied to the first library. See <a href="#">“Mapping Layers” on page B-16</a> for more information.
<code>-mf2 filename</code> <code>-map_file2 filename</code>	Specifies the layer mapping file applied to the second library. See <a href="#">“Mapping Layers” on page B-16</a> for more information.
<code>baseline_layout</code>	Baseline library name. Include the path if the library is not in the current directory.  The following examples show how to specify input layouts for each format: <ul style="list-style-type: none"> <li>• OASIS. /path/to/file.oas</li> <li>• GDSII. /path/to/file.gds</li> </ul>
<code>check_layout</code>	Check library name. Include the path if the library is not in the current directory.  The following examples show how to specify input layouts for each format: <ul style="list-style-type: none"> <li>• OASIS. /path/to/file.oas</li> <li>• GDSII. /path/to/file.gds</li> </ul>
<code>-suppress_empty_layer_errors</code>	Suppresses the reporting of empty-layer error polygons.
<code>-spl "layer1[-layer1max] [,dtype1[-dtype1max]] [...]"</code> <code>-select_polygon_layers "layer1[-layer1max] [,dtype1[-dtype1max]] [...]"</code>	Specifies the polygon layers and datatypes that are used in the comparison. Ranges can be used. If a datatype is not specified, it defaults to all datatypes. <sup>3</sup>
<code>-oa_dm4</code>	Uses OpenAccess shared libraries that are compatible with DM4 plug-ins.



Table B-7 Syntax for the Quick LVL Utility (Continued)

Syntax	Description
<code>-oa_dm5</code>	Uses OpenAccess shared libraries that are compatible with DM5 plug-ins. The default is <code>-oa_dm5</code> .
<code>-upl "layer1[-layer1max] [,dtype1[-dtype1max]] [...]"</code>	Specifies the polygon layers and datatypes that are excluded from the comparison. Ranges can be used. If a datatype is not specified, it defaults to all datatypes. <sup>3</sup>
<code>-unselect_polygon_layers "layer1[-layer1max] [,dtype1[-dtype1max]] [...]"</code>	
<code>-sel "layer1[-layer1max] [,dtype1[-dtype1max]] [...]"</code>	Specifies the edge layers and datatypes that are used in the comparison. Ranges can be used. If a datatype is not specified, it defaults to all datatypes. <sup>2</sup>
<code>-select_edgen_layers "layer1[-layer1max] [,dtype1[-dtype1max]] [...]"</code>	
<code>-uel "layer1[-layer1max] [,dtype1[-dtype1max]] [...]"</code>	Specifies the edge layers and datatypes that are excluded from the comparison. Ranges can be used. If a datatype is not specified, it defaults to all datatypes. <sup>3</sup>
<code>-unselect_edgen_layers "layer1[-layer1max] [,dtype1[-dtype1max]] [...]"</code>	
<code>-stl "layer1[-layer1max] [,dtype1[-dtype1max]] [...]"</code>	Specifies the text layers and datatypes that are used in the comparison. Ranges can be used. If a datatype is not specified, it defaults to all datatypes. <sup>3 3</sup>
<code>-select_text_layers "layer1[-layer1max] [,dtype1[-dtype1max]] [...]"</code>	
<code>-utl "layer1[-layer1max] [,dtype1[-dtype1max]] [...]"</code>	Specifies the text layers and datatypes that are excluded from the comparison. Ranges can be used. If a datatype is not specified, it defaults to all datatypes. <sup>3 4</sup>
<code>-unselect_text_layers "layer1[-layer1max] [,dtype1[-dtype1max]] [...]"</code>	

1. If the libraries have the same cell name, provide the cell name with the `-c` command-line option. If the libraries have different cell names, you must provide both cell names using the `-c1` and `-c2` command-line options.
2. If you are comparing two OpenAccess libraries, you can use the layer name and purpose name instead of layer number and datatype number.
3. Text matches only when both location and text string match. Text differences appear as an X in VUE. Text comparison is only a cell level operation. For cases of hierarchy mismatch, if a baseline cell is not present in the check layer, then all the baseline text in the cell is reported as an error; if a check layer cell is not present in the baseline, a single-cell mismatch error is reported.

## Differences Between LVL and QuickLVL

Cell-level Quick LVL provides a list of differences between the `baseline_layout` and `check_layout` library names. This differs from traditional full-layout-LVL in the following major points:

- The results are reported at the cell level.
- Element-wise differences are reported; not geometrical differences. For example, if a rectangle is bigger in one layout, both rectangle elements are reported entirely, not their geometrical difference.
- If a transformation is defined on the command line, the tool transforms the coordinates in the checked layout's top cell before trying to match the baseline layout.
- If an element is represented in different formats (for example, path versus rectangle or AREF versus a set of SREFs) in two layouts, they are not matched, even if they geometrically represent identical shapes. These elements are reported as “missing” and “new” elements.
- QuickLVL internally tries to match elements between layouts. Elements with certain similarities are reported as “different” between the baseline and the check. Other elements are reported as missing in the check layout or as new shapes in the baseline layout.
- It is possible to have some false-positive results. This is traded for faster speed.

This tool provides the following output from the `LVL_ERRORS` file:

```
Path Mismatch (5:0)
  Baseline shapes different in Check ..... 25 violations found.

Placement Mismatch
  Baseline shapes missing in Check ..... 20 violations found.
  New shapes in Check not in Baseline ..... 150 violations found.

Path Mismatch (10:0)
  New shapes in Check not in Baseline ..... 75 violations found.
```

---

## Auto-Incremental DRC Focused on Differing Layers and Regions

With the `-idrc_i` command-line option, the IC Validator tool first runs cell-level Quick LVL to compare the current layout to the baseline layout. Differences between these layouts are used to improve IC Validator run time by focusing on layers and windows that contain differences. The window ambit can be adjusted by setting

```
incremental_options(window_ambit = ...);
```

### Note:

Using options of the `incremental_options()` function other than `window_ambit` is not recommended in the Auto-Incremental DRC flow.

### Note:

The Auto-Incremental DRC flow requires the baseline layout and checked layout to have the same top-cell name.

### Note:

Results from an Auto-Incremental DRC run might not be a complete replacement for a full signoff run.

The Auto-Incremental DRC flow supports the baseline layout in the following layout formats:

- OASIS
- GDSII

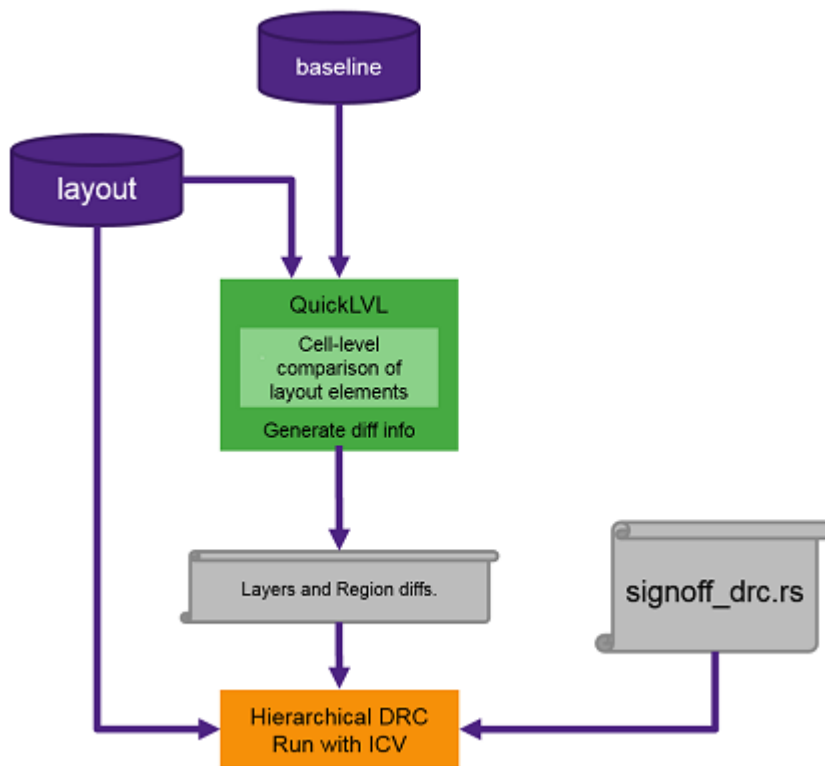
[Table B-8](#) describes the command-line options specific to an Auto-Incremental DRC run.

*Table B-8 Syntax for the IC Validator Options for Auto-Incremental DRC Flow*

Syntax	Description
<code>-idrc_i baseline_layout</code>	Determines different regions and layers of the (original) input layout using Quick LVL. The IC Validator run focuses on these regions and layers.
<code>-idrc_c top_cell_name</code>	Indicates the baseline top-cell name.
<code>-idrc_f OASIS   GDSII</code>	Specifies the layout format of the baseline layout, which must include the <code>-idrc_i baseline_layout</code> command-line option. Otherwise it has no impact.

---

[Figure B-2](#) shows the Auto-Incremental DRC Flow.

*Figure B-2 Auto-Incremental DRC Flow*

# C

## Layout Integrity Management

---

*This appendix describes the layout integrity management feature that is available with the IC Validator tool.*

The layout integrity management feature is described in the following sections:

- [Using Layout Integrity Management](#)
- [Layout Integrity Management Examples](#)
- [The icv\\_lidb Utility](#)
- [The icv\\_lidb\\_report Utility](#)

---

## Using Layout Integrity Management

Layout integrity management allows you to verify the integrity of golden intellectual property (IP), standard cell layout, or any other layout cell. It provides a convenient way to verify that certain layout cells have not changed without doing a full layout-versus-layout comparison. The `icv_lidb` utility creates a layout integrity database (LIDB) from an input layout. The resultant LIDB can then be used in an IC Validator run to check layout cells as the design is read. The `icv_lidb_report` utility views the content of an LIDB.

Note:

Although false matches are extremely unlikely, this flow is not a substitute for a full verification run before tapeout.

For each cell, the LIDB contains:

- One checksum per layer of all geometric data and text on the layer.
- One checksum per unique child cell name of all references to the child cell.

Each cell in the LIDB can have multiple sets of checksums. The design cell is considered a match if it matches any set of checksums for that cell stored in the LIDB. This matching allows a single LIDB to serve equivalent cells with different checksums. For example, different layout formats can generate slightly different checksums, or a tool in the flow can change the data, such as by merging adjacent or overlapping shapes. In these cases, a new set of checksums can be merged into an existing LIDB so that any version of the cell passes the layout integrity check.

There are three ways that cells in a design can be checked for layout integrity:

- Use the `icv_lidb` utility to validate an input layout against an existing LIDB without doing a full IC Validator run. The utility matches layout cells to a set of checksums in the LIDB by cell name.
- Use the `layout_integrity_by_cell()` function in an IC Validator runset to check cells by name. A design cell is considered a match if its calculated checksums match any set of checksums for a cell of the same name in any input LIDB specified in the `layout_integrity_options()` function.
- Use the `layout_integrity_by_marker_layer()` function in an IC Validator runset to check cells by marker layer. A design cell is considered a match if its calculated checksums match any set of checksums for a cell with data present on the marker layer in any input LIDB specified in the `layout_integrity_options()` function.

Layout integrity mismatches found during an IC Validator run are stored in the error database, are reported in the `LAYOUT_ERRORS` file, and are visible in VUE. There are no specific coordinates associated with the errors, so there is no highlighting of these errors in VUE.

The results of layout integrity checking can be used to control the action taken on DRC errors subsequently found in those cells. The errors can be discarded, reported as normal, or classified using DRC classification.

---

## Layout Integrity Management Examples

The following section provides several examples of the layout integrity management feature:

- [Basic Example](#)
- [Running on a Different Layout Format](#)
- [Reporting of Mismatches From a Merged LIDB](#)
- [Marker Layer Check](#)
- [Mixed Milkyway and GDSII Flow](#)

---

### Basic Example

The following example shows basic layout integrity checking by cell name. First, create an LIDB from the input layout:

```
icv_lidb -i EX_ADDER_3.GDS -o orig.lidb
```

Next, add the following to your runset, and the IC Validator tool runs as planned.

```
layout_integrity_options(  
    databases = {  
        { db_name = "orig.lidb" }  
    }  
);  
  
{  
    @ "Layout Integrity Check";  
    layout_integrity_by_cell(  
        cells = {"*"}  
    );  
}
```

For this run, no layout integrity mismatches are reported in the LAYOUT\_ERRORS file because you are running from the same input layout from which the LIDB is generated. However, you can see that the layout integrity checking is performed by examining the run\_details/lidb.log file.

```

-----
                        Layout Integrity Processing Log
                        11/08/2010 08:12:50
-----

Processing layout cell VIA
  Cell check:
    Result      = Matched
    Matched Cell = VIA
    Database     = /home/user/orig.lidb

Processing layout cell INV
  Cell check:
    Result      = Matched
    Matched Cell = INV
    Database     = /home/user/orig.lidb
...

```

Also, the layout integrity status of each cell that was checked can be seen in the `run_details/cell.tree0` file.

```

CELL = AD4FUL
      Cell Status           = USED
      GDSII Library         = MAINLIB
      Layout Integrity Check = Passed Completely

```

There are three possible values for the layout integrity check status in the `cell.tree0` file.

- Passed Completely. The cell and all cells placed underneath it were checked and found to be a match.
- Passed. The cell itself passed the layout integrity check, but at least one child cell was not checked or failed the check.
- Failed. No layout integrity match was found for the cell.

