SIEMENS EDA

Calibre® Pattern Matching User's Manual

Software Version 2021.2
Document Revision 14

SIEMENS

Support Center: support.sw.siemens.com
Send Feedback on Documentation: support.sw.siemens.com/doc_feedback_form

Revision History

Revision	Changes	Status/ Date
14	Modifications to improve the readability and comprehension of the content. Approved by Lucille Woo. All technical enhancements, changes, and fixes listed in the *Calibre Release Notes* for this products are reflected in this document. Approved by Michael Buehler.	Released April 2021
13	Modifications to improve the readability and comprehension of the content. Approved by Lucille Woo. All technical enhancements, changes, and fixes listed in the *Calibre Release Notes* for this products are reflected in this document. Approved by Michael Buehler.	Released January 2021
12	Modifications to improve the readability and comprehension of the content. Approved by Lucille Woo. All technical enhancements, changes, and fixes listed in the *Calibre Release Notes* for this products are reflected in this document. Approved by Michael Buehler.	Released October 2020
11	Modifications to improve the readability and comprehension of the content. Approved by Lucille Woo. All technical enhancements, changes, and fixes listed in the *Calibre Release Notes* for this products are reflected in this document. Approved by Michael Buehler.	Released July 2020

Author: In-house procedures and working practices require multiple authors for documents. All associated authors for each topic within this document are tracked within the Siemens EDA documentation source. For specific topic authors, contact the Siemens Digital Industries Software documentation department.

Revision History: Released documents maintain a revision history of up to four revisions. For earlier revision history, refer to earlier releases of documentation which are available on https://support.sw.siemens.com/.

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction to Calibre Pattern Matching

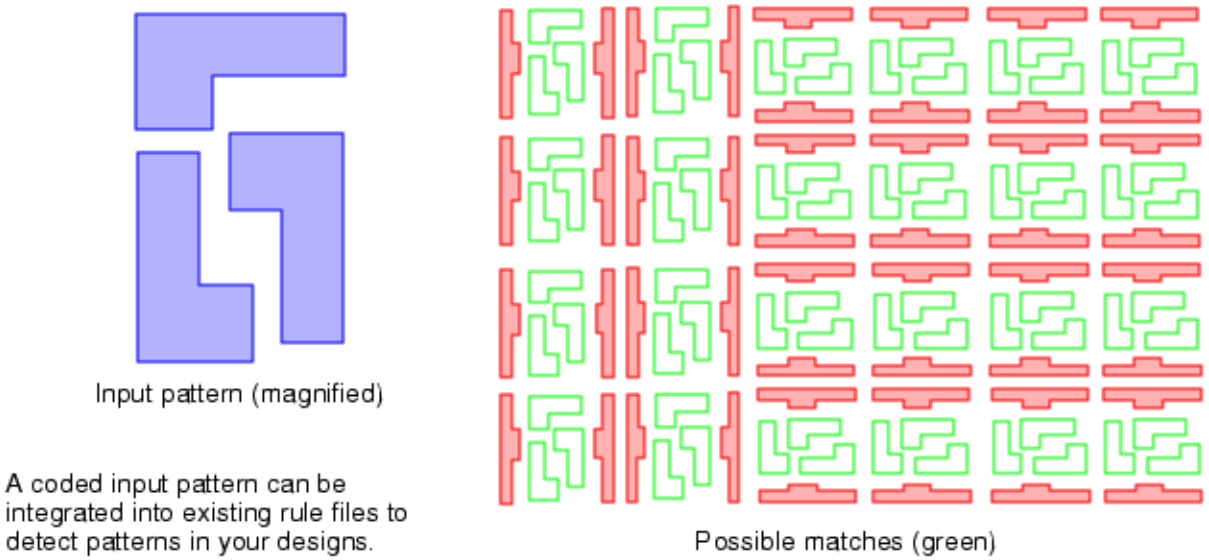
Calibre® Pattern Matching in physical verification is the process of locating specific polygons in a design from a user-specified pattern. You can use Calibre Pattern Matching to simplify certain Calibre® nmDRC™ rule checks which are complex to code in Standard Verification Rule Format (SVRF) syntax. Rule file development time may also be shortened when existing rule files are combined with Calibre Pattern Matching.

Calibre Pattern Matching Overview

Pattern matching enables you to find targeted patterns for special Calibre nmDRC checks and identify known lithographic problem areas.

Pattern matching can also assist in device classification by generating auxiliary layers to be used in LVS Device specification statements. However, pattern matching does not consider connectivity. You cannot use Calibre Pattern Matching for complete LVS verification.

Figure 1-1. Input Pattern with an Array of Matches



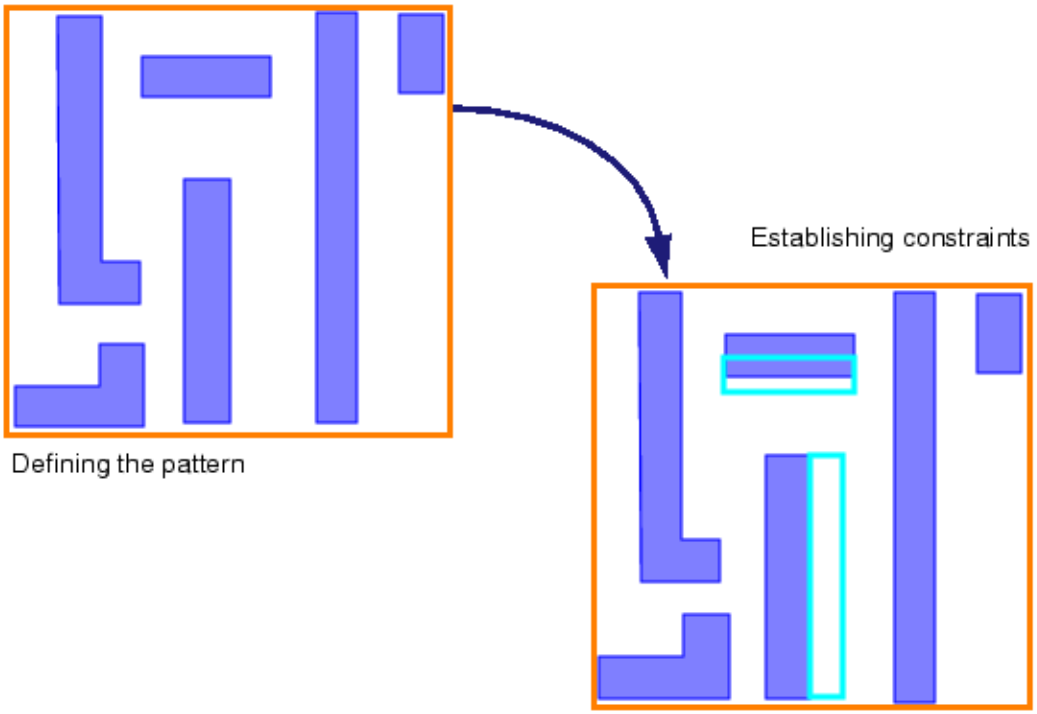
Calibre Pattern Matching allows you to control the matching behavior with defined constraints. Patterns can consist of multiple polygons and multiple layers. Patterns can be matched in any mirrored or rotated orientation.

A pattern can be drawn with any number of angled edges, even all edges being angled, however the placement of a pattern in the design must be at a 0, 90, 180, or 270 degree rotation.

Note

Matching of skew pattern edges (those that are not at a 90, 180, or 270 degree rotation) could result in a missed match due to snapping. This occurs when the edge is not able to align with the grid in the layout, and thus “snaps” onto nearby grid points, which results in a loss of accuracy.

Figure 1-2. Defining the Pattern and Establishing Constraints



Pattern matching allows you to locate exact matches, matches of edges based on a range of constraint values, and polygons shifted with respect to the input pattern. Patterns are defined in the Calibre Pattern Matching GUI. or captured from an existing layout.

Pattern matching is not a replacement for existing DRC methods, but an extension for checks difficult to code using typical methods. You integrate this new capability into your existing physical verification flows by building up your pattern libraries with the Calibre Pattern Matching GUI. Pattern matching can reverse the trend of adding large and complex DRC rules by providing an alternate and far simpler method of dealing with the most complex checks required for advanced processes.

Pattern Matching Components

The major components of Calibre Pattern Matching simplify complex rule checks required for advanced IC processes by replacing text-based design rule checks with a visual geometry.

The components are detailed below:

- **Calibre Pattern Matching GUI** — The GUI is used for creating and editing pattern libraries. You can also run the pattern library utilities from the GUI.

- **Pattern Library Manager Utility** — Pattern libraries can be managed with the Pattern Library Manager Utility. The utility is invoked at the command line. You can compile and merge libraries, among other actions.
- **Pattern Matching API** — The pattern matching Tcl-based API is a batch interface that allows you to edit and examine pattern libraries. The pattern matching API interface is the described in the *Calibre Pattern Matching Reference Manual*.
- **Pattern Library** — A pattern library is a collection of one or more patterns created with the Calibre Pattern Matching GUI or captured using one of the pattern capture operations. When the pattern library is complete, the pattern library is compiled into an SVRF DMACRO pattern file.
- **SVRF DMACRO Pattern File** — Pattern libraries are compiled into SVRF DMACRO pattern files. These files contain encrypted DMACRO calls that contain pattern matching operations. The Include SVRF specification statement and CMACRO command are used to call the compiled DMACRO pattern library file for pattern matching.

  See “DMACRO and CMACRO for Pattern Matching” on page 79.
- **SVRF DFM Pattern Capture and DFM Pattern Classify Operations** — These SVRF operations are used to generate pattern libraries. The DFM Pattern Capture operation captures patterns on specified lithographic hotspots or based on a specified mask layer. The DFM Pattern Classify operation identifies and classifies patterns based on interactions between polygons on target layers and a seed or pattern mask layer.

  See DFM Pattern Capture and DFM Pattern Classify in the *Standard Verification Rule Format (SVRF) Manual*.
- **DFM RDB or Standard RDB** — Matched results are returned as marker shapes on a derived polygon layer. If the pattern library included properties, those properties are attached to the marker shapes as DFM properties. The markers shapes can be saved to a DFM RDB or a standard DRC results database (RDB), as with other derived layers and DRC results. The RDB can be viewed in Calibre® RVE™, a graphical debug and results viewing tool. This process is outlined in “Using Calibre RVE to View Match Results” on page 187.

Calibre Pattern Matching Product Requirements

There are licensing and environment variable requirements for the Calibre Pattern Matching tool.

- **Licensing**

  The Calibre Pattern Matching tool requires a Calibre Pattern Matching license. See “Calibre Pattern Matching” in the *Calibre Administrator’s Guide*.

Running pattern matching within Calibre® nmDRC™ or Calibre® nmDRC-H™ requires a Calibre Pattern Matching license as well as the Calibre nmDRC or Calibre nmDRC-H license.

Running pattern capture from a Calibre layout viewer (Calibre® DESIGNrev™ or Calibre® WORKbench™) requires the layout viewer license and the Calibre Pattern Matching license.

- **Environment Variable**

  CALIBRE_HOME — Required variable that is used to specify the path of the Calibre software tree. Refer to "CALIBRE_HOME Environment Variable" in the *Calibre Administrator's Guide*.

- **Temporary Directory**

  The Calibre Pattern Matching GUI requires access to a writable directory for temporary files. The directory */usr/tmp* is used by default. To specify an alternate directory, set the environment variable CALIBRE_PMATCH_UI_TMP_DIR to the directory path.

Best Practices for Calibre Pattern Matching

This section provides some guidelines for running Calibre Pattern Matching.

Managing Pattern Libraries for Best Performance

Calibre pattern libraries target specific layers. All patterns within a library need to share the same target layers. Libraries should be combined when patterns target the same input layers, for example M1 V1 patterns should all exist within the same library. Otherwise, the patterns should remain in separate libraries. For example, patterns for poly layers should be in a separate library than patterns for contact layers. All Patterns that exist in the same library run concurrently, so having patterns from the same layers in one library ensures the best performance.

Properties are another feature that plays an important role in runtime and memory. Every marker layer generated in pattern matching receives the complete list of properties from a library. This means that a property named TYPE set for only pattern1 will be output to the marker polygons for all other pattern in the library as well. When a property is not set for a given pattern, the marker shape will receive the property with value -1. An additional consideration for performance with properties is the number of matches as every marker polygon output by pattern matching will contain the complete list of properties from the library.

Identifying the Source Pattern

There are two ways to identify the source pattern for each match location in the results of a pattern matching run:

- **Property** — Add a property to all patterns, where the property contains a unique numeric value for each pattern. This can be done as follows, depending on how you create the pattern library:
  - With DFM Pattern Capture — Use the "INDEX_PROP *index_prop_name*" keyword set. This adds the property *index_prop_name* with the pattern ordinal to each pattern.
  - With DFM Pattern Classify — Use the "class" property, which is automatically added to each pattern. See the command reference for details.
  - With the Calibre Pattern Matching GUI — Add the property and unique numeric value manually to all patterns. See "Defining Custom Property Values in a Pattern" on page 159.
  - With the merge utility — Use the argument set "index *index_prop*" to add or update the property *index_prop* in the merged library.

  It is particularly useful to use count markers in combination with the unique property value if you want to identify the source pattern for a match. See "Count Markers" on page 45 for details.
- **Unique Marker Name** — Create a unique marker name for each pattern using the Calibre Pattern Matching GUI and give the marker a type other than empty. Compile the library using pdl_lib_mgr compile with the gen_all_markers option. Each pattern in the compiled DMACRO has all markers, but only the marker created for the corresponding pattern is non-empty and produces geometries on the marker output layer.

  Note

  Calibre Pattern Matching can generate up to 278 unique marker names concurrently. Exceeding this limit is similar to running two pattern matching calls to the library.

Avoiding Keyword - Name Conflicts in the Calibre Run

User-defined names in pattern matching should not conflict with keywords in the SVRF rule file; see "Keyword - Name Conflicts" in the *Standard Verification Rule Format (SVRF) Manual*. User-defined names also should not conflict with reserved words in the Tcl programming language. In pattern matching, this restriction applies to marker names, pattern layer names, pattern names, library names, and the input and output layer parameters in the CMACRO command. For example, you should not name a pattern marker "extent" or "extents" because those strings correspond to an SVRF layer operation. Prefixing marker names with a string, for example "m_", can avoid this problem. A similar practice can be used for other names.

For example, this CMACRO command causes an error because the output layer name “extent” is a reserved word in SVRF:

```
CMACRO pm:libM2 m2 via2 " " match_m2 extent // ERROR
```

This works:

```
CMACRO pm:libM2 m2 via2 " " match_m2 m_extent
```

Hierarchical vs. Flat Results

Pattern matching results are typically reported at the top or higher levels of hierarchy due to promotion, ensuring accurate matches. Using the Push operation pattern matching output can increase the hierarchy of the output.

Note

The Push operation does not preserve property data so properties need to be reattached to the pushed layer using DFM Property.

When DFM Pattern Capture or DFM Pattern Classify is run with a relatively flat hotspot or mask layer, then it is critical to run with TURBOflex enabled in the rule file.

Modes of Operation

Topological pattern matching in physical verification is used to find targeted placements of polygons, misaligned polygons, lithographic hotspots, and DRC rule checks that are difficult to code using SVRF syntax.

Calibre Pattern Matching has three pattern types: Topological Edge Match (TEM), Barcode Match (BCM), and crossbar (XBAR) patterns. For most pattern matching applications, TEM patterns are used to define a pattern. BCM is used for one-dimensional or barcode-like patterns.

For additional details on pattern types, refer to “Patterns” on page 25.

Syntax Conventions

The command descriptions use font properties and several metacharacters to document the command syntax.

Table 1-1. Syntax Conventions

Convention	Description
Bold	Bold fonts indicate a required item.
Italic	Italic fonts indicate a user-supplied argument.

Table 1-1. Syntax Conventions (cont.)