---

## Running on a Different Layout Format

The following example shows the results of running on a different layout format of the same design. Using a different layout format can result in different checksum data. Common reasons for this difference are:

- Some formats, such as Milkyway, can have a cell boundary layer while others, such as GDSII, do not.
- Some formats represent cell placements differently. For example, OASIS does not directly support the concept of an AREF. Instead arrayed cell placements are represented as SREFs with a repetition placing the cell multiple times. The IC Validator tool, however, might interpret these cell placements as AREFs internally.



- There are several ways in which geometric data can differ between formats while remaining physically equivalent. Point ordering, merging of overlapping shapes, and collinear point removal are examples of how geometric data can differ. The layout integrity management feature checks the original input layout as is, without attempting to transform it in any way.

When the IC Validator run from the previous example is given an equivalent Milkyway input layout, layout integrity mismatches because of the cell boundary layer are reported.

```
...
Layout Integrity Check
  layout_integrity_by_cell:layer_mismatch ..... 9 violations found.
...

runset1.rs:17:layout_integrity_by_cell:layer_mismatch
- - - - -
Structure Layer      LIDB      Golden Layout
- - - - -
AD4FUL      (255;0) orig.lidb
VIA          (255;0) orig.lidb
ADFULAH     (255;0) orig.lidb
POLYHD      (255;0) orig.lidb
INV          (255;0) orig.lidb
INVH        (255;0) orig.lidb
DGATE       (255;0) orig.lidb
CONT        (255;0) orig.lidb
TGATE       (255;0) orig.lidb
```

When running on an equivalent OASIS layout, layout integrity mismatches because of cell placements being interpreted as AREFs are reported.

```
...
Layout Integrity Check
  layout_integrity_by_cell:placement_mismatch .... 6 violations found.
...

runset1.rs:17:layout_integrity_by_cell:placement_mismatch
- - - - -
Structure Child Cell LIDB      Golden Layout
- - - - -
AD4FUL      ADFULAH      orig.lidb
AD4FUL      VIA          orig.lidb
ADFULAH     INV          orig.lidb
ADFULAH     INVH         orig.lidb
DGATE       TGATE        orig.lidb
DGATE       VIA          orig.lidb
```

Because these three layouts are equivalent, they can be merged into a single LIDB. The LIDB passes on all three formats.

```
icv_lidb -i EX_ADDER_3 EX_ADDER_3.GDS EX_ADDER_3.oas -o merged.lidb
```

---

## Reporting of Mismatches From a Merged LIDB

Because a single cell can have multiple sets of checksums, some cells checksums can be matched but not others. A cell is considered a match only if all layer and placement checksums are a match for one set of checksums in the LIDB. The following example shows the differences between layout integrity error reporting in different scenarios in a merged LIDB.

```
validate_layout.rs:14:layout_integrity_by_cell:layer_mismatch
- - - - -
Structure Layer LIDB          Golden Layout
- - - - -
TOP          (1;0) merged.lib 2.gds
TOP          (2;0) merged.lib 1.gds
TOP          (3;0) merged.lib
```

In the preceding example, there is no complete match found for the cell TOP. Layer 1 matches cell TOP in the layout 1.gds, but does not match 2.gds, so the mismatch is reported against the golden layout 2.gds. Layer 2 matches cell TOP in 2.gds, but does not match 1.gds, so the mismatch is reported against the golden layout 1.gds. Layer 3 does not match any golden layout, so that column is left blank.

Cell placement mismatches for the same scenario are shown in the following example.

```
validate_layout.rs:15:layout_integrity_by_cell:placement_mismatch
- - - - -
Structure Child Cell LIDB      Golden Layout
- - - - -
TOP          A          merged.lib LIB2
TOP          B          merged.lib LIB1
TOP          C          merged.lib
```

In the preceding example, no complete match found for the cell TOP. Placements of child cell A match the placements in cell TOP of LIB1, but not LIB2, so the mismatch is reported against LIB2. Placements of child cell B match the placements in cell TOP of LIB2 but not LIB1, so the mismatch is reported against LIB1. Placements of child cell C do not match any golden layout, so that column is left blank.

---

## Hierarchical Mode

The IC Validator tool has two modes of operation for layout integrity checking:

- **Cell level.** Each cell is considered individually without regard to whether any children in the hierarchy passed or failed the check. This is the default mode.
- **Hierarchical.** The status of child cells in the hierarchy is considered. Any layout integrity mismatch of a child cell is reported also as a placement mismatch in the parent cell. This reporting continues all the way up the hierarchy.

Mismatches are only reported in cells specified in the check function:

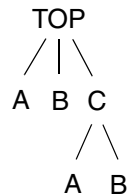
- `layout_integrity_by_cell()`. The cells are specified by the `cells` argument.
- `layout_integrity_by_marker_layer()`. The cells are those that contain a polygon on the layer specified by the `marker_layer` argument.

---

## Hierarchical Cell Check Example

This example shows behavior of checks for the `layout_integrity_by_cell()` function with hierarchical mode. Consider the design hierarchy shown in [Figure C-1](#).

Figure C-1 Design Hierarchy for Hierarchical Cell Check Example



In this example, there is a difference in a layer shape in cell A from the checksum in the layout integrity database.

Note:

Although the example uses placed cells with matching cell names, this is not required in hierarchical mode. Cells placed under the check cell must have the same content but not the same cell name.

The following code checks all cells in hierarchical mode:

```

layout_integrity_by_cell(
    cells = { "*" },
    processing_mode = HIERARCHICAL
);

```

The difference is reported in A, C, and TOP. The output in the LAYOUT\_ERRORS file is:

```

test.rs:15:layout_integrity_by_cell:layer_mismatch
-----
Structure Layer  LIDB          Golden Layout
-----
A              (31;0) test.lidb

test.rs:15:layout_integrity_by_cell:placement_mismatch
-----
Structure Child Cell LIDB          Golden Layout
-----
TOP            A          test.lidb
TOP            C          test.lidb
C              A          test.lidb

```

The following code checks cells C and TOP, but not cell A:

```
layout_integrity_by_cell(
    cells = { "*", "!A" },
    processing_mode = HIERARCHICAL
);
```

The difference is still reported in C and TOP. The hierarchical mode specifies that everything below a particular cell being checked is also be considered. The output is:

```
test.rs:15:layout_integrity_by_cell:placement_mismatch
-----
Structure Child Cell LIDB      Golden Layout
-----
TOP        A        test.lidb
TOP        C        test.lidb
C          A        test.lidb
```

The run\_details/lidb.log file contains additional details that indicate the status of each cell in the hierarchy. In this example, cell A is processed, but there is no "Cell check" entry because cell A was not specified in the `layout_integrity_by_cell()` function call:

```
-----
                        Layout Integrity Processing Log
                        04/23/2014 06:13:36
-----

Processing layout cell A
  Database: test.lidb
    All Layers: No hierarchically equivalent cells found

Processing layout cell B
  Database: test.lidb
    All Layers: Hierarchically equivalent cells found
  Cell check (hierarchical):
    Result      = Matched
    Matched Cell = B
    Database    = test.lidb

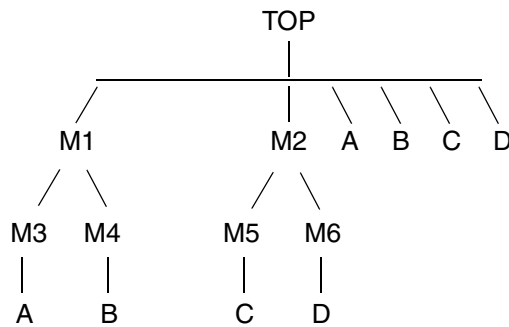
Processing layout cell C
  Database: test.lidb
    All Layers: No hierarchically equivalent cells found
  Cell check (hierarchical):
    Result      = No Match Found

Processing layout cell TOP
  Database: test.lidb
    All Layers: No hierarchically equivalent cells found
  Cell check (hierarchical):
    Result      = No Match Found
```

## Hierarchical Marker Layer Check Example

This example shows behavior of checks for the `layout_integrity_by_marker_layer()` function with hierarchical mode. Consider the design hierarchy shown in [Figure C-2](#).

Figure C-2 Design Hierarchy for Hierarchical Layer Check Example



In this example, only cells M1-M6 have data on the marker layer. Cell B has a difference on layer 3. Cell M5 has a difference on layer 4.

### Note:

Although the example uses placed cells with matching cell names, this is not required in hierarchical mode. Cells placed under the check cell must have the same content but not the same cell name.

The following shows several hierarchical mode checks:

```
// This reports mismatches in M1, M2, M4, and M5
layout_integrity_by_marker_layer(
    marker_layer = {1},
    processing_mode = HIERARCHICAL
);

// This check only reports mismatches in M2 and M5 since the only
// layer with a difference being checked is layer 4
layout_integrity_by_marker_layer(
    marker_layer = {1},
    check_layers = { {2}, {4} },
    processing_mode = HIERARCHICAL
);

// This check only reports mismatches in M1 and M4 since the only
// layer with a difference being checked is layer 3
layout_integrity_by_marker_layer(
    marker_layer = {1},
    check_layers = { {3} },
    processing_mode = HIERARCHICAL
);

// This check reports no mismatches, since no layers being checked
```

```
// contain a difference
layout_integrity_by_marker_layer(
    marker_layer = {1},
    check_layers = { {2} },
    processing_mode = HIERARCHICAL
);
```

The mismatches are reported in the LAYOUT\_ERRORS file:

```
test2.rs:17:layout_integrity_by_marker_layer:cell_mismatch
- - - - -
Structure Marker Layer
- - - - -
M1          (1;0)
M2          (1;0)
M4          (1;0)
M5          (1;0)

test2.rs:22:layout_integrity_by_marker_layer:cell_mismatch
- - - - -
Structure Marker Layer
- - - - -
M2          (1;0)
M5          (1;0)

test2.rs:28:layout_integrity_by_marker_layer:cell_mismatch
- - - - -
Structure Marker Layer
- - - - -
M1          (1;0)
M4          (1;0)
```

Details on the status of each cell are reported in the run\_details/lidb.log file:

```
Processing layout cell C
  Database: test.lidb
    All Layers: Hierarchically equivalent cells found
    Layers (3;0): Hierarchically equivalent cells found
    Layers (4;0) (2;0): Hierarchically equivalent cells found
    Layers (2;0): Hierarchically equivalent cells found
  Marker Layer Check:
    Result      = No Data on Marker Layer

Processing layout cell A
  Database: test.lidb
    All Layers: Hierarchically equivalent cells found
    Layers (3;0): Hierarchically equivalent cells found
    Layers (4;0) (2;0): Hierarchically equivalent cells found
    Layers (2;0): Hierarchically equivalent cells found
  Marker Layer Check:
    Result      = No Data on Marker Layer

Processing layout cell B
```

```
Database: test.lidb
  All Layers: No hierarchically equivalent cells found
  Layers (3;0): No hierarchically equivalent cells found
  Layers (4;0) (2;0): Hierarchically equivalent cells found
  Layers (2;0): Hierarchically equivalent cells found
Marker Layer Check:
  Result      = No Data on Marker Layer

Processing layout cell D
Database: test.lidb
  All Layers: Hierarchically equivalent cells found
  Layers (3;0): Hierarchically equivalent cells found
  Layers (4;0) (2;0): Hierarchically equivalent cells found
  Layers (2;0): Hierarchically equivalent cells found
Marker Layer Check:
  Result      = No Data on Marker Layer

Processing layout cell M6
Database: test.lidb
  All Layers: Hierarchically equivalent cells found
  Layers (3;0): Hierarchically equivalent cells found
  Layers (4;0) (2;0): Hierarchically equivalent cells found
  Layers (2;0): Hierarchically equivalent cells found
Marker Layer Check:
  Result      = Matched
  Marker Layer = (1;0)
  Matched Cell = M6
  Database     = test.lidb

Processing layout cell M5
Database: test.lidb
  All Layers: No hierarchically equivalent cells found
  Layers (3;0): Hierarchically equivalent cells found
  Layers (4;0) (2;0): No hierarchically equivalent cells found
  Layers (2;0): Hierarchically equivalent cells found
Marker Layer Check:
  Result      = Matched
  Marker Layer = (1;0)
  Matched Cell = M5
  Database     = test.lidb

Processing layout cell M4
Database: test.lidb
  All Layers: No hierarchically equivalent cells found
  Layers (3;0): No hierarchically equivalent cells found
  Layers (4;0) (2;0): Hierarchically equivalent cells found
  Layers (2;0): Hierarchically equivalent cells found
Marker Layer Check:
  Result      = Matched
  Marker Layer = (1;0)
  Matched Cell = M4
  Database     = test.lidb
```

```

Processing layout cell M3
Database: test.lidb
  All Layers: Hierarchically equivalent cells found
  Layers (3;0): Hierarchically equivalent cells found
  Layers (4;0) (2;0): Hierarchically equivalent cells found
  Layers (2;0): Hierarchically equivalent cells found
Marker Layer Check:
  Result      = Matched
  Marker Layer = (1;0)
  Matched Cell = M3
  Database     = test.lidb

Processing layout cell M2M
Database: test.lidb
  All Layers: No hierarchically equivalent cells found
  Layers (3;0): Hierarchically equivalent cells found
  Layers (4;0) (2;0): No hierarchically equivalent cells found
  Layers (2;0): Hierarchically equivalent cells found
Marker Layer Check:
  Result      = Matched
  Marker Layer = (1;0)
  Matched Cell = M2
  Database     = test.lidb

Processing layout cell M1
Database: test.lidb
  All Layers: No hierarchically equivalent cells found
  Layers (3;0): No hierarchically equivalent cells found
  Layers (4;0) (2;0): Hierarchically equivalent cells found
  Layers (2;0): Hierarchically equivalent cells found
Marker Layer Check:
  Result      = Matched
  Marker Layer = (1;0)
  Matched Cell = M1
  Database     = test.lidb

Processing layout cell TOP
Marker Layer Check:
  Result      = No Data on Marker Layer