`Monospace`	Monospace fonts indicate a shell command, line of code, or URL. A bold monospace font identifies text you enter.
Underline	Underlining indicates either the default argument or the default value of an argument.
UPPercase	For certain case-insensitive commands, uppercase indicates the minimum keyword characters. In most cases, you may omit the lowercase letters and abbreviate the keyword.
[ ]	Brackets enclose optional arguments. Do not include the brackets when entering the command unless they are quoted.
{ }	Braces enclose arguments to show grouping. Do not include the braces when entering the command unless they are quoted.
‘ ’	Quotes enclose metacharacters that are to be entered literally. Do not include single quotes when entering braces or brackets in a command.
| or ||	Vertical bars indicate a choice between items. Do not include the bars when entering the command.
…	Three dots (an ellipsis) follows an argument or group of arguments that may appear more than once. Do not include the ellipsis when entering the command.

Example:

DEVice {***element_name*** [‘(’*model_name*‘)’]}
device_layer {***pin_layer*** [‘(’*pin_name*‘)’] …}
[‘<’*auxiliary_layer*‘>’ …]
[‘(’*swap_list*‘)’ …]
[BY NET | BY SHAPE]

Chapter 2
Calibre Pattern Matching Key Concepts

Several key principles and concepts pertain to pattern matching.

Pattern Libraries

A collection of patterns is stored in a Pattern Matching Database (PMDB), also called a pattern library.

The pattern library can be created using different methods:

- With the Calibre Pattern Matching GUI — See "Creating a New Pattern Library" on page 94.
- With the DFM Pattern Capture operation — "Generating and Merging PMDB Library Files" on page 172.
- With DFM Pattern Classify operation — See the command reference.

The pattern library can be viewed and edited in the Calibre Pattern Matching GUI. See "Calibre Pattern Matching GUI Basics" on page 82.

Once your pattern library is complete, it must be compiled into a format that can be used for a pattern matching run. See "DMACRO and CMACRO for Pattern Matching" on page 79.

You can compare libraries with two methods:

- pdl_lib_mgr compare utility — Compares pattern geometries, extents, regions, and markers.
- Calibre Pattern Matching Reference Library Flow — Compares pattern geometries and pattern extents using the DFM Pattern Classify operation. This method is more efficient than the compare utility for large pattern databases.

The rest of this section discusses guidelines for pattern libraries.

Restrictions and Guidelines for Pattern Libraries

There are several restrictions and guidelines for all pattern libraries.

Naming rules:

- Names can contain only alphanumeric characters (a-z, A-Z, and 0-9) and underscores (_). They cannot contain spaces. The naming rules apply to library names, keys, property names, and constraint labels.

Pattern libraries must abide by the following:

- All patterns in a pattern library must have the same input layers and the input layers must be listed in the same order.

- Each pattern in a pattern library can have its own precision value.

The following pattern objects must be Manhattan geometries:

- Pattern extents.
- Pattern shapes in BCM and XBAR patterns. In addition, these patterns should not have jogs.

Caution

Non-Manhattan pattern polygon edges should not intersect the pattern extent, due to the potential of off-grid snapping, which changes the pattern geometry.

Also see "Best Practices for Calibre Pattern Matching" on page 17.

About Merging Pattern Libraries

You can merge two or more pattern libraries into one library. The merging process removes patterns that are duplicates. Combining libraries is useful because it reduces the number of pattern matching calls. However, care must be taken that the libraries being merged have the same layer list.

By default, patterns are considered duplicate if the pattern shape, layer extents, regions, and markers are identical.

Only exact patterns are evaluated for duplicate removal, meaning any pattern with constraints (with the exception of cglobal) is passed directly to the output library. If two otherwise identical patterns have different cglobal values, the larger value is used in the pattern saved to the output library.

Merging pattern libraries can be useful for capturing errors from different rule checks and then combining them into one pattern library. For example, if a lithographic hotspot is found on a layer, it can be saved as a pattern that can be searched on multiple other metal layers with the same design requirements. Merging libraries when patterns are used for the same purpose, such as identifying lithographic hotspots, can reduce the number of calls to the pattern matching engine and thus improve performance.

Caution

When merging libraries, the tool only uses the layer definitions from the first input library. It is highly important that layer names, order, and number of layers are identical for all the input libraries. An error is reported if the number of layers differ between input libraries and the merge does not take place.

When layers definitions differ, patterns defined in subsequent libraries use the layer definitions from the first input library of the merge operation, resulting in patterns that do not apply to the intended layer and incorrect pattern matches.

For example, consider the following two libraries and their incorrectly merged output library:

Table 2-1. Incorrect Output Library for Merged Libraries

Library 1	Library 2	Output Library
Layers:	Layers:	Layers:
1. M1	1. V1	1. M1
2. M2	2. M1	2. M2
3. V1	3. M3	3. V1

The two libraries are merged, where Library 1 is the first input library. The layer names for the patterns that were in Library 2 are now as follows: V1 is now named M1, M1 is now named M2, and M3 is now named V1. The patterns from Library 2 now apply to different layers than originally written. To avoid this error, be consistent with the layer names and order within your libraries.

You can add Don't Care pattern layers to libraries so that a two or more libraries have the same number and order of pattern layers. See "Don't Care Pattern Layers" on page 55.

You can merge libraries from the GUI or with the merge utility. See "Merging Pattern Libraries Using the GUI" on page 99 and merge. The merge command reference discusses details regarding the handling of custom properties and pattern orientation.

Patterns

A pattern defines the basic structure of the polygons Calibre searches for in a pattern matching run.

Calibre Pattern Matching has three types of patterns to cover all topological pattern matches. The pattern type is determined automatically by the Calibre Pattern Matching GUI. These are the supported pattern types:

- Topological Edge Match (TEM) — Most patterns include at least one vertex that is not on the pattern extent.
- Barcode Match (BCM) — One dimensional patterns in which all vertices are on the pattern extent. All pattern edges must be parallel.
- Crossbar Pattern (XBAR) — Multilayer patterns in which each layer is a BCM pattern and edges on at least two of the layers are perpendicular to each other.

All patterns require layers and edges. TEM patterns also require vertices. Each pattern has a limit of up to 278 input layers.

Vertices in Patterns

There are two types of vertices: real and virtual. Overlapping vertices are allowed in TEM patterns, but they have special considerations.

- **Real Vertex**

  A real vertex lies inside the pattern extent. A real vertex must match a vertex in a matched layout region. A constraint on an edge adjacent to a real vertex affects the location of the vertex.

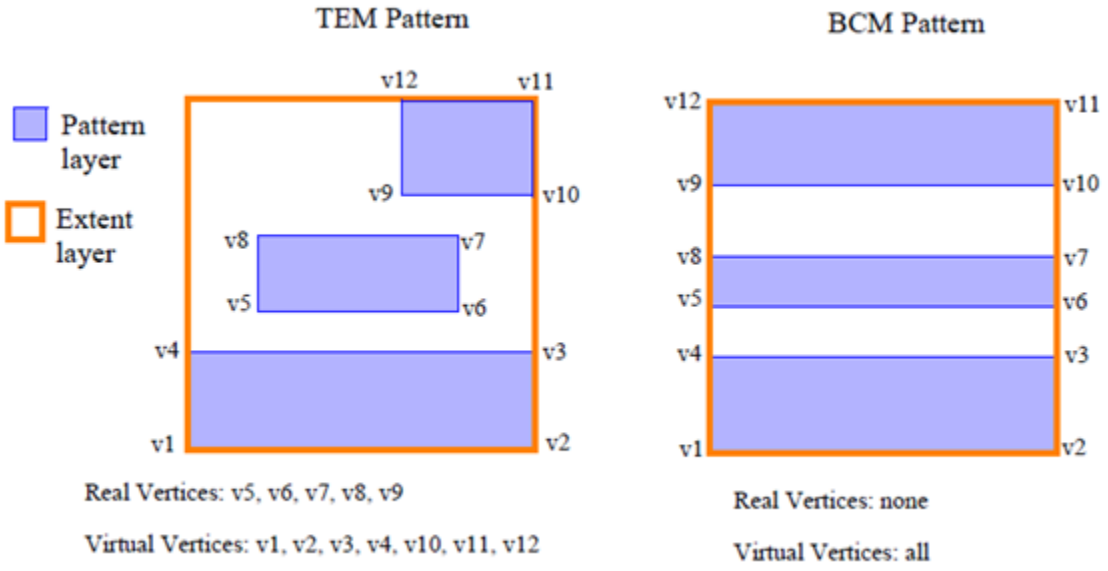
  BCM and XBAR patterns do not have any real vertices.