```

---

## Marker Layer Check

The `layout_integrity_by_marker_layer()` function does not check cells by name, but rather checks any cell with data present on the specified marker layer. It looks for any matching set of checksums for any cell in the input LIDB that also contains data on the marker layer.



```
{
  @ "Layout Integrity Check";
  layout_integrity_by_marker_layer(
    marker_layer = { 8,0 }
  );
}
```

This example checks cells with data on layer 8 as the marker layer. The following example shows a matching cell and a mismatched cell in the run\_details/lidb.log file.

```
-----
                        Layout Integrity Processing Log
                        11/08/2010 09:45:33
-----

Processing layout cell VIA
  Marker Layer Check:
    Result      = No Match Found
    Marker Layer = (8;0)

Processing layout cell INV
  Marker Layer Check:
    Result      = Matched
    Marker Layer = (8;0)
    Matched Cell = INV
    Database     = /home/user/orig.lidb
```

You can see from the `cell.LAYOUT_ERRORS` file that mismatches for the marker layer check are reported on a pass or fail basis. No individual layer or cell reference mismatches are reported.

```
...

Layout Integrity Check
  layout_integrity_by_marker_layer:cell_mismatch ... 3 violations found.
...

runset8.rs:17:layout_integrity_by_marker_layer:cell_mismatch
-----
Structure Marker Layer
-----
AD4FUL      (8;0)
VIA         (8;0)
TGATE       (8;0)
```

If you want to check only specific layers in the marked cells, use the `check_layers` argument.

```
{
  @ "Layout Integrity Check";
  layout_integrity_by_marker_layer(
    marker_layer = { 8,0 },
    check_layers = { {==6}, {==8}, {==10} }
  );
}
```

In the preceding example, if a cell matches on layers 6, 8, and 10, it is considered a complete match.

---

## Mixed Milkyway and GDSII Flow

For a mixed Milkyway and GDSII flow, the `layout_integrity_options()` function specifies the LIDB to be used for checking Milkyway cells. For layout integrity checking to be performed on replacement GDSII or OASIS files, the LIDB must be specified for each replacement library in the `milkyway_merge_library_options()` function.

```
library(
  library_name = "EX_ADDER_3",
  format = MILKYWAY,
  cell = "AD4FUL"
);

milkyway_merge_library_options(
  libraries = {
    {
      library_name = "EX_ADDER_3",
      replacement_libraries = {
        {
          file = "EX_ADDER_3.GDS",
          format = GDSII,
          layout_integrity = {
            { db_name = "gds.lidb" }
          }
        }
      }
    }
  }
);

layout_integrity_options(
  databases = {
    { db_name = "milkyway.lidb" }
  }
);

{
  @ "Layout Integrity Check";
```

```

    layout_integrity_by_cell(
        cells = {"*"}
    );
}

```

After the IC Validator run, the `run_details/lidb.log` file shows that the Milkyway cell is checked against the `milkyway.lidb` file, while the GDSII cells are checked against the `gds.lidb` file.

```

-----
                                Layout Integrity Processing Log
                                11/08/2010 09:56:43
-----

Processing layout cell AD4FUL
Cell check:
    Result          = Matched
    Matched Cell    = AD4FUL
    Database        = /home/user/milkyway.lidb

Processing layout cell VIA
Cell check:
    Result          = Matched
    Matched Cell    = VIA
    Database        = /home/user/gds.lidb

```

---

## The icv\_lidb Utility

The `icv_lidb` utility manages LIDBs. Use it to:

- Create an LIDB from an input layout.
- Merge multiple LIDBs.
- Quickly validate an input layout using an existing LIDB.

---

## Command-Line Options

The `icv_lidb` utility command-line syntax supports its three simple use models.

- To generate an LIDB from an input layout:

```
icv_lidb -i input_layout -o output_lidb [-c top_cell]
```

- To validate an input layout using an existing LIDB:

```
icv_lidb -i input_layout -validate golden_lidb [-c top_cell]
```

- To merge multiple LIDBs together:

```
icv_lidb -merge lidb1 lidb2... -o output_lidb
```

**Table C-1** describes the command-line options of the utility.

**Table C-1** *icv\_lidb Utility Command-Line Options*

Syntax	Description
<code>-c cell</code>	The top cell of the input layout. If omitted, all cells in the library are processed.
<code>-help</code>	Displays a list of the command-line options and exits.
<code>-holding_cell cell</code>	Optional holding cell. All subcells of the holding cell are processed, but no checksums are calculated for the holding cell itself.
<code>-i input_layout ... input_layout</code>	<p>Input layout name when creating an LIDB or validating a layout against an existing LIDB. If creating an LIDB, you can specify multiple files to incorporate checksums from all layouts into the output LIDB.</p> <p>The following examples show how to specify input layouts for various formats:</p> <ul style="list-style-type: none"> <li>• GDSII: <code>icv_lidb -i /path/to/file.gds</code></li> <li>• OASIS: <code>icv_lidb -i /path/to/file.oas</code></li> <li>• Milkyway: <code>icv_lidb -i /library/path/LIBRARY_NAME</code></li> <li>• OpenAccess: <code>icv_lidb -i /path/to/lib.defs/ LIBRARY_NAME</code></li> </ul>
<code>-lf filename</code>	<p>Selects a subset of all layers and object types (polygons, edges, and text) to be written to the layout integrity database.</p> <p>For example, edges on layer1 are written only to the layout integrity database only if the layer file contains an <code>assign_edge()</code> function that includes layer1. The layer file is stored in the layout integrity database so that any validation runs against this database are restricted to the layers and object types found in the database of the layer file.</p> <p>The layer file must not contain options functions or runset functions. If the file is empty or does not contain assign statements, this command-line option is ignored and the IC Validator tool behaves as if it is not present.</p>
<code>-merge lidb ... lidb</code>	Merges the specified input LIDB into the output database.
<code>-o output_lidb</code>	The output layout integrity database (LIDB) that is generated. If the file exists, it is overwritten.

Table C-1 *icv\_lidb Utility Command-Line Options (Continued)*

Syntax	Description
<code>-oa_dm4</code>	Uses OpenAccess shared libraries that are compatible with DM4 plug-ins.
<code>-oa_dm5</code>	Uses OpenAccess shared libraries that are compatible with DM5 plug-ins. The default is <code>-oa_dm5</code> .
<code>-oa_layer_map layer_mapping_file</code>	Optional OpenAccess layer mapping file for OpenAccess databases. Data on any layers not present in the mapping file is not read. See the <code>layer_mapping_file</code> argument of the <a href="#">openaccess_options()</a> function for information about the mapping file.
<code>-oa_view view_name</code>	Optional view of the top cell for OpenAccess databases.
<code>-of options_file</code>	Optional file containing an IC Validator runset OPTIONS section. This file can contain any of the OPTIONS section functions, including the <code>gds_options()</code> , <code>milkyway_options()</code> , and <code>oasis_options()</code> functions.
<code>-v</code>	Displays the utility version and exits.
<code>-validate golden_lidb</code>	Validates the input layout by performing layout integrity checking against the specified golden LIDB. Results are in the <code>cell.LAYOUT_ERRORS</code> file.

## Exit Status

The exit status values of the `icv_lidb` executable are shown in [Table C-2](#).

Table C-2 *icv\_lidb Utility Exit Status Values*

Exit status value	Definition
0	Ran successfully. If validating an input layout, no differences were found.
1	Ran successfully and found differences.
negative number	An error occurred.

---

## Output Files

The output files of the `icv_lidb` utility are shown in [Table C-3](#).

*Table C-3* `icv_lidb` Utility Output Files

File name	Description
<code>lidb_details/icv_lidb.sum</code>	Summary file containing information about the run. All output from the screen is captured in this file. It contains <ul style="list-style-type: none"><li>• The command-line options used for the run.</li><li>• Information about input and output files.</li><li>• Runtime and memory used.</li><li>• Success or failure indication.</li></ul>
<code>lidb_details/icv_lidb.log</code>	A log of all cells processed for layout validation or database generation runs. For layout validation runs, this file might indicate whether a design cell was a match for a golden LIDB cell. It might also contain details on which golden LIDB cell the design cell matches.
<code>cell.LAYOUT_ERRORS</code>	Error file containing all layout integrity errors found during a layout validation run. If no top cell is given, the file is named <code>cell.LAYOUT_ERRORS</code> .

---

---

## The `icv_lidb_report` Utility

The `icv_lidb_report` utility reports the content of an LIDB. For each set of checksums contained in the database, this utility shows the golden layout path and the user ID and timestamp, and lists the cells with entries. Output is sent to the screen as well as the `icv_lidb_report.sum` summary file in the current directory.

---

## Command-Line Options

The `icv_lidb_report` utility command-line syntax is:

```
icv_lidb_report -i input_lidb
```

[Table C-4](#) describes the command-line options of the utility.

*Table C-4 icv\_lidb\_report Utility Command-Line Options*

Syntax	Description
-help	Displays a list of the command-line options and exits.
-i <i>input_lidb</i>	Input layout integrity database to report.
-v	Displays the utility version and exits.

---

## Output Files

The output files of the `icv_lidb_report` utility are shown in [Table C-5](#).

*Table C-5 icv\_lidb\_report Utility Output Files*

File name	Description
<code>icv_lidb_report.sum</code>	Summary file containing information about the run. All output from the screen is captured in this file. It contains <ul style="list-style-type: none"><li>• The command-line options used for the run.</li><li>• Information about input and output files.</li></ul>





# D

## IC Validator PXL Debugger

---

*The IC Validator PXL Debugger assists you in constructing a runset or part of a runset by allowing you to look at the intermediate stages in each of the design rules or device configuration functions you are writing.*

The PXL Debugger is described in the following sections:

- [Running the PXL Debugger](#)
- [PXL Debugger Commands](#)

## Running the PXL Debugger

To run the PXL Debugger use the `-debug` command-line option with the IC Validator command:

```
% icv -debug runset.rs [-debug_source file]
```

The `-debug_source` command-line option allows you to run the PXL Debugger in batch mode. Specify a file written with the [record](#) command in a previous debug run. The output of the PXL Debugger displays on screen.

When runset execution starts,

- A message on screen indicates the start of the PXL Debugger.
- The run pauses at the first breakpoint. Five lines before and four lines after the executable line are displayed.
- The PXL Debugger prompt (ICVD>>) displays.

For example,

```
Starting ICV Debugger
13 |         t = t + dtol((x * y**2) / z);
14 |     }
15 |     t = dtol(t / x);
16 |     note (" f2(3) = "+ t);
17 | }
--> 18 | xx:integer;
    19 | f2(3);
    20 | note("end. ");
ICVD>>
```

At the prompt enter the action you want the PXL Debugger to take. See the following “[PXL Debugger Commands](#)” section.

Note:

The PXL Debugger is designed to execute PXL code sequentially. If distributed processing options are enabled, a warning is generated and the options are ignored.

The PXL Debugger steps through code in runset and user functions, but not in remote functions.

## PXL Debugger Commands

The IC Validator PXL Debugger has the commands shown in [Table D-1](#).

*Table D-1 PXL Debugger Commands*

Commands <sup>1</sup>	Definition
<code>h[elp]</code>	Print a listing of the PXL Debugger commands.
<code>s[tep]</code>	Step into the function call. See the <a href="#">step</a> command description.
<code>n[ext]</code>	Function is executed all at once. See the <a href="#">next</a> command description.
<code>q[uit]</code>	Quit the PXL Debugger and the IC Validator run. See the <a href="#">quit</a> command description.
<code>c[ont]</code>	Continue until the next breakpoint or the end of run. See the <a href="#">cont</a> command description.
<code>fin[ish]</code>	Execute until the current function returns. See the <a href="#">finish</a> command description.
<code>b[reak]</code> <code>b[reak]</code> <code>[file:]line</code> <code>b[reak]</code> <code>function</code>	Set a breakpoint. The default is at the current file:line. See the <a href="#">break</a> command description.
<code>info b[reak]</code> <code>[number]</code>	Show the breakpoint status. The default is all breakpoints. See the <a href="#">info break</a> command description.
<code>dis[able]</code> <code>[number]</code>	Disable the breakpoint. The default is disable all breakpoints. See the <a href="#">disable</a> command description.
<code>en[able]</code> <code>[number]</code>	Enable the breakpoint. The default is enable all breakpoints. See the <a href="#">enable</a> command description.
<code>d[ele]te</code> <i>number</i>	Delete the specified breakpoint. See the <a href="#">delete</a> command description.
<code>l[ist]</code> <code>l[ist]</code> <code>[file:]line</code> <code>l[ist] function</code>	List the source. The default is the current file and line. See the <a href="#">list</a> command description.
<code>where</code>	Show the function call stack. See the <a href="#">where</a> command description.

Table D-1 PXL Debugger Commands (Continued)

Commands <sup>1</sup>	Definition
<code>pwd</code>	Prints the current working directory of the IC Validator PXL Debugger. For example:  ICVD>> <code>pwd</code> Working directory /remote/data/ICV
<code>p[rint]</code> <code>expression</code>	Print the value of the expression. See the <a href="#">print</a> command description.
<code>record [file]</code>	Save the commands from the current debug session. The default is <code>ICVpid.pdi</code> . See the <a href="#">record</a> command description.
<code>source file</code>	Execute commands from the file. See the <a href="#">source</a> command description.

1. The commands can be abbreviated as shown with the brackets.

---

## break

The `break` command sets breakpoints. There are four ways to set breakpoints:

### 1. `break`

When specified without any options, `break` sets a breakpoint at the current line in the current file.

```

      4 |
      5 | f2 : function (
      6 |     x : integer
      7 | ) returning ans : integer
      8 | {
-->   9 |     ans = x;
     10 |     note("0. ans = " + ans);
     11 | }
     12 |
     13 | xx: integer = 0;
ICVD>> b
Breakpoint #1,  f2() at test.rs:9
ICVD>>
```

### 2. `break line`

Sets a breakpoint at a specific line number in the current file.

```

ICVD>> b 13
Breakpoint #1,  f2() at test.rs:13
ICVD>>
```

### 3. `break file:line`

Sets a breakpoint at the specified line number in the specified file.

```

ICVD>> b test.rs:15
Breakpoint #2,  _MAIN_() at      test.rs:15
ICVD>>
```

### 4. `break function`

Sets the breakpoint in the first executable line within the specified function.

```

      8 | {
      9 |     ans = x;
     10 |     note("0. ans = " + ans);
     11 | }
     12 |
-->   13 | xx: integer = 0;
     14 | xx = f2(3);
     15 | note("1. f2(3)= " + xx);
ICVD>> b f2
Breakpoint #1,  f2() at test.rs:9
ICVD>>
```

This method allows setting breakpoints on overloaded functions. In this example, f2 is overloaded as it can return either an integer or void. Specifying 2 selected option 2; f2 returns an integer.

```
ICVD>> b f2
[0] cancel
[1] all
[2] f2() returning integer, at test.rs:9
[3] f2() returning void, at test.rs:16
> 2
```

In this example, specifying 1 selected option 1; f2 returns an integer and void.

```
Breakpoint #1, f2() at test.rs:9
ICVD>> b f2
[0] cancel
[1] all
[2] f2() returning integer, at test.rs:9
[3] f2() returning void, at test.rs:16
> 1
Breakpoint #2, f2() at test.rs:9
Breakpoint #3, f2() at test.rs:16
ICVD>>
```

---

## cont

The `cont` command executes the runset to completion or stops at a specified breakpoint.

In this example, the runset completed the debugger run.

```

1 | #include <math.rh>
2 | #include <diagnostics.rh>
3 |
--> 4 | f: list of integer = {1, 1};
5 |
6 | for (i in 2 thru 19) {
7 |     f.push_back(f[i-2] + f[i-1]);
8 | }
ICVD>> c

```

Completed ICV Debugger Run

ICVD>>

In this example, execution stops at a breakpoint that is in a loop. (See the [break](#) command description for information about the `break` command.)

```

ICVD>> b 13
Breakpoint #1, f2() at test.rs:13
ICVD>> c
8 | {
9 |     t : integer = 0;
10 |    y : integer = 10;
11 |    z : integer = 100;
12 |    for ( i=1 to x) {
B --> 13 |        t = t + dtoi((x * y**2) / z);
14 |    }
15 |    t = dtoi(t / x);
16 |    note (" f2(3) = "+ t);
17 | }
Breakpoint #1, f2() at test.rs:13
ICVD>> c
8 | {
9 |     t : integer = 0;
10 |    y : integer = 10;
11 |    z : integer = 100;
12 |    for ( i=1 to x) {
B --> 13 |        t = t + dtoi((x * y**2) / z);
14 |    }
15 |    t = dtoi(t / x);
16 |    note (" f2(3) = "+ t);
17 | }
Breakpoint #1, f2() at test.rs:13
ICVD>>

```

---

## delete

The `delete` command deletes the specified breakpoint. The command format is

`delete number`

```
ICVD>> info b
Breakpoint #1, _MAIN() at test.rs:7 Enabled
Breakpoint #2, _MAIN() at test.rs:16 Enabled
ICVD>> d 1
ICVD>> info b
Breakpoint #2, _MAIN() at test.rs:16 Enabled
ICVD>>
```



---

## disable

The `disable` command disables the specified breakpoint. There are two ways to disable breakpoints:

### 1. `disable`

Using the command without a breakpoint number disables all breakpoints.

```
ICVD>> info b
Breakpoint #1, _MAIN() at test.rs:7 Enabled
Breakpoint #2, _MAIN() at test.rs:18 Enabled
ICVD>> dis
ICVD>> info b
Breakpoint #1, _MAIN() at test.rs:7 Disabled
Breakpoint #2, _MAIN() at test.rs:18 Disabled
```

### 2. `disable number`

If a breakpoint number is specified, only that breakpoint is disabled.

```
ICVD>> dis 1
ICVD>> info b
Breakpoint #1, _MAIN() at test.rs:13 Disabled
Breakpoint #2, _MAIN() at test.rs:16 Enabled
ICVD>>
```

---

## enable

The `enable` command enables a previously disabled breakpoint. There are two ways to enable breakpoints:

1. `enable`

Using the command without a breakpoint number enables all breakpoints.

```
ICVD>> info b
Breakpoint #1, _MAIN() at test.rs:7 Disabled
Breakpoint #2, _MAIN() at test.rs:18 Disabled
ICVD>> en
ICVD>> info b
Breakpoint #1, _MAIN() at test.rs:7 Enabled
Breakpoint #2, _MAIN() at test.rs:18 Enabled
ICVD>>
```

2. `enable number`

If a breakpoint number is specified, only that breakpoint is disabled.

```
ICVD>> info b
Breakpoint #1, _MAIN() at test.rs:7 Disabled
ICVD>> en 1
ICVD>> info b
Breakpoint #1, _MAIN() at test.rs:7 Enabled
ICVD>>
```

---

## finish

The `finish` command executes to the end of the function it is in. Execution does stop, however, at breakpoints.

In this example, the run continues until the current function is completed.

```

      8 | {
      9 |     t : integer = 0;
     10 |     y : integer = 10;
     11 |     z : integer = 100;
     12 |     for ( i=1 to x) {
--> 13 |         t = t + dtoi((x * y**2) / z);
     14 |     }
     15 |     t = dtoi(t / x);
     16 |     note (" f2(3) = "+ t);
     17 | }
ICVD>>  fin
test.rs:16  f2(3) = 3
     15 |     t = dtoi(t / x);
     16 |     note (" f2(3) = "+ t);
     17 | }
     18 | xx:integer;
     19 | f2(3);
--> 20 | note("end. ");

```

In this example, the run continues to until a breakpoint is encountered.

```

ICVD>>  fin
test.rs:11  fibonacci(16) = 1597
      6 | for (i in 2 thru 19) {
      7 |     f.push_back(f[i-2] + f[i-1]);
      8 | }
      9 |
     10 | for (i in 0 thru 19) {
B --> 11 |     note("fibonacci(" + i + ") = " + f[i]);
     12 | }
Breakpoint #2, _MAIN_() at      test.rs:11
ICVD>>  fin
test.rs:11  fibonacci(17) = 2584
      6 | for (i in 2 thru 19) {
      7 |     f.push_back(f[i-2] + f[i-1]);
      8 | }
      9 |
     10 | for (i in 0 thru 19) {
B --> 11 |     note("fibonacci(" + i + ") = " + f[i]);
     12 | }
Breakpoint #2, _MAIN_() at      test.rs:11
ICVD>>

```

---

## info break

The `info break` command lists out information about the breakpoints set during the PXL Debugger session. There are two ways to list the breakpoint information:

1. `info break`

When specified without a breakpoint number, all breakpoints are listed.

```
ICVD>> info b
Breakpoint #1, _MAIN() at      test.rs:7      Enabled
Breakpoint #2, _MAIN() at      test.rs:16     Enabled
```

2. `info break number`

If a breakpoint number is specified, only the breakpoint information for it is listed.

```
ICVD>> info b 2
Breakpoint #2, _MAIN() at      test.rs:16     Enabled
ICVD>>
```

---

## list

The `list` command list lines from the source code. There are four ways to list lines:

### 1. `list`

When specified without any option, the command lists 10 lines from the last printed line.

```

1 | #include <math.rh>
2 | #include <diagnostics.rh>
3 |
--> 4 | f: list of integer = {1, 1};
5 |
6 | for (i in 2 thru 19) {
7 |     f.push_back(f[i-2] + f[i-1]);
8 | }
ICVD>> list
9 |
10 | /* comment 1
11 |    comment 2
12 |    comment 3
13 |    comment 4
14 |    comment 5
15 | */
16 |
17 | for (i in 0 thru 19) {
18 |     note("fibonacci(" + i + ") = " + f[i]);
ICVD>>
```

### 2. `list line`

Print outs, from the current file, five lines before and four lines after the specified line number.

### 3. `list file:line`

Print outs, from the specified file, five lines before and four lines after the specified line number.

```

ICVD>> list test.rs:4
1 | #include <math.rh>
2 | #include <diagnostics.rh>
3 |
--> 4 | f: list of integer = {1, 1};
5 |
6 | for (i in 2 thru 19) {
7 |     f.push_back(f[i-2] + f[i-1]);
8 | }
ICVD>>
```

### 4. `list function`

Prints out five lines before and four lines after the first executable line within the specified function.

```
ICVD>> list f2
4 |
5 | f2 : function (
6 |   x : integer
7 | ) returning void
8 | {
9 |   t : integer = 0;
10 |  y : integer = 10;
11 |  z : integer = 100;
12 |  for ( i=1 to x) {
13 |    t = t + dtol((x * y**2) / z);
ICVD>>
```

---

## next

The `next` command moves the PXL Debugger prompt to the next executable line in the runset. A `next` on a function call completes function execution, and then moves to the next executable line in the runset.

In this example, the `next` command is given at an executable line in the runset:

```

13 |           t = t + dtol((x * y**2) / z);
14 |       }
15 |       t = dtol(t / x);
16 |       note (" f2(3) = "+ t);
17 |   }
--> 18 |   xx:integer;
19 |   f2(3);
20 |   note("end. ");
ICVD>>  n
14 |       }
15 |       t = dtol(t / x);
16 |       note (" f2(3) = "+ t);
17 |   }
18 |   xx:integer;
--> 19 |   f2(3);
20 |   note("end. ");
ICVD>>

```

In this example, the `next` command is given on a function.

```

8 |   {
9 |       ans = x;
10 |   }
11 |
12 |   xx: integer = 0;
--> 13 |   xx = f2(3);
14 |   note("1. f2(3)= " + xx);
ICVD>>  n
9 |       ans = x;
10 |   }
11 |
12 |   xx: integer = 0;
13 |   xx = f2(3);
--> 14 |   note("1. f2(3)= " + xx);
ICVD>>

```

---

## print

The `print` command prints values of variables and simple expressions. The command format is

```
print expression
```

In this example, the value of a variable is printed:

```

5 | f2 : function (
6 |     x : integer
7 | ) returning ans : integer
8 | {
B   9 |     ans = x;
--> 10 |     note("0. ans = " + ans);
11 | }
12 |
13 | xx: integer = 0;
14 | xx = f2(3);
ICVD>> p ans
ans = 3
```

In this example, the value of a loop variable is printed:

```

2 | #include <diagnostics.rh>
3 |
4 | f: list of integer = {1, 1};
5 |
6 | for (i in 2 thru 19) {
B --> 7 |     f.push_back(f[i-2] + f[i-1]);
8 | }
9 |
10 | /* comment 1
11 |    comment 2
Breakpoint #1, _MAIN_() at      test.rs:7
ICVD>> p i
i = 3
```

In this example, the value of an expression is printed.

```

19 | */
20 |     t = dtol(t / x);
21 |     note (" f2(3) = "+ t);
22 | }
23 | xx:integer = 0;
--> 24 | f2(3);
25 | note("end. ");
ICVD>> p xx
xx = 0
ICVD>> p (xx+7)*20
(xx+7)*20 = 140
ICVD>>
```



---

## quit

The `quit` command ends the debugging session and the IC Validator run.

```
1 | #include <math.rh>
2 | #include <diagnostics.rh>
3 |
--> 4 | f: list of integer = {1, 1};
5 |
6 | for (i in 2 thru 19) {
7 |     f.push_back(f[i-2] + f[i-1]);
8 | }
ICVD>> q
IC Validator Run: Time= 0:00:04 User=0.06 System=0.01
Overall icv_engine time = 0:00:04
IC Validator is done.
```

---

## record

The `record` command saves the commands from the current debugging session into an ASCII file. There are two ways to save commands:

### 1. `record`

If a file name is not specified, the commands are saved in `ICVpid.pdi`, where *pid* is the UNIX process ID.

```
ICVD>> record
Saving Commands to 'ICV22347.pdi'
ICVD>>
```

### 2. `record file`

You can specify a file name, with or without a path.

```
ICVD>> record myfile.cmd
Saving Commands to 'myfile.cmd'
ICVD>>
```