- **Virtual Vertex**

  A virtual vertex lies on the pattern extent. A virtual vertex is not required to match a vertex in a matched layout region.

The following figure shows the different types of vertices.

Figure 2-1. Vertices



TEM patterns can have overlapping real vertices on the same pattern layer. For example, given a pattern that is drawn using layer metal1, it is possible to include metal1 polygons in the pattern that have overlapping real vertices.

You can create TEM patterns with overlapping real vertices in one of two ways:

- Draw polygons with overlapping real vertices in the Pattern Matching GUI.
- Use the Pattern Matching GUI capture flow or DFM Pattern Capture to capture a pattern from a layout.

Note

Overlapping real vertices are not allowed in patterns with cglobal definitions. Calibre Pattern Matching generates an error in this case.

Figure 2-2 shows an example of a pattern with overlapping real vertices. Vertex v6 from polygon P2 is at the same location as vertex v12 from polygon P3.

Figure 2-2. Pattern with Overlapping Vertices

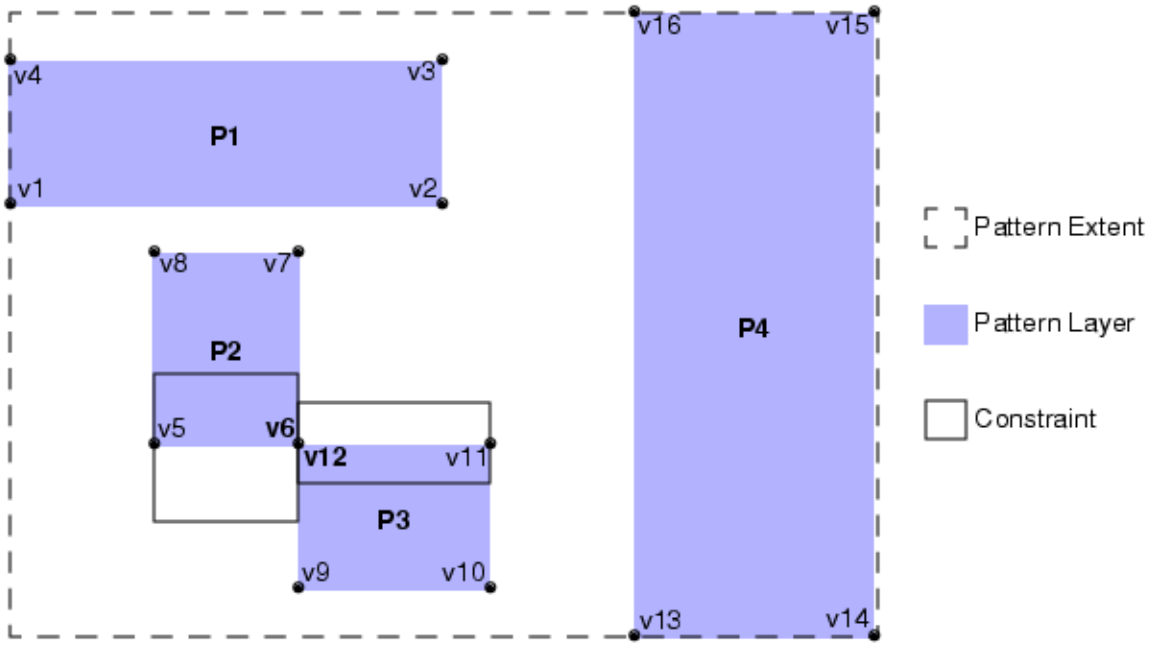
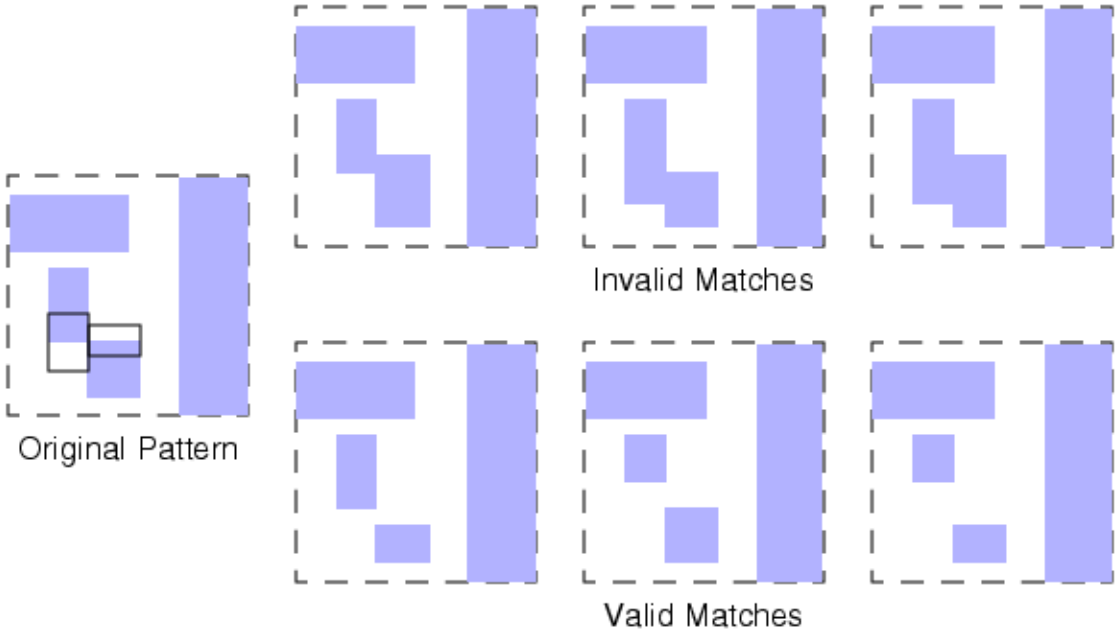


Figure 2-3 shows some possible valid results and non-matches from the TEM pattern shown in Figure 2-2. As long as the specified constraints do not result in a circuit topology change, Calibre Pattern Matching returns a valid match. The constraint definitions in the original pattern are shown for reference.

Figure 2-3. Examples of Matching a Pattern with Overlapping Real Vertices



Related Topics

Edges in Patterns

Capturing Patterns From a Layout

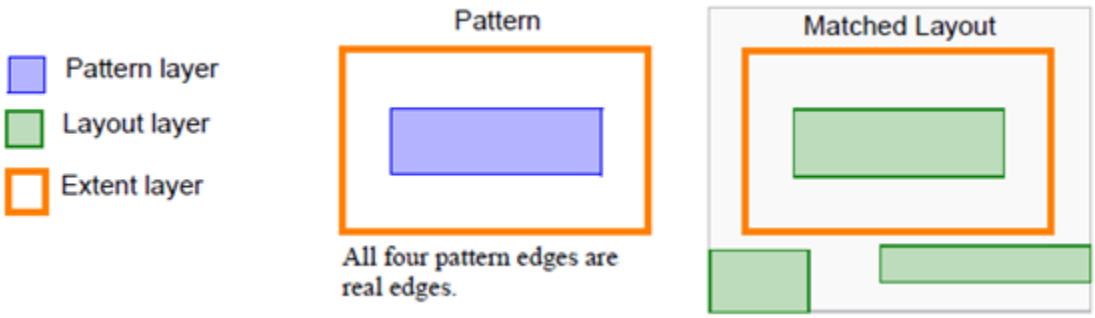
Edges in Patterns

An edge is the line segment between two vertices. There are three types of pattern edges: real, partial, and virtual.

- **Real Edge**

  A pattern edge that is completely enclosed in the pattern extent. A real pattern edge must match an edge in a matched layout region. An edge with no constraints is termed a fixed edge. Constraints affect the location of the edge.

Figure 2-4. Real Edges on a Pattern



BCM and XBAR patterns do not have any real edges.