The ASCII file stores all commands, except `break` and `list`, verbatim from the PXL Debugger. The `break` and `list` commands are stored with the resolved file path. For example,

```
Starting ICV Debugger
  2 | #include "math.rh"
  3 | #include "string.rh"
  4 |
  5 | #include "b.rs"
  6 |
--> 7 | xx: integer = 0;
    8 | xx = f2(3);
    9 | note("2. f2(3)= " + xx);
ICVD>> record
Saving Commands to 'ICV21589.pdi'
ICVD>> b f2
Breakpoint #1, f2() at b.rs:5
ICVD>> list f2
  1 | f2 : function (
  2 |     x : integer
  3 | ) returning ans : integer
  4 | {
B   5 |     ans = x;
    6 |     note("0. ans = " + ans);
    7 | }
ICVD>> b 6
Breakpoint #2, f2() at b.rs:6
ICVD>> q
```

The recorded file (ICV21589.pdi) contains:

```
b f2
list f2
b b.rs:6
q
```

---

## source

The `source` command executes the PXL Debugger commands in the specified file. After the commands are completed, control of the PXL Debugger returns to the user unless the PXL Debugger completed because of a `quit` command specified in the source file. The command format is

```
source file
```

For example,

```
ICVD>> source ICVDebug9398.pdi
ICVD>> n
    2 | #include <diagnostics.rh>
    3 |
    4 | f: list of integer = {1, 1};
    5 |
    6 | for (i in 2 thru 19) {
--> 7 |     f.push_back(f[i-2] + f[i-1]);
    8 | }
    9 |
   10 | /* comment 1
   11 |     comment 2
ICVD>> b 7
Breakpoint #1, _MAIN() at test.rs:7
ICVD>> b 16
Breakpoint #2, _MAIN() at test.rs:16
ICVD>> c
    2 | #include <diagnostics.rh>
    3 |
    4 | f: list of integer = {1, 1};
    5 |
    6 | for (i in 2 thru 19) {
B --> 7 |     f.push_back(f[i-2] + f[i-1]);
    8 | }
    9 |
   10 | /* comment 1
   11 |     comment 2
Breakpoint #1, _MAIN() at test.rs:7
End of commands from commandfile
ICVD>>
```

---

## step

The `step` command allows you to step into functions calls, executing each line of within the function. The cursor moves to the first executable line in the function.

In this example, the PXL Debugger steps through `f2`, where the first executable line is 9.

```

14 |     }
15 |     t = dtol(t / x);
16 |     note (" f2(3) = "+ t);
17 | }
18 | xx:integer;
--> 19 | f2(3);
20 | note("end. ");
ICVD>> s
4 |
5 | f2 : function (
6 |     x : integer
7 | ) returning void
8 | {
--> 9 |     t : integer = 0;
10 |     y : integer = 10;
11 |     z : integer = 100;
12 |     for ( i=1 to x) {
13 |         t = t + dtol((x * y**2) / z);
ICVD>>

```

---

## where

The `where` command prints the function call stack of a statement in the runset. In this example the command shows that there have been two function calls.

```
ICVD>> s
    22 |      c = a + b;
    23 |      };
    24 |
    25 |  sum: function (x: integer, y: integer) returning z: integer
    26 |      {
-->  27 |      z = add(x,y);
    28 |      note("z = " + z);
    29 |      }
    30 |
    31 |  q = add(x,y);
ICVD>> where
#0 | sum()      at test.rs:27
#1 | _MAIN_()   at test.rs:32
```

In this example the command shows that there have been three function calls.

```
ICVD>> s
    17 |  note("x is now #2 = " + x);
    18 |  note("y is now #2 = " + y);
    19 |
    20 |  add: function (a: integer, b: integer) returning c: integer
    21 |      {
-->  22 |      c = a + b;
    23 |      };
    24 |
    25 |  sum: function (x: integer, y: integer) returning z: integer
    26 |      {
ICVD>> where
#0 | add()      at test.rs:22
#1 | sum()      at test.rs:27
#2 | _MAIN_()   at test.rs:32
ICVD>>
```

# E

## PYDB Perl API

---

*This appendix describes how to access the IC Validator error database (PYDB) using a Perl application programming interface (API) that is provided with the IC Validator tool.*

The PYDB Perl API gives you direct access to the error data produced by the IC Validator tool. For example, you can

- Generate error information for a specific violation.
- Count violations for tracking purposes.

The database and the Perl API are described in the following sections:

- [PYDB Perl API Overview](#)
- [Accessing the PYDB](#)
- [Examples of Common Tasks](#)
- [PYDB Schema](#)

---

## PYDB Perl API Overview

Using the PYDB Perl API you can access information about violations found with the IC Validator tool and process the data for your needs. The IC Validator tool stores only violations and cells that contain error data in the PYDB. The LAYOUT\_ERRORS file typically shows most of the information that is available in the PYDB.

Error data produced by the IC Validator tool is stored in tables in an SQLite database; this database is the PYDB. The structure of the tables and how they are related to each other is called the database schema. See [“PYDB Schema”](#) for information about the schema.

To properly query a complex database, you must know the database schema. You can use SQLite commands to access the database and return information. These commands can be complex depending on the information you want to be returned.

For convenient access to error data without needing to learn the database schema, The provided PYDB Perl API connects to and retrieves data from a PYDB. It makes interfacing with the error database much easier than using SQLite commands. It has built-in functions for

- Connecting to the PYDB server.
- Submitting queries and ensuring that multiple database servers are not competing with each other.
- Providing high-level functions to retrieve PYDB specific information without needing to create complex SQLite queries.
- Processing the results.

The PYDB Perl API is located in \$ICV\_HOME\_DIR/contrib/pydb/. The subdirectories are shown in [Table E-1](#).

*Table E-1 Subdirectories of the pydb Directory*

Subdirectory	Description
bin/	Contains example scripts. These scripts show examples of how to use the PYDB modules.
lib/	Contains public library modules. Each of these files has a short description followed by detailed descriptions of all the class methods provided by the module.



---

## Accessing the PYDB

The PYDB Perl API provides you with a high level interface that allows you to work with error data, as well as providing the ability to run direct SQLite queries on the error database, if desired.

**Note:**

The PYDB schema and server communication protocol can change between versions. Therefore, to access the PYDB you must use the API that is from the IC Validator version which generated the PYDB.

To access the PYDB using the PYDB Perl API provided with the IC Validator tool, follow these steps:

1. Load the PYDB library.

Include the following code in your Perl script before any database commands. This code imports the needed modules from the IC Validator installation into the script.

```
push @INC, $ENV{"ICV_HOME_DIR"}."/contrib/pydb/lib";
require "PYDB.pm";
```

2. Connect to the database.

Include the following code in your Perl script to connect to the database.

```
$dbh = PYDB::Connect($db_path,$db_name);
if (! $dbh) {
    print "ERROR: Error connecting to database: $PYDB::errstr\n";
    exit;
}
```

If the connection succeeds, a database handle is returned.

3. Perform database queries.

Using the PYDB Perl API you can access information about the errors and process the data for your needs. Only violations and cells that contain error data are stored in the database. The LAYOUT\_ERRORS file typically shows most of the information that is available in the PYDB. See [“Examples of Common Tasks”](#) for examples.

4. Disconnect from the database.

Include the following code in your Perl script to disconnect from the database.

```
$dbh->Disconnect();
```

---

## Examples of Common Tasks

The following examples show some common tasks.

- List violations in the database.

```
# All violations stored in the PYDB
foreach $vio ($dbh->Violations()) {
    print "Violation: \"\".$vio->Comment().\"\"\\n";
}

# Find a specific violation
$vio = $dbh->GetViolationByComment("M1.S1.1: Metal 1 space");
if ($vio) {
    print "Found M1 violation, with \"\".$vio->ErrorCount().
        " error(s).\\n";
}

# Find violations that match a regular expression
foreach $vio ($dbh->GetViolationsByCommentRegex(qr/M1\\.*/)) {
    print "M1 rule: \"\".$vio->Comment().\"\"\\n";
}
```

- List cells in the database.

```
# All cells stored in the PYDB
foreach $cell ($dbh->Cells()) {
    print "Cell: \"\".$cell->CellName().\"\"\\n";
}

# Find a specific cell
$cell = $dbh->GetCellByName("TOP");
if ($cell) {
    print "Found top cell.\\n";
}

# Find cells that match a regular expression
foreach $cell ($dbh->GetCellsByNameRegex(qr/STDCELL.*/)) {
    print "Found cell: \"\".$cell->CellName().\"\"\\n";
}
```

- Access errors in a specific violation.

```
$vio = $dbh->GetViolationByComment("M1.S1.1: Metal 1 space");
if ($vio) {

    foreach $command ($vio->Commands()) {
        print "Command \"\".$command->Comment().
            " produced \"\".$command->NumErrorsUnlimited().
            " errors, details are available for \"\".
            $command->NumErrors().\"\"\\n";
    }
}
```

```

foreach $cell ($command->Cells()) {
    # Errors are stored by command/cell, the error
    # iterator can be used to access them
    $iter = $dbh->ListErrors($command, $cell);
    if (! $iter) {
        print "ERROR: $PYDB::errstr\n";
        $dbh->Disconnect();
        exit;
    }

    while ($error = $iter->NextError()) {
        # Bounding box of error
        @bbox = $error->GetBBox();
        # Error polygons
        foreach $polygon ($error->Polygons()) {
            # List of points making up the polygon
            @points = $polygon->Points();
        }
        # Access to specific data fields by column name
        $distance = $error->Get("Distance");
        # Certain types of errors might have child errors
        foreach $child_error ($error->Children()) {
            # ...
        }
    }
}
}
}
}
}

```

- Perform a direct SQLite query. The Perl API provides an interface similar to the DBI module that comes with most Perl installations.

```

# Prepare the query
$ssth = $dbh->prepare("SELECT CellID, CellName FROM pyCellNames");
if (! $ssth) {
    print "ERROR: $PYDB::errstr\n";
    $dbh->Disconnect();
    exit;
}

# Execute the query
if (! $ssth->execute()) {
    print "ERROR: $PYDB::errstr\n";
    $dbh->Disconnect();
    exit;
}

# Retrieve the data
while ($ref = $ssth->fetchrow_hashref()) {
    print "Cell name=".$ref->{'CellName'}.
        ", id=".$ref->{'CellID'}."\\n";
}

```

---

## PYDB Schema

In the PYDB schema, errors are stored in error tables.

- Each table is made up of rows of data that can reference several related columns, similar to a spreadsheet.
- Each table needs at least one unique key to identify individual records. Some tables reference several keys.

There is at least one table per command type, with additional tables for commands with child errors. Error classification information is also stored in the PYDB in several tables. [Table E-2](#) shows some of the commonly used tables in the PYDB.

*Table E-2 PYDB Organization*

Table	Information stored
pyCellNames	Contains cell information. Each command can write errors from different cells.
pyCommandCells	Lists the cells that contain errors for a command.
pyCommands	Contains one record for each runset command that produced errors. Multiple commands, for example, the <code>internal1()</code> and <code>external2()</code> functions, can write to the same violation comment.
pyErrors_<command type>	Contains individual error data. The error tables for a particular command type are created on the fly and contain data columns specific to that type of command.
pyErrorTableFormat	Defines the data columns of an error table.
pyErrorTables	Defines the hierarchy of error tables associated with a type of command.
pyPolygons_<command type>	Contains individual error shapes.
pyRunInfo	Contains general information.
pyViolations	Defines the violation rule comments for which errors are stored in the PYDB.

# F

## Third-Party Licenses

---

This appendix provides the license information for third-party software that is used in the IC Validator tool.

- [Licensing Overview](#)
- [Third-Party Software](#)
  - [ANTLR](#)
  - [Boost](#)
  - [cx\\_Freeze](#)
  - [libdp](#)
  - [libxml2](#)
  - [MariaDB Connector](#)
  - [MariaDB](#)
  - [MCPD Public Domain Code](#)
  - [patchELF](#)
  - [Python 2.7 License](#)
  - [Shroud-1.0](#)
  - [SQLite](#)

- [TclLib](#)

- [Tcl/Tk](#)

Note:

The tbcload code is part of Tcl.

- [zlib](#)

---

## Licensing Overview

This document includes licensing information relating to free and open-source software ("FOSS") included with Synopsys's<sup>®</sup> IC Validator products (the "SOFTWARE"). The terms of the applicable FOSS license(s) govern Synopsys's<sup>®</sup> distribution and your use of the SOFTWARE. Synopsys<sup>®</sup> and the third-party authors, licensors, and distributors of the SOFTWARE disclaim all warranties and all liability arising from any and all use and distribution of the SOFTWARE. To the extent the FOSS is provided under an agreement with Synopsys<sup>®</sup> that differs from the applicable FOSS license(s), those terms are offered by Synopsys<sup>®</sup> alone.

Synopsys<sup>®</sup> has reproduced below copyright and other licensing notices appearing within the FOSS packages. While Synopsys<sup>®</sup> seeks to provide complete and accurate copyright and licensing information for each FOSS package, Synopsys<sup>®</sup> does not represent or warrant that the following information is complete, correct, or error-free. SOFTWARE recipients are encouraged to (a) investigate the identified FOSS packages to confirm the accuracy of the licensing information provided herein and (b) notify Synopsys<sup>®</sup> of any inaccuracies or errors found in this document so that Synopsys<sup>®</sup> may update this document accordingly.