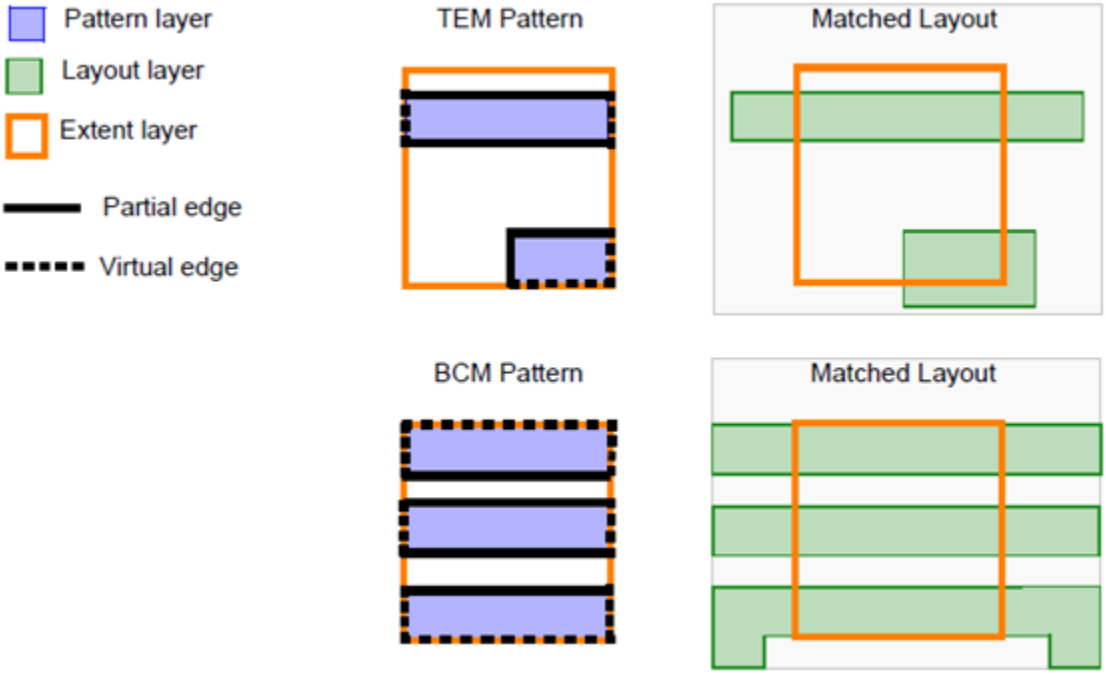
- **Partial Edge**

  A pattern edge that has one or two vertices that lie on the pattern extent. At least one adjacent edge must be a virtual edge. A partial edge does not have a defined length in the matched layout region. See Figure 2-5.

- **Virtual Edge**

  A pattern edge that is coincident with the pattern extent. Virtual edges are not required to match an edge in the layout.

Figure 2-5. Virtual and Partial Edges in a Pattern

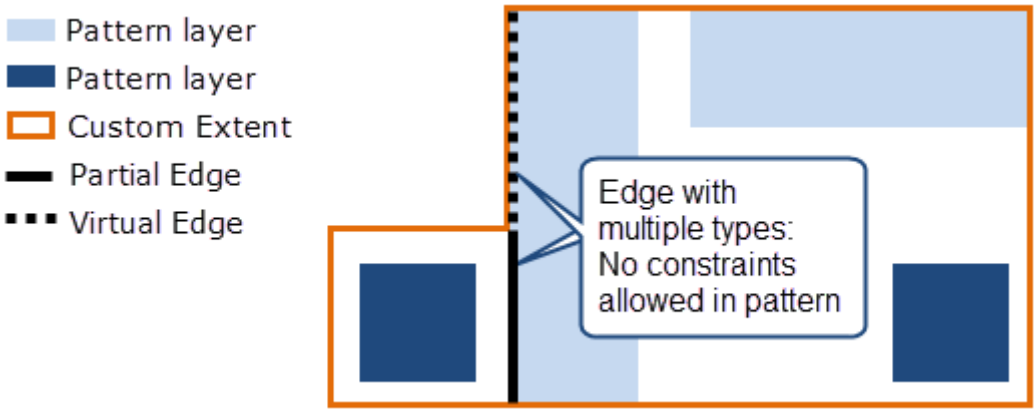


Pattern edges with more than one type are discussed in the next section.

Edges With More Than One Type

In some cases a pattern edge may have more than one type along its length. In particular, this can happen when the pattern extent is not rectangular, as shown in the next figure. If an edge with multiple types exists in a pattern, the pattern cannot have constraints.

Figure 2-6. Edge With Mixed Types



If you require constraints in the pattern, you may be able to edit the pattern to avoid the edge with multiple types. Depending on the intent of the pattern, a Don't Care Region can be used instead of a custom layer extent to achieve the same effect, or the custom pattern extent can be moved. Both solutions are shown in the following figure.

Related Topics

Constraints

Vertices in Patterns

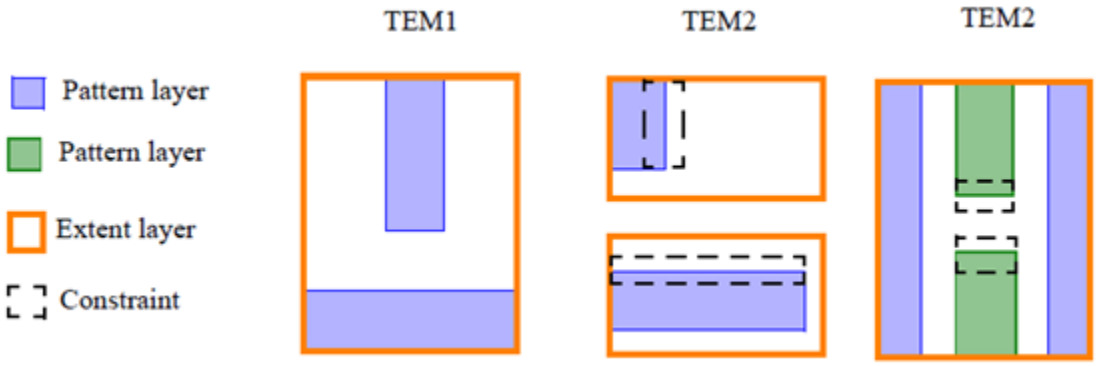
Extent Layer

Topological Edge Match (TEM) Patterns

Topological Edge Match (TEM) patterns are the default pattern type.

The following figure shows some TEM pattern examples.

Figure 2-7. TEM Pattern Examples



For multilayer TEM patterns, one or more layers may be full or empty layers, but at least one layer must have a vertex or edge.

TEM patterns are categorized into the following performance classes:

Class TEM1 — Fast performance. TEM1 patterns satisfy all of these conditions:

- No constraints
- Single rectangular extent

Most TEM1 patterns have at least one real vertex.

Class TEM2 — Average performance. TEM2 patterns satisfy at least one of these conditions:

- Constraints
- Non-rectangular extent, multiple extents, layer extents, or regions

Class INVALID — Pattern does not compile due to errors.

Related Topics

Marker Types for TEM and XBAR Patterns

Extent Layer

Vertices in Patterns

Full and Empty Pattern Layers

Pattern Summary

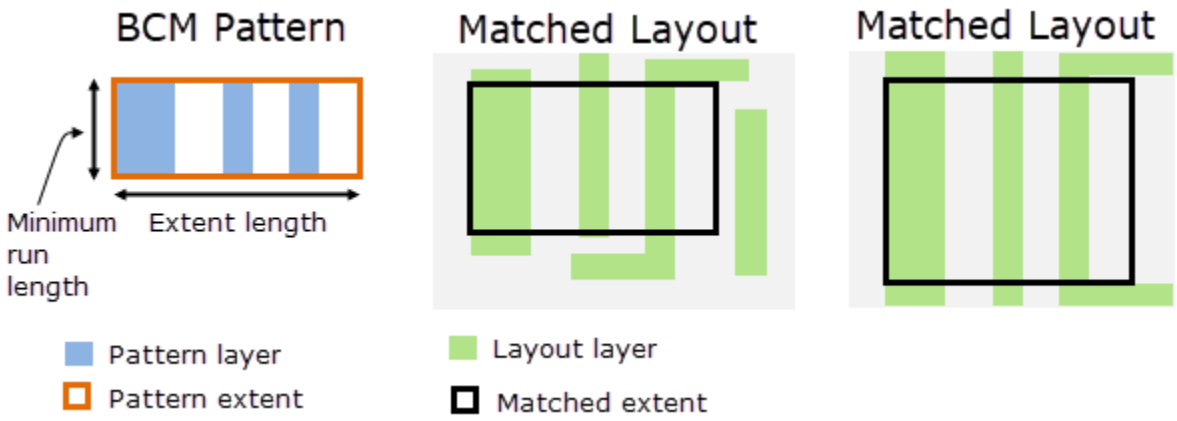
Barcode Match (BCM) Patterns

Barcode Match (BCM) patterns are used to find one-dimensional or barcode-like patterns. In contrast to TEM patterns, BCM patterns have only four unique orientations.