Certain FOSS licenses (such as the GNU General Public Licenses, GNU Library/Lesser General Public Licenses, Affero General Public Licenses, Mozilla Public Licenses, Common Development and Distribution Licenses, Common Public License, and Eclipse Public License) require that the source code corresponding to distributed FOSS binaries be made available to recipients or other requestors under the terms of the same FOSS license. Recipients or requestors who would like to receive a copy of such corresponding source code should submit a request to Synopsys<sup>®</sup> by post at:

Synopsys

Attn: Open Source Requests

690 E. Middlefield Road

Mountain View, CA 94043

Please provide the following information in all submitted FOSS requests:

- The FOSS packages for which you are requesting source code;
- The Synopsys<sup>®</sup> product (and any available version information) with which the requested FOSS packages are distributed;
- An email address at which Synopsys<sup>®</sup> may contact you regarding the request (if available); and
- The postal address for delivery of the requested source code.

---

## Third-Party Software

---

### ANTLR

#### ANTLR 2 License

*[The BSD License]*

Copyright (c) 2012 Terence Parr and Sam Harwell

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- 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.
- Neither the name of the author 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 COPYRIGHT HOLDERS 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 COPYRIGHT OWNER 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.

---

### Boost

Boost Software License - Version 1.0 - August 17th, 2003

Permission is hereby granted, free of charge, to any person or organization obtaining a copy of the software and accompanying documentation covered by this license (the "Software") to use, reproduce, display, distribute, execute, and transmit the Software, and to prepare



derivative works of the Software, and to permit third-parties to whom the Software is furnished to do so, all subject to the following:

The copyright notices in the Software and this entire statement, including the above license grant, this restriction and the following disclaimer, must be included in all copies of the Software, in whole or in part, and all derivative works of the Software, unless such copies or derivative works are solely in the form of machine-executable object code generated by a source language processor.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR ANYONE DISTRIBUTING THE SOFTWARE BE LIABLE FOR ANY DAMAGES OR OTHER LIABILITY, WHETHER IN CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

---

## **cx\_Freeze**

Project homepage: <http://cx-freeze.readthedocs.io/> [https://pypi.python.org/pypi/cx\\_Freeze/4.3.4](https://pypi.python.org/pypi/cx_Freeze/4.3.4)

Project license: Licensing Copyright © 2007-2014, Anthony Tuininga. Copyright © 2001-2006, Computronix (Canada) Ltd., Edmonton, Alberta, Canada. All rights reserved. NOTE: this license is derived from the Python Software Foundation License which can be found at <http://www.python.org/psf/license>

License for cx\_Freeze 5.0 This LICENSE AGREEMENT is between the copyright holders and the Individual or Organization (“Licensee”) accessing and otherwise using cx\_Freeze software in source or binary form and its associated documentation. Subject to the terms and conditions of this License Agreement, the copyright holders hereby grant Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use cx\_Freeze alone or in any derivative version, provided, however, that this License Agreement and this notice of copyright are retained in cx\_Freeze alone or in any derivative version prepared by Licensee. In the event Licensee prepares a derivative work that is based on or incorporates cx\_Freeze or any part thereof, and wants to make the derivative work available to others as provided herein, then Licensee hereby agrees to include in any such work a brief summary of the changes made to cx\_Freeze. The copyright holders are making cx\_Freeze available to Licensee on an “AS IS” basis. THE COPYRIGHT HOLDERS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, THE COPYRIGHT HOLDERS MAKE NO AND DISCLAIM ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF CX\_FREEZE WILL

NOT INFRINGE ANY THIRD PARTY RIGHTS. THE COPYRIGHT HOLDERS SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF CX\_FREEZE FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING CX\_FREEZE, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF. This License Agreement will automatically terminate upon a material breach of its terms and conditions. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between the copyright holders and Licensee. This License Agreement does not grant permission to use copyright holder's trademarks or trade name in a trademark sense to endorse or promote products or services of Licensee, or any third party. By copying, installing or otherwise using cx\_Freeze, Licensee agrees to be bound by the terms and conditions of this License Agreement. Computronix® is a registered trademark of Computronix (Canada) Ltd.

---

## **libdp**

1. The "Software", below, refers to the Tcl-DP system, developed by the Tcl-DP group (in either source-code, object-code or executable-code form), and related documentation, and a "work based on the Software" means a work based on either the Software, on part of the Software, or on any derivative work of the Software under copyright law: that is, a work containing all or a portion of the Tcl-DP system, either verbatim or with modifications. Each licensee is addressed as "you" or "Licensee."
2. Cornell University as the parent organization of the Tcl-DP group holds copyrights in the Software. The copyright holder reserves all rights except those expressly granted to licensees, and U.S. Government license rights.
3. Permission is hereby granted to use, copy, modify, and to redistribute to others. If you distribute a copy or copies of the Software, or you modify a copy or copies of the Software or any portion of it, thus forming a work based on the Software, and make and/or distribute copies of such work, you must meet the following conditions:
  1. If you make a copy of the Software (modified or verbatim) it must include the copyright notice and this license.
  2. You must cause the modified Software to carry prominent notices stating that you changed specified portions of the Software.
4. LICENSEE AGREES THAT THE EXPORT OF GOODS AND/OR TECHNICAL DATA FROM THE UNITED STATES MAY REQUIRE SOME FORM OF EXPORT CONTROL LICENSE FROM THE U.S. GOVERNMENT AND THAT FAILURE TO OBTAIN SUCH EXPORT CONTROL LICENSE MAY RESULT IN CRIMINAL LIABILITY UNDER U.S. LAWS.
5. Portions of the Software resulted from work developed under a U.S. Government Contract and are subject to the following license: the Government is granted for itself and

others acting on its behalf a paid-up, nonexclusive, irrevocable worldwide license in this computer software to reproduce, prepare derivative works, and perform publicly and display publicly.

6. Disclaimer of warranty: Licensors provides the software on an "as is" basis. Licensors does not warrant, guarantee, or make any representations regarding the use or results of the software with respect to its correctness, accuracy, reliability or performance. The entire risk of the use and performance of the software is assumed by licensee. ALL WARRANTIES INCLUDING, WITHOUT LIMITATION, ANY WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE OR MERCHANTABILITY ARE HEREBY EXCLUDED.
7. Lack of maintenance or support services: Licensee understands and agrees that licensors is under no obligation to provide maintenance, support or update services, notices of latent defects, or correction of defects for the software.
8. Limitation of liability, indemnification: Even if advised of the possibility of damages, under no circumstances shall licensors be liable to licensee or any third party for damages of any character, including, without limitation, direct, indirect, incidental, consequential or special damages, loss of profits, loss of use, loss of goodwill, computer failure or malfunction. Licensee agrees to indemnify and hold harmless licensors for any and all liability licensors may incur as a result of licensee's use of the software.

---

## libxml2

Libxml2 is the XML C parser and toolkit developed for the Gnome project (but usable outside of the Gnome platform), it is free software available under the MIT License. The MIT license:

The MIT License (MIT)

Copyright (c) <year> <copyright holders>

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER

IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

---

## MariaDB Connector

Synopsys modifications: None

MariaDB Connector is licensed under the GNU Lesser General Public License, version 2.1, (LGPL-2.1). Software licensed under the LGPL-2.1 is free software that comes with ABSOLUTELY NO WARRANTY. You may modify, redistribute, or otherwise use the GPL software under the terms of the LGPL-2.1 (<http://www.gnu.org/licenses/old-licenses/lgpl-2.1.html>).

### GNU LESSER GENERAL PUBLIC LICENSE

Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc. 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

[This is the first released version of the Lesser GPL. It also counts as the successor of the GNU Library Public License, version 2, hence the version number 2.1.]

### TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:
  - The modified work must itself be a software library.
  - You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.
  - You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.
  - If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

(For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

- Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)
- Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.
- Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.
- If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.
- Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:
  - Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.
  - Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.
8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.
9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.
10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.
11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by



court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

## NO WARRANTY

- BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.
- IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

---

## MariaDB

Synopsys modifications: None

MariaDB is licensed under the GNU General Public License, version 2, (GPL-2.0). Software licensed under the GPL-2.0 is free software that comes with ABSOLUTELY NO WARRANTY. You may modify, redistribute, or otherwise use the GPL software under the terms of the GPL-2.0 (<http://www.gnu.org/licenses/old-licenses/gpl-2.0.html>)."

## GNU GENERAL PUBLIC LICENSE

Version 2.1, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc. 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

## TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License.

The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:
  - You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
  - You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.
  - If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose

permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this

License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.
6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.
7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

## **NO WARRANTY**

- BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.
- IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

---

## MCPD Public Domain Code

Copyright (c) 1998, 2002-2007 Kiyoshi Matsui <kmatsui@t3.rim.or.jp>  
All rights reserved.

This software including the files in this directory is provided under the following license.

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.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR "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 AUTHOR 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.

---

## patchELF

Project homepage: <https://nixos.org/patchelf.html>

Project license:

COPYING

GNU GENERAL PUBLIC LICENSE Version 3, 29 June 2007

Copyright (C) 2007 Free Software Foundation, Inc. <<http://fsf.org/>> Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program--to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.



## TERMS AND CONDITIONS

### 0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

### 1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that

Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

## 2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

## 3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

#### 4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

#### 5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

- a) The work must carry prominent notices stating that you modified it, and giving a relevant date.
- b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".
- c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.
- d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

## 6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

- a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.
- b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.
- c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.
- d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.
- e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a

consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

## 7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

- a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
- b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
- c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or
- d) Limiting the use for publicity purposes of names of licensors or authors of the material; or
- e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or
- f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

## 8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly

and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

#### 9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

#### 10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

#### 11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.



Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

#### 12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

#### 13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

#### 14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

#### 15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE

COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

#### 16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

#### 17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

### END OF TERMS AND CONDITIONS

#### How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the program's name and a brief idea of what it does.> Copyright (C) <year>  
<name of author>

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

```
<program> Copyright (C) <year> <name of author> This program comes with
ABSOLUTELY NO WARRANTY; for details type `show w'. This is free software, and you are
welcome to redistribute it under certain conditions; type `show c' for details.
```

The hypothetical commands `show w' and `show c' should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see <http://www.gnu.org/licenses/>.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read <http://www.gnu.org/philosophy/why-not-lgpl.html>.

---

## Python 2.7 License

PYTHON SOFTWARE FOUNDATION LICENSE VERSION 2

-----

1. This LICENSE AGREEMENT is between the Python Software Foundation ("PSF"), and the Individual or Organization ("Licensee") accessing and otherwise using this software ("Python") in source or binary form and its associated documentation.
2. Subject to the terms and conditions of this License Agreement, PSF hereby grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use Python alone or in any derivative version, provided, however, that PSF's License Agreement and PSF's notice of copyright, i.e., "Copyright (c) 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014 Python

Software Foundation; All Rights Reserved" are retained in Python alone or in any derivative version prepared by Licensee.

3. In the event Licensee prepares a derivative work that is based on or incorporates Python or any part thereof, and wants to make the derivative work available to others as provided herein, then Licensee hereby agrees to include in any such work a brief summary of the changes made to Python.
4. PSF is making Python available to Licensee on an "AS IS" basis. PSF MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, PSF MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.
5. PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.
6. This License Agreement will automatically terminate upon a material breach of its terms and conditions.
7. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between PSF and Licensee. This License Agreement does not grant permission to use PSF trademarks or trade name in a trademark sense to endorse or promote products or services of Licensee, or any third party.
8. By copying, installing or otherwise using Python, Licensee agrees to be bound by the terms and conditions of this License Agreement.

---

## Shroud-1.0

Project homepage: <http://www.cpan.org/authors/id/C/CR/CRAIC/shroud-1.0>

Project license:

```
#!/usr/bin/perl
```

```
# shroud
```

```
# Copyright 2000 Robert Jones, Craic Computing, All rights reserved.
```

```
# This program is free software; you can redistribute it and/or modify it
```

```
# under the same terms as Perl itself.
```

# The software is supplied as is, with absolutely no warranty.

Perl5 is Copyright (C) 1993-2005, by Larry Wall and others.

It is free software; you can redistribute it and/or modify it under the terms of either:

a) the GNU General Public License as published by the Free Software Foundation; either external linkversion 1, or (at your option) any later versionexternal link, or

b) the "Artistic License".

For those of you that choose to use the GNU General Public License, my interpretation of the GNU General Public License is that no Perl script falls under the terms of the GPL unless you explicitly put said script under the terms of the GPL yourself.

Furthermore, any object code linked with perl does not automatically fall under the terms of the GPL, provided such object code only adds definitions of subroutines and variables, and does not otherwise impair the resulting interpreter from executing any standard Perl script. I consider linking in C subroutines in this manner to be the moral equivalent of defining subroutines in the Perl language itself. You may sell such an object file as proprietary provided that you provide or offer to provide the Perl source, as specified by the GNU General Public License. (This is merely an alternate way of specifying input to the program.) You may also sell a binary produced by the dumping of a running Perl script that belongs to you, provided that you provide or offer to provide the Perl source as specified by the GPL. (The fact that a Perl interpreter and your code are in the same binary file is, in this case, a form of mere aggregation.)

This is my interpretation of the GPL. If you still have concerns or difficulties understanding my intent, feel free to contact me. Of course, the Artistic License spells all this out for your protection, so you may prefer to use that.