BCM patterns have only partial and virtual edges and no real vertices. All edges in a BCM pattern must be parallel, including marker edges. BCM patterns have a single rectangular extent. Multilayer BCM patterns must have at least one layer with a partial edge.

A simple BCM pattern with no constraints is shown in the following figure. The figure defines the extent length and run length. The figure also shows two possible matches in a layout.

Figure 2-8. BCM Pattern (Basic)



A match must meet the minimum run length. There is no maximum run length.

Caution

Single edge BCM patterns, while allowed, produce matches for all parts of the layout. This has a negative impact on runtime for pattern matching, and is thus not recommended.

There are two classes of BCM patterns, determined by the extent type:

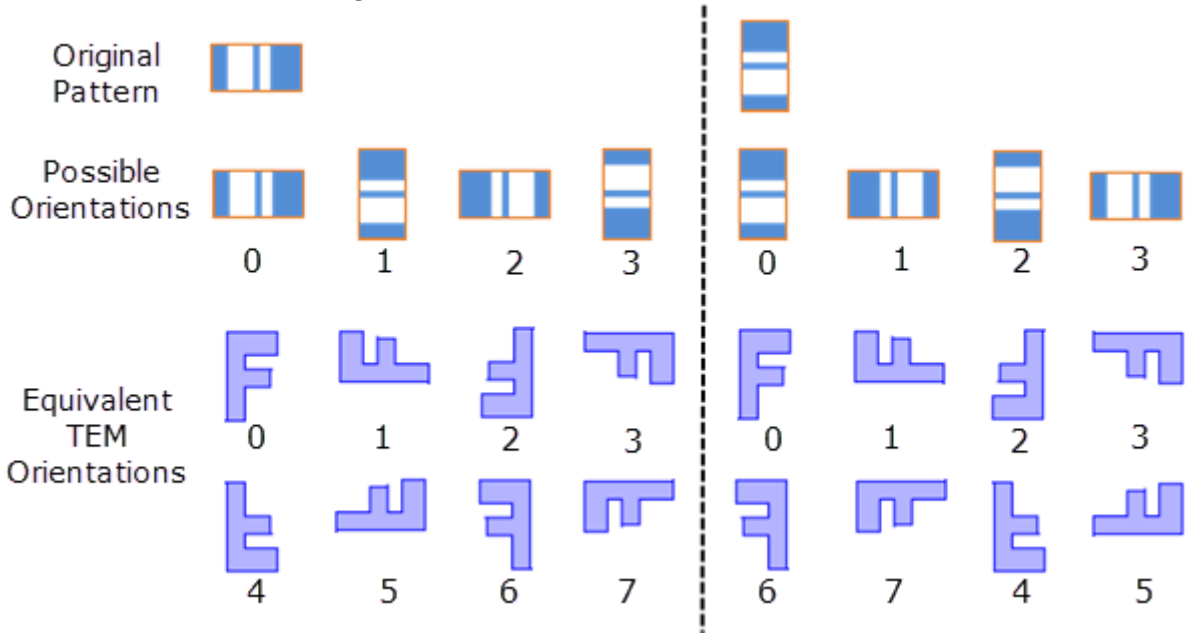
- **Class BCM1** — Accordion Extent. The width of a wire or the width of the gap between wires can change due to constraints. The constraints expand and contract the extent length of the matched pattern. See “Edge to Edge Constraints in Accordion Extent BCM Patterns (BCM1)” on page 63.
- **Class BCM2** — Fixed Extent. The extent length is unchanged when constraints change the width of a wire or the gap between wires. See “Edge to Edge Constraints in Fixed Extent BCM Patterns (BCM2)” on page 62.

Orientation in BCM Patterns

BCM patterns have only four unique orientations because they are one-dimensional, in contrast to the eight possible orientations for a TEM or XBAR pattern. The orientation of a match to a BCM pattern is reported as 0, 1, 2, or 3. If a BCM pattern is saved with an orientation of 4, 5, 6, or 7 and matched in the same orientation, the matched orientation is converted to the equivalent orientation in the range 0-3.

The following figure shows the possible orientations for two BCM patterns, one with vertical wires and one with horizontal wires. The equivalent TEM pattern orientation in the range 4-7 is also shown.

Figure 2-9. BCM Pattern Orientations



Related Topics

Marker Types for BCM Patterns

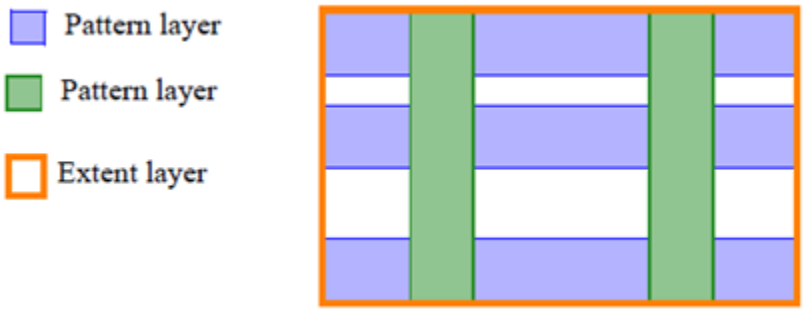
Pattern Summary

Full and Empty Pattern Layers

Crossbar (XBAR) Patterns

Crossbar patterns are multilayer patterns in which each layer is a BCM pattern and edges on at least two of the layers are perpendicular to each other. They have no real vertexes.

The following figure shows an example.



Crossbar patterns cannot have constraints, regions, multiple extents, layer extents, or non-rectangular extents. These features are disabled for crossbar patterns.

Related Topics

Barcode Match (BCM) Patterns

Marker Types for TEM and XBAR Patterns

Pattern Summary

Pattern Summary

Some pattern features are only supported for certain pattern types.

The following table summarizes the pattern features supported for each pattern type.

Table 2-2. Summary of Pattern Features by Pattern Type

Feature	TEM	BCM	XBAR
Constraints that specify movements	Yes	Yes	
Global variability (global constraints)	Yes	Yes	
Matching exact features	Yes	Yes	Yes
Attaching properties to output	Yes	Yes	Yes
Pattern library generation with SVRF	Yes	Yes	Yes
Multiple output markers	Yes	Yes	Yes
Customized extent regions	Yes		
Multiple extent definitions	Yes		
Match maximum run lengths without additional constraints		Yes	

Table 2-2. Summary of Pattern Features by Pattern Type (cont.)

Feature	TEM	BCM	XBAR
Support accordion (or flexible) extent		Yes	

Related Topics

Layers

Vertices in Patterns

Edges in Patterns

Constraints

Extent Layer

Properties

Pattern Clustering

Calibre Pattern Matching enables you to group similar patterns into clusters. Clustering can be performed using either the Calibre Pattern Matching GUI or the pdl_lib_mgr cluster utility.

Similarity in patterns is defined as the edge-position difference between two or more patterns. Patterns are clustered by adding a cluster ID property to each pattern. Patterns in the same cluster have the same value of the cluster ID property. See cluster for detailed information on behavior and options.

You open the Cluster Library GUI by choosing **Tools > Cluster Library** in the Calibre Pattern Matching GUI . See “Defining Pattern Clusters in a Library” on page 110.

The batch pdl_lib_mgr write_oasis utility and the Generate OASIS tool in the Calibre Pattern Matching GUI can visualize groups using the cluster property information. The OASIS®[1] output from each cluster is placed in its own OASIS cell.

Related Topics

write_oasis

Exporting an OASIS File

1. OASIS® is a registered trademark of Thomas Grebinski and licensed for use to SEMI®, San Jose. SEMI® is a registered trademark of Semiconductor Equipment and Materials International.

Layers

Different layers are generated at various stages of pattern library development.

For all patterns, extent layers and markers play a pivotal role in how patterns are detected and matches are reported.

A pattern can have empty layers, where a pattern layer is defined but no geometries on that layer exist within the extent. An empty layer can be useful to check that a metal pattern does not contain any vias within an area.

Requirements and default behaviors of layers in Calibre Pattern Matching are

- Patterns must contain one or more layers.
- Patterns require the use of an extent layer (default or custom).
- A default marker layer called ‘Marker’ is created automatically for patterns, but it can be customized for a different returned result.

Extent Layer

The extent layer defines the borders of the pattern and isolates the pattern from surrounding geometries. For TEM patterns, you can define multiple extents, including an extent for the pattern as a whole and extents for individual layers. BCM and XBAR patterns have a single, rectangular extent.

The polygons that fall within the extent are considered for a match, while polygons outside of the extent are not considered. By default, the pattern extent is rectangular, but custom extents can be defined for TEM patterns.

Note

Extents must be Manhattan geometries, containing only 90 degree angles.

Caution

Non-Manhattan pattern polygon edges should not intersect the pattern extent, due to the potential of off-grid snapping.

You can define these extents:

- **Default Extent**

  The default extent of a pattern is the minimum bounding box that encloses the pattern. BCM and XBAR patterns support only the single default extent. The default extent applies to all layers in the pattern unless a custom or per-layer extent is specified. Any custom or per-layer extent has priority over the default extent.

- **Per-Layer Extent**

  TEM patterns support per-layer extents. During the matching process, layer polygons outside the per-layer extent are not considered. Per-layer extents can be a custom extent or the default extent for the layer.

- **Custom Extent**

  A custom extent is an extent you define in the Calibre Pattern Matching GUI and can be any Manhattan polygon. It can be defined for the pattern as a whole or as a per-layer extent. Custom extents enable more control over the region that is matched than is possible with the default extent. Custom extents are supported only for TEM patterns.

- **Dynamic Extent**

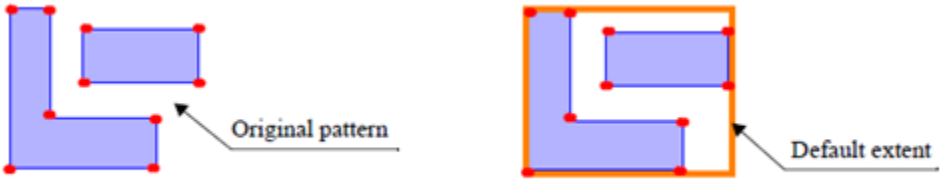
  A dynamic extent moves with a pattern edge. Dynamic extents are supported only in TEM patterns.

  For BCM patterns, accordion extent (BCM1) patterns have an adjustable extent by definition. Fixed extent (BCM2) patterns have a fixed extent by definition.

Default Extents

The default pattern extent is a minimum bounding box that encloses all polygons and constraints in the pattern. A default pattern extent is created automatically by the Calibre Pattern Matching GUI.

The following figure shows the default extent.



Custom Extents

A custom extent is a polygonal boundary that is defined for the pattern as a whole or for a specific layer. A custom extent enables a different extent shape than the default extent, which is the pattern bounding box. Polygons outside the custom extent are not considered in the matching process, therefore custom extents enable more control over the region that is matched than is possible with the default extent. Custom extents can be considered as a way to define a Don’t Care region.

The custom extent for the pattern must enclose all polygons, along with any defined constraints, otherwise the pattern is changed. Likewise, a per-layer custom extent must enclose all polygons, including constraints, for the layer.

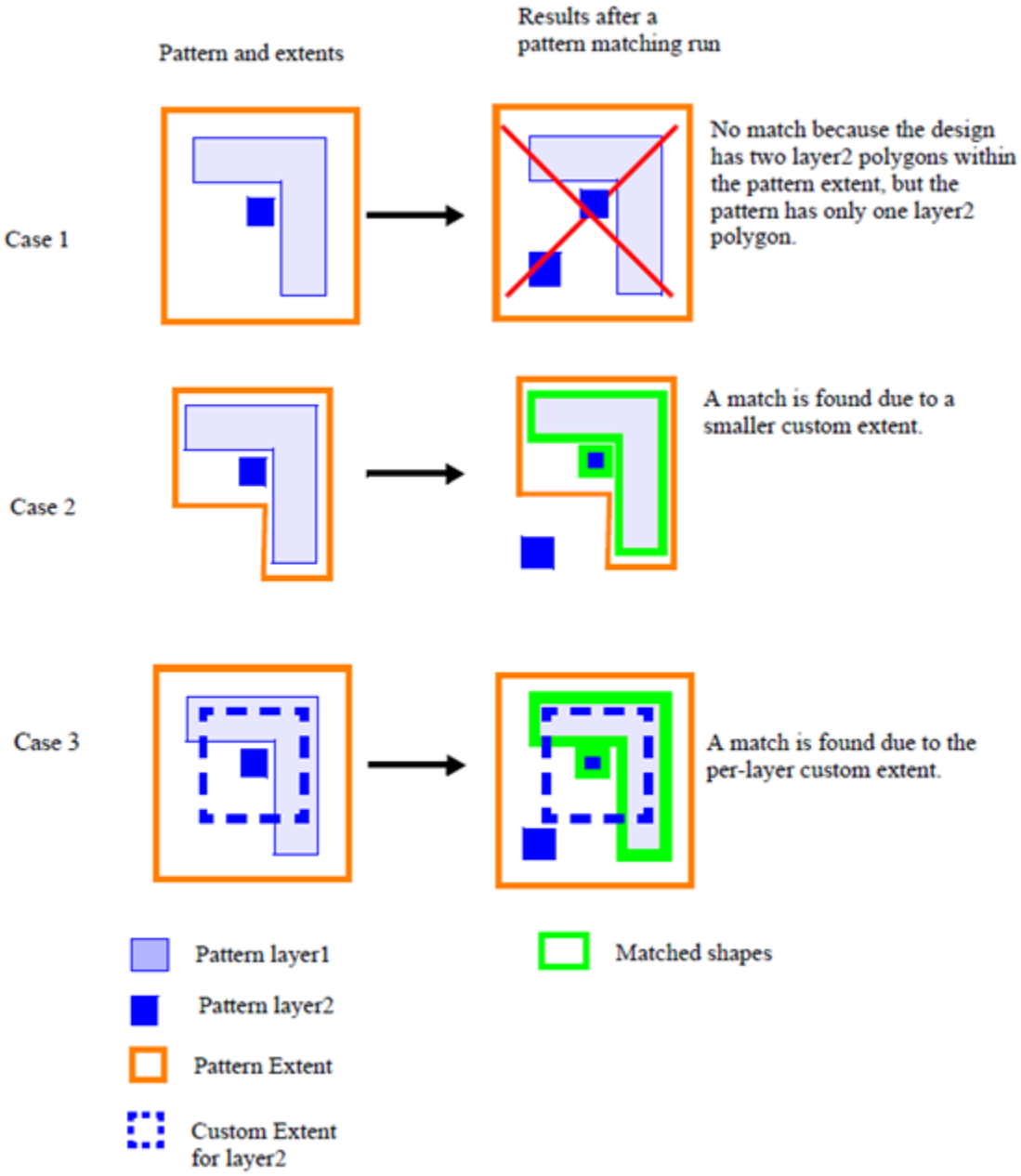
Custom extents and per-layer extents are only supported in TEM patterns.

For details on how to define a custom extent using the Calibre Pattern Matching GUI, see “Defining Custom and Per-Layer Extents for a TEM Pattern” on page 130.