-- Larry Wall

The "Artistic License"

Preamble

The intent of this document is to state the conditions under which a Package may be copied, such that the Copyright Holder maintains some semblance of artistic control over the development of the package, while giving the users of the package the right to use and distribute the Package in a more-or-less customary fashion, plus the right to make reasonable modifications.

Definitions:

"Package" refers to the collection of files distributed by the Copyright Holder, and derivatives of that collection of files created through textual modification.

"Standard Version" refers to such a Package if it has not been modified, or has been modified in accordance with the wishes of the Copyright Holder as specified below.

"Copyright Holder" is whoever is named in the copyright or copyrights for the package.

"You" is you, if you're thinking about copying or distributing this Package.

"Reasonable copying fee" is whatever you can justify on the basis of media cost, duplication charges, time of people involved, and so on. (You will not be required to justify it to the Copyright Holder, but only to the computing community at large as a market that must bear the fee.)

"Freely Available" means that no fee is charged for the item itself, though there may be fees involved in handling the item. It also means that recipients of the item may redistribute it under the same conditions they received it.

1. You may make and give away verbatim copies of the source form of the Standard Version of this Package without restriction, provided that you duplicate all of the original copyright notices and associated disclaimers.

2. You may apply bug fixes, portability fixes and other modifications derived from the Public Domain or from the Copyright Holder. A Package modified in such a way shall still be considered the Standard Version.

3. You may otherwise modify your copy of this Package in any way, provided that you insert a prominent notice in each changed file stating how and when you changed that file, and provided that you do at least ONE of the following:

a) place your modifications in the Public Domain or otherwise make them Freely Available, such as by posting said modifications to Usenet or an equivalent medium, or placing the modifications on a major archive site such as uunet.uu.net, or by allowing the Copyright Holder to include your modifications in the Standard Version of the Package.

b) use the modified Package only within your corporation or organization.

c) rename any non-standard executables so the names do not conflict with standard executables, which must also be provided, and provide a separate manual page for each non-standard executable that clearly documents how it differs from the Standard Version.

d) make other distribution arrangements with the Copyright Holder.

4. You may distribute the programs of this Package in object code or executable form, provided that you do at least ONE of the following:

a) distribute a Standard Version of the executables and library files, together with instructions (in the manual page or equivalent) on where to get the Standard Version.

b) accompany the distribution with the machine-readable source of the Package with your modifications.

c) give non-standard executables non-standard names, and clearly document the differences in manual pages (or equivalent), together with instructions on where to get the Standard Version.

d) make other distribution arrangements with the Copyright Holder.

5. You may charge a reasonable copying fee for any distribution of this Package. You may charge any fee you choose for support of this Package. You may not charge a fee for this Package itself. However, you may distribute this Package in aggregate with other (possibly commercial) programs as part of a larger (possibly commercial) software distribution provided that you do not advertise this Package as a product of your own. You may embed this Package's interpreter within an executable of yours (by linking); this shall be construed as a mere form of aggregation, provided that the complete Standard Version of the interpreter is so embedded.

6. The scripts and library files supplied as input to or produced as output from the programs of this Package do not automatically fall under the copyright of this Package, but belong to whoever generated them, and may be sold commercially, and may be aggregated with this Package. If such scripts or library files are aggregated with this Package via the so-called "undump" or "unexec" methods of producing a binary executable image, then distribution of such an image shall neither be construed as a distribution of this Package nor shall it fall under the restrictions of Paragraphs 3 and 4, provided that you do not represent such an executable image as a Standard Version of this Package.

7. C subroutines (or comparably compiled subroutines in other languages) supplied by you and linked into this Package in order to emulate subroutines and variables of the language defined by this Package shall not be considered part of this Package, but are the equivalent of input as in Paragraph 6, provided these subroutines do not change the language in any way that would cause it to fail the regression tests for the language.

8. Aggregation of this Package with a commercial distribution is always permitted provided that the use of this Package is embedded; that is, when no overt attempt is made to make this Package's interfaces visible to the end user of the commercial distribution. Such use shall not be construed as a distribution of this Package.

9. The name of the Copyright Holder may not be used to endorse or promote products derived from this software without specific prior written permission.

10. THIS PACKAGE IS PROVIDED "AS IS" AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

The End

GNU GENERAL PUBLIC LICENSE Version 1, February 1989

Copyright (C) 1989 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307, USA Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

## Preamble

The license agreements of most software companies try to keep users at the mercy of those companies. By contrast, our General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. The General Public License applies to the Free Software Foundation's software and to any other program whose authors commit to using it. You can use it for your programs, too.

When we speak of free software, we are referring to freedom, not price. Specifically, the General Public License is designed to make sure that you have the freedom to give away or sell copies of free software, that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of a such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must tell them their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

The precise terms and conditions for copying, distribution and modification follow. GNU GENERAL PUBLIC LICENSE TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any work containing the Program or a portion of it, either verbatim or with modifications. Each licensee is addressed as "you".

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the



notices that refer to this General Public License and to the absence of any warranty; and give any other recipients of the Program a copy of this General Public License along with the Program. You may charge a fee for the physical act of transferring a copy.

2. You may modify your copy or copies of the Program or any portion of it, and copy and distribute such modifications under the terms of Paragraph 1 above, provided that you also do the following:

a) cause the modified files to carry prominent notices stating that you changed the files and the date of any change; and

b) cause the whole of any work that you distribute or publish, that in whole or in part contains the Program or any part thereof, either with or without modifications, to be licensed at no charge to all third parties under the terms of this General Public License (except that you may choose to grant warranty protection to some or all third parties, at your option).

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the simplest and most usual way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this General Public License.

d) You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

Mere aggregation of another independent work with the Program (or its derivative) on a volume of a storage or distribution medium does not bring the other work under the scope of these terms. 3. You may copy and distribute the Program (or a portion or derivative of it, under Paragraph 2) in object code or executable form under the terms of Paragraphs 1 and 2 above provided that you also do one of the following:

a) accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Paragraphs 1 and 2 above; or,

b) accompany it with a written offer, valid for at least three years, to give any third party free (except for a nominal charge for the cost of distribution) a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Paragraphs 1 and 2 above; or,

c) accompany it with the information you received as to where the corresponding source code may be obtained. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form alone.)

Source code for a work means the preferred form of the work for making modifications to it. For an executable file, complete source code means all the source code for all modules it contains; but, as a special exception, it need not include source code for modules which are

standard libraries that accompany the operating system on which the executable file runs, or for standard header files or definitions files that accompany that operating system.

4. You may not copy, modify, sublicense, distribute or transfer the Program except as expressly provided under this General Public License. Any attempt otherwise to copy, modify, sublicense, distribute or transfer the Program is void, and will automatically terminate your rights to use the Program under this License. However, parties who have received copies, or rights to use copies, from you under this General Public License will not have their licenses terminated so long as such parties remain in full compliance.

5. By copying, distributing or modifying the Program (or any work based on the Program) you indicate your acceptance of this license to do so, and all its terms and conditions.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. 7. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of the license which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the license, you may choose any version ever published by the Free Software Foundation.

8. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

#### NO WARRANTY

9. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

10. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

#### END OF TERMS AND CONDITIONS Appendix: How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to humanity, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

Copyright (C) 19yy

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 1, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307, USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) 19xx name of author Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'. This is free software, and you are welcome to redistribute it under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w' and `show c'; they could even be mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program `Gnomovision' (a program to direct compilers to make passes at assemblers) written by James Hacker.

, 1 April 1989 Ty Coon, President of Vice

That's all there is to it!

---

## SQLite

All of the code and documentation in SQLite has been dedicated to the public domain by the authors. All code authors, and representatives of the companies they work for, have signed affidavits dedicating their contributions to the public domain and originals of those signed affidavits are stored in a firesafe at the main offices of Hwaci. Anyone is free to copy, modify, publish, use, compile, sell, or distribute the original SQLite code, either in source code form or as a compiled binary, for any purpose, commercial or non-commercial, and by any means.

---

## TclLib

This software is copyrighted by Ajuba Solutions and other parties. The following terms apply to all files associated with the software unless explicitly disclaimed in individual files.

The authors hereby grant permission to use, copy, modify, distribute, and license this software and its documentation for any purpose, provided that existing copyright notices are retained in all copies and that this notice is included verbatim in any distributions. No written agreement, license, or royalty fee is required for any of the authorized uses. Modifications to this software may be copyrighted by their authors and need not follow the licensing terms described here, provided that the new terms are clearly indicated on the first page of each file where they apply.

IN NO EVENT SHALL THE AUTHORS OR DISTRIBUTORS BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF THIS SOFTWARE, ITS DOCUMENTATION, OR ANY DERIVATIVES THEREOF, EVEN IF THE AUTHORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

THE AUTHORS AND DISTRIBUTORS SPECIFICALLY DISCLAIM ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT. THIS SOFTWARE IS PROVIDED ON AN "AS IS" BASIS, AND THE AUTHORS AND DISTRIBUTORS HAVE NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

GOVERNMENT USE: If you are acquiring this software on behalf of the U.S. government, the Government shall have only "Restricted Rights" in the software and related documentation as defined in the Federal Acquisition Regulations (FARs) in Clause 52.227.19 (c) (2). If you are acquiring the software on behalf of the Department of Defense, the software shall be classified as "Commercial Computer Software" and the Government shall have only "Restricted Rights" as defined in Clause 252.227-7013 (c) (1) of DFARs. Notwithstanding the foregoing, the authors grant the U.S. Government and others acting in its behalf permission to use and distribute the software in accordance with the terms specified in this license.

---

## **Tcl/Tk**

The following terms apply to the all versions of the core Tcl/Tk releases, the Tcl/Tk browser plug-in version 2.0, Tcllib, and TclBlend and Jacl version 1.0. Please note that the TclPro tools are under a different license agreement. This agreement is part of the standard Tcl/Tk distribution as the file named "license.terms".

### **Tcl/Tk License Terms**

This software is copyrighted by the Regents of the University of California, Sun Microsystems, Inc., Scriptics Corporation, Ajuba Solutions, and other parties. The following terms apply to all files associated with the software unless explicitly disclaimed in individual files.

The authors hereby grant permission to use, copy, modify, distribute, and license this software and its documentation for any purpose, provided that existing copyright notices are retained in all copies and that this notice is included verbatim in any distributions. No written agreement, license, or royalty fee is required for any of the authorized uses. Modifications to this software may be copyrighted by their authors and need not follow the licensing terms described here, provided that the new terms are clearly indicated on the first page of each file where they apply.

IN NO EVENT SHALL THE AUTHORS OR DISTRIBUTORS BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF THIS SOFTWARE, ITS DOCUMENTATION, OR ANY DERIVATIVES THEREOF, EVEN IF THE AUTHORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

THE AUTHORS AND DISTRIBUTORS SPECIFICALLY DISCLAIM ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT. THIS SOFTWARE IS PROVIDED ON AN "AS IS" BASIS, AND THE AUTHORS AND DISTRIBUTORS HAVE NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

GOVERNMENT USE: If you are acquiring this software on behalf of the U.S. government, the Government shall have only "Restricted Rights" in the software and related documentation as defined in the Federal Acquisition Regulations (FARs) in Clause 52.227.19 (c) (2). If you are acquiring the software on behalf of the Department of Defense, the software shall be classified as "Commercial Computer Software" and the Government shall have only "Restricted Rights" as defined in Clause 252.227-7013 (c) (1) of DFARs. Notwithstanding the foregoing, the authors grant the U.S. Government and others acting in its behalf permission to use and distribute the software in accordance with the terms specified in this license.

---

## **zlib**

### **License**

/\* zlib.h -- interface of the 'zlib' general purpose compression library version 1.2.8, April 28th, 2013

Copyright (C) 1995-2013 Jean-loup Gailly and Mark Adler

This software is provided 'as-is', without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software.

Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgement in the product documentation would be appreciated but is not required.
2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
3. This notice may not be removed or altered from any source distribution.

Jean-loup Gailly (jloup@gzip.org)      Mark Adler (madler@alumni.caltech.edu)

\*/

# Index

---

## Symbols

- #define 4-64
- #elif 4-64
- #else 4-64
- #error 4-64
- #if 4-64
- #ifdef 4-64
- #ifndef 4-64
- #include 4-64
- #line 4-64
- #pragma
  - envvar 1-32, 1-33
  - prune\_assign 1-34
  - PXL encrypt 1-53
- #undef 4-64

## A

- account file 3-13
- acct file 3-13
- ANTLR open source software, license F-4
- argument bindings 4-57
- arithmetic conversions, standard 4-45
- assignment operators 4-27
- associativity and precedence 4-31

- Auto-Incremental DRC Focused on Differing Layers and Regions B-29

## B

- batch queuing systems, support for 1-47
- bbox file 3-13
- bct command-line option 1-7
- binary operator 4-22
- binary qualifier 4-68, 4-69
- bitwise operators 4-29
- boolean, type 4-33
- Boost open source software, license F-4
- bounding box file 3-13

## C

- C command-line option 1-8
- c command-line option 1-8
- C libraries, passing data to 5-1
- cache files 1-42
- cache-only command-line option 1-8
- cell name prefix, overriding with command-line option 1-14
- cell name, command-line option 1-8, 1-21
- cell\_lvs.log file 3-14
- cell, overriding with command-line option 1-8

- cell.acct file 3-13
- cell.bbox file 3-13
- cell.err file 1-6
- cell.LAYOUT\_ERRORS file 2-10, 3-6
- cell.LVS\_ERRORS file 3-12
- cell.net file 3-13
- cell.RESULTS file 3-3
- cell.rules file 3-7
- cell.sp file 3-13
- cell.sum file 2-10, 3-7, 3-16
- cell.tree files 3-9
- cell.vcell file 3-9
- checkpoint command-line option 1-9
- checkpoint runset 1-9
- classifications, error 8-15
- clf command-line option 1-9
- command-line, IC Validator tool
  - options 1-6
- comments 4-22
- compare directory 3-15
- compare log file 3-14
- compare tree files 3-14
- comparing doubles 4-45
- compiler directives 1-28
  - #pragma envvar 1-32, 1-33
- concatenation
  - lists 4-40
  - strings 4-46
- concatenation of strings 4-5
- conditional operator 4-28
- const, qualifier 4-68, 4-69
- constraint of, type 4-34
- conventions for documentation 1-xv
- cpydb command-line option 1-10
- cPYDB, see error classification database
- Custom Compiler tool, running IC Validator tool
  - with 1-5
- Custom Compiler tool, see *Custom Compiler tool* 1-5

- customer support 1-xvi

## D

- D command-line option 1-10
- data types 4-32, 4-33
- database files, output 2-10
- DATE macros 4-67
- db\_name 8-4
- db\_path, setting 8-4
- debug command-line option 1-10, 1-11
- debug option D-2
- debugger, PXL D-1
  - commands D-3
- #define 4-64
- deprecated, qualifier 4-68, 4-69
- design flow 1-2
- designs, small 1-20
- device leveling process file 3-14
- distributed log file 3-9
- distributed processing 1-39
- distributed processing options configuration
  - file 1-40
- distributed statistics file 3-10
- distributed work directory 3-10
- dls file 3-14
- double, type 4-33
- dp command-line option
  - IC Validator 1-11
  - LVL utility B-9, B-10
- dp\_options.conf file 1-11, 1-40, B-9
- dp\_work directory 3-10
- dpcache command-line option
  - IC Validator 1-12
  - LVL utility B-10
- dphosts command-line option
  - IC Validator 1-12
  - LVL utility B-10
- dplearn command-line option, IC Validator
  - 1-13



- dp.log file 3-9
- dpm command-line option
  - IC Validator 1-13
  - LVL utility B-11
- dpmem command-line option
  - IC Validator 1-13
- dpopts command-line option
  - IC Validator 1-13
  - LVL utility B-11
- dp.stats file 3-10
- DRC Error Classification 8-1
- DRC output files 3-2
- dynamic threading 1-26
- dynamic-link library support 5-1

## E

- e command-line option 1-14
- ece command-line option 1-14
- editing errors 3-16
- #elif 4-64
- #else 4-64
- encryption 1-53
  - directives 1-53
  - nesting 1-53
- enumerated type elements 4-9
- environment variables 1-32
  - using 4-23
- environment, networked 2-15
- ep command-line option 1-14
- equiv options, augmenting with command-line option 1-14
- equivalence file 3-13
- equivalence summary file 3-15
- equiv.run file 3-13
- #error 4-64
- error classification
  - best practices 8-3

- error classification database
  - format 8-21
  - using 8-2
- error classifications 8-15
- error data 3-10
- error database
  - library, loading E-3
  - name 8-4
  - path 8-4
  - schema E-6
- error file 3-6
- error information, SQLite database E-2
- error output 4-59
- error\_options() function, DRC classification flow 8-2
- errors, fixing 2-10
- ex command-line option 1-14
- exit code 1-6
- exit codes 1-34
- extracted layout, SPICE 3-13

## F

- f command-line option 1-14
- file names, overriding with command-line option 1-19
- flat designs 2-10
- flatten command-line option, LVL utility B-4
- for loop 4-50
- foreach loop 4-51
- formats, overriding with command-line option 1-19
- functions
  - call
    - operator-style 4-56
    - standard 4-55
  - definitions 4-52
  - overloading 4-58
  - prototypes 4-53

## G

- g command-line option 1-15
- GDSII file, input database 2-9
- graphic output database, smaller with hierarchical input database 2-10
- graphical error database 3-10
- group files 2-9
- group path name, overriding with command-line option 1-15

## H

- handle, type 4-34
- hash of
  - a hash 4-47
  - type 4-41
- HERCULES\_DEVICE license 2-5
- HERCULES\_DRC license 2-5
- HERCULES\_ERC license 2-6
- HERCULES\_LVS license 2-6
- HERCULES\_MASK license 2-6
- HERCULES-DP\_MT license 2-9
- HERCULES-NETLIST license 2-9
- Hierarchical DRC Error Classification 8-3
- hierarchy optimizations 8-2
  - command-line option 1-22

## I

- I command-line option 1-16
- i command-line option 1-15
- IC Compiler II tool, running IC Validator tool with 1-3
- IC Compiler tool, running IC Validator tool with 1-2
- ICV\_LICENSE\_WAIT environment variable 2-2
- icv\_lidb utility

- command-line options C-15
- exit status C-17
- output files C-18
- icv\_lidb\_report utility
  - command-line options C-18
  - output files C-19
- icv\_lvl executable, *see also* LVL utility B-1
- ICValidator2-CompareEngine license 2-3
- ICValidator2-GeometryEngine license 2-3
- ICValidator2-Manager license 2-3
- icvread command-line option 1-17
- identifiers 4-3
  - #if 4-64
  - if command-line option 1-49
  - if statement 4-49
  - #ifdef 4-64
  - #ifndef 4-64
  - ihic command-line option 1-15
  - il command-line option 1-18
  - ilic command-line option 1-18
  - iln command-line option 1-19
- in\_out, qualifier 4-69
- #include 4-64
- include file, command-line option 1-16
- incremental layer checking 1-18, 1-19
- In-Design flow, keywords 4-7
- initializing variables 4-24
- inline qualifier 1-35, 4-69
  - definition 4-69
- inlined functions 1-35
- installation guide, link to 2-15
- installing IC Validator tool 2-1
- integer type 4-33

## K

- keywords 4-6
- keywords, In-Design flow 4-7

## L

- lay.cell file 3-16
- layer mapping file 1-19, 1-49, B-16
  - OpenAccess B-13
  - overriding with command-line option 1-23
  - specifying 1-19
- Layout Integrity Management C-1
- layout netlist file
  - extracted 3-13
  - partitioned 3-16
  - SPICE 3-13
- LAYOUT\_ERRORS file 3-6, 3-16
- lay.tree file 3-14
- lf command-line option 1-19
- libdp open source software, license F-6
- library definition file, overriding with
  - command-line option 1-23
- library format, command-line option 1-14
- library format, overriding with command-line
  - option 1-14
- library name, overriding with command-line
  - option 1-15
- library path, command-line option 1-23
- library path, overriding with command-line
  - option 1-23
- libxml2 open source software, license F-7
- license waiting 2-2
- licenses, open source F-1
- licensing
  - determining needed 2-3
  - function 1-15, 2-4
  - technology 1-18, 2-3
- licensing schemes, for IC Validator tool 2-2
- limits, system 2-15
- #line 4-64
- list of a list 4-48
- lists, concatenation 4-40
- literal, qualifier 4-69
- ln command-line option 1-19

- lnf command-line option 1-19
- log file 3-9
- logical operators 4-28
- loops 4-50
- LSF (Load Sharing Facility) support 1-47
- LVL utility B-1
  - command-line options B-3
  - comparing a subset of layers B-17
  - error files B-21
  - exit status B-21
  - layout formats supported B-3, B-24, B-29
  - output B-20
  - use models B-23
  - using layer files B-15
- LVS debugging file 3-12
- LVS output files 3-2, 3-11
- LVS\_ERRORS file 3-12

## M

- mapping file, layer 1-19, 1-49, B-16
- MCCP public domain code, license F-19
- member reference 4-30
- memory usage 2-10
  - reported 2-12
- milkyway\_version command-line option 1-20
- MIT license F-7
- ml command-line option 1-20

## N

- naming restrictions 4-3
- native licensing 2-2
- ndg command-line option 1-20
- ndm\_version command-line option 1-21
- net file 3-13
- netlist-versus-netlist flow 1-6, 1-22
- NET-TRAN license 2-8
- networked environment 2-15

## NFS settings 2-15

- nho command-line option 1-22
- norscache command-line option 1-22
- nro command-line option 1-22
- numeric constants 4-6
- numeric operators 4-29
- nv command-line option 1-22
- nv\_netlist\_mode command-line option 1-22
- nwl command-line option 1-22

## O

- oa\_layer\_map command-line option 1-23
- oa\_lib\_defs command-line option 1-23
- oa\_view command-line option 1-23
- obsolete, qualifier 4-69
- Open Source Initiative, licenses F-7
- open source software, licenses F-1
- OpenAccess
  - layer mapping file B-13
  - LVL Utility B-11
- operating systems 2-15
- operators 4-22
- optimizations for hierarchy, command-line option 1-22
- out, qualifier 4-69
- output database files 2-10
- output files 3-2

## P

- p command-line option 1-23
- page faults 2-10
- parameter passing 4-46
- Pattern Matching 9-1
- pc command-line option 1-23, 1-24
- performance tips
  - hierarchical database produces smaller error output 2-10

- keep files on local machine for faster execution 2-11
- provide enough memory to avoid accessing swap space 2-10

- #pragma
  - envvar 1-32, 1-33
  - PXL encrypt 1-53
- precedence and associativity 4-31
- preprocessor 4-63
- primitive types 4-33
- program structure 4-59
- program version, command-line option 1-26
- prune layers 1-34
- prune\_assign command-line option 1-24
- PXL debugger D-1
- pxlcrpt utility 1-53
- pydb directory 3-10
- PYDB Perl API 4-60, E-1
- pydb\_export utility 8-17
- pydb\_report utility 8-13
- PYDB, see error database
- PYDB.pm E-3
- Python open source software, license F-31

## Q

- qualifiers, language specification 4-68

## R

- rd command-line option 1-24
- relational expression 4-27
- reserved words 4-6
- resource requirements 2-9
- restart runset 1-9
- RESULTS file 3-3
- ro command-line option 1-24
- rules file 3-7
- run details directory, command-line option 1-24

- run details directory, overriding with command-line option 1-24
- run-only execution 1-31
- runset caching 1-42
- runset encryption 1-53
- runset optimization, turn off with command-line option 1-22
- runtime, reported 2-12

## S

- s command-line option 1-25
- sch.cell file 3-16
- schematic netlist file, partitioned 3-16
- sch.tree file 3-14
- scoping 4-52
- selecting rules 1-26
- sf command-line option 1-25
- sfn command-line option 1-26
- SGE (Sun Grid Engine) support 1-47
- shared licensing 2-2
- signoff\_check\_drc, IC Compiler II command 1-3
- signoff\_create\_metal\_fill, IC Compiler II command 1-3
- signoff\_drc, IC Compiler command 1-2
- signoff\_fix\_drc, IC Compiler II command 1-3
- signoff\_metal\_fill, IC Compiler command 1-2, 1-3
- small-design processing 1-20
- SolvNet
  - accessing 1-xvi
  - documentation 1-xiv
- sp file 3-13
- SPICE netlist file 3-13
- SQLite database, error information E-2
- StarRC tool, running IC Validator tool with 1-4
- stc command-line option 1-25
- string concatenation 4-46
- string literals 4-5
- string, type 4-33
- sum.cell.cell file 3-15
- summary file 3-7
- summary file, runtime and memory usage 2-12
- svc command-line option 1-26
- svn command-line option 1-26
- swap space 2-9, 2-10
- symbolic names
  - define, command-line option 1-10
  - undefine, command-line option 1-26
- system limits 2-15
- system requirements 2-10
  - flat database 2-10
  - reducing memory with bottom up error fixing 2-10
  - swap space 2-9, 2-10
- systems, operating 2-15

## T

- Tcl/Tk open source software, license F-41
- ternary operator 4-22
- text, handling in LVL utility B-19
- threads command-line option
  - LVL utility B-11, B-25
- top cell, overriding with command-line option 1-25
- tree structure files 3-9
- turbo command-line option
  - IC Validator 1-26
  - LVL utility B-11
- type
  - boolean 4-33
  - constraint of double 4-34
  - constraint of integer 4-34
  - double 4-33
  - enum of 4-38
  - hash of 4-41
  - integer 4-33
  - list of 4-38
  - newtype 4-37

- primitive 4-33
- string 4-33, 4-34
- struct of 4-43, 4-44
- type analysis 4-44
- type promotion 4-45
- typing, data 4-32

## U

- U command-line option 1-26
- ufn command-line option 1-26
- unary operator 4-22
- #undef 4-64
- unselecting rules 1-26
- usage command-line option 1-26
- user-defined types 4-36
- using in runsets 4-22
- uvc command-line option 1-26
- uvn command-line option 1-26

## V

- V command-line option 1-26
- values, default injection 4-47
- variable, named 1-32, 1-33

- variables, initializing 4-24
- vcell file 3-9
- Version command-line option 1-27
- VERSION macros 4-66
- version numbers, command-line option 1-27
- vfn command-line option 1-27
- view overriding with command-line option 1-23
- violation blocks 3-10
- violations 4-59
- virtual cell file 3-9
- VUE
  - command-line option 1-27
  - extract short data, command-line option 1-28
- vue command-line option 1-27
- VUE debugging 3-10
- vueshort command-line option 1-28

## W

- while loop 4-51
- wireLog file 1-22

## Z

- zlib open source software, license F-42