You can capture patterns with per-layer custom extents using the DFM Pattern Capture and DFM Pattern Classify SVRF operations. When patterns are captured using a per-layer halo setting, the per-layer halo is saved as a per-layer custom extent. A per-layer extent is shown in “Case 3” of Figure 2-10.

The following figure shows the effect of using a custom extent.

Figure 2-10. Matching with a Custom Extent



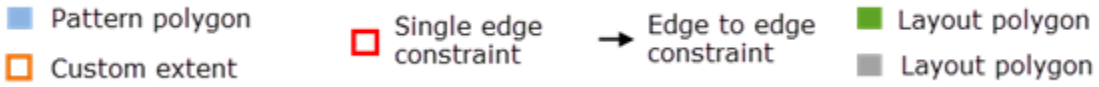
Dynamic Extents

A pattern extent is a polygon that defines the limits of a pattern. Dynamic extents shrink or expand in size with a constrained pattern edge. Dynamic extents are defined using the Calibre Pattern Matching GUI.

Dynamic extents are supported for custom extents in TEM patterns. A dynamic extent is specified as an edge to edge constraint with a fixed distance from the extent to a pattern edge

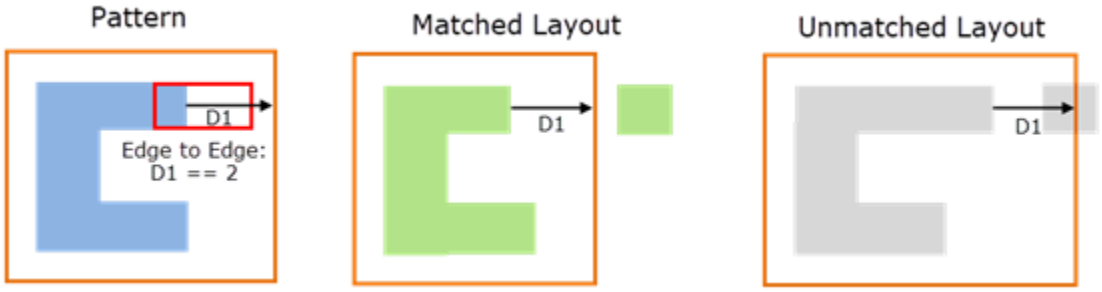
that has a single edge constraint. See “Specifying a Dynamic Extent for a TEM Pattern” on page 150. Some examples of dynamic extents follow.

These conventions are used in the figures:



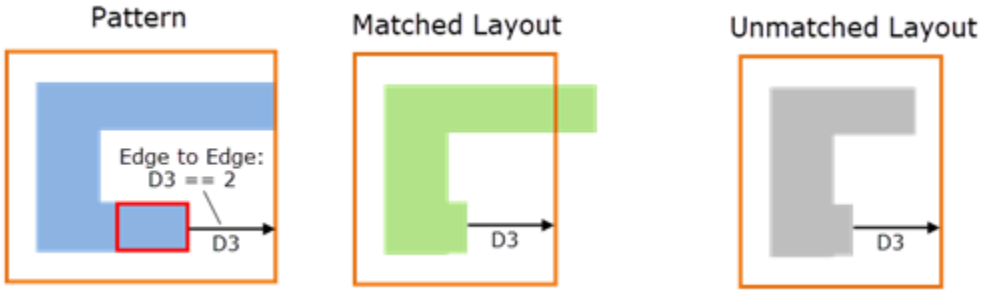
In the following figure, the pattern extent must be a fixed distance D1 from the upper line end, which has a single edge constraint. The layout in the middle is a match because the square shape is outside the pattern extent. The layout on the right does not match because the square shape is within the pattern extent.

Figure 2-11. Basic Dynamic Extent



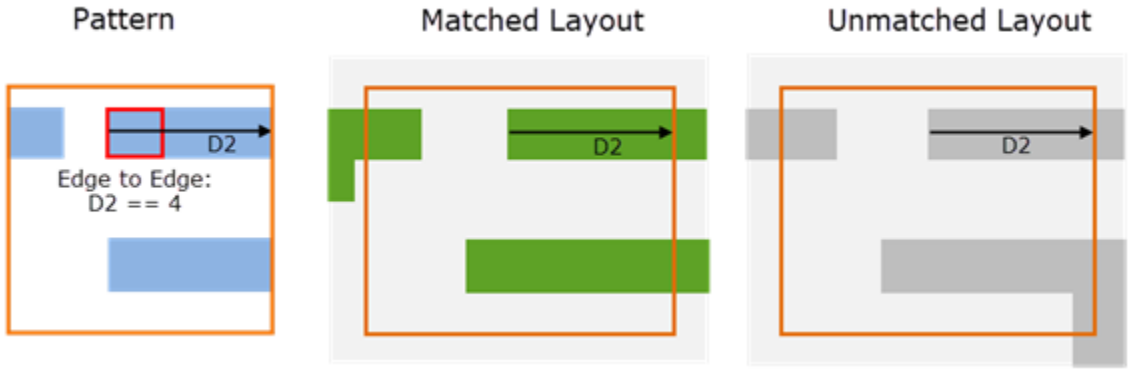
The behavior when a dynamic extent moves inward with a virtual edge at the extent is shown in the following figure.

Figure 2-12. Dynamic Extent Moving in From Virtual Edge



When a virtual edge is at a dynamic extent that moves outward, the virtual edge also moves outward, as shown in the following figure. For the unmatched layout example on the far right, the layout would match to the pattern if the dynamic extent were removed.

Figure 2-13. Dynamic Extent Moving Out From Virtual Edge



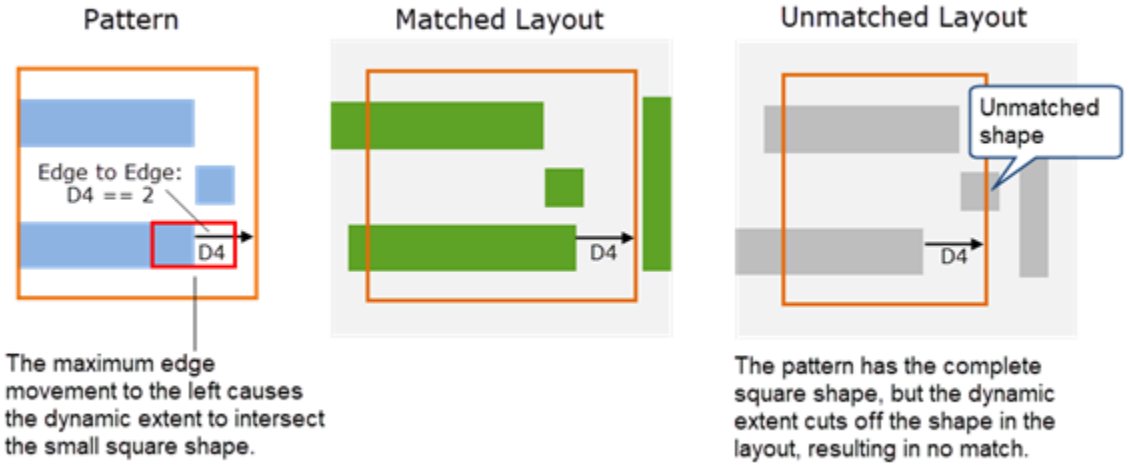
Dynamic extents can be used in multilayer patterns. You can specify a dynamic extent to a custom pattern extent or to a custom layer extent.

Dynamic Extents to Avoid

A dynamic extent cannot change the pattern topology—it only affects the layout shapes that are considered for a match. For this reason, certain dynamic extent constructions should be avoided.

The allowed movement of a dynamic extent should not cross a real pattern edge, as shown in the following figure. The real pattern edge remains in the pattern, but the dynamic extent causes only a portion of the layout shape to be included in the matching process.

Figure 2-14. Avoid: Dynamic Extent That Crosses a Real Pattern Edge



Related Topics

Edge to Edge Constraints in TEM Patterns

Relational Constraints

Marker Layer

Marker layers are output layers generated by the pattern matching tool. The shapes on the marker layer indicate the locations of pattern matches. The typical practice is to output the marker layer to the results database and/or a DFM RDB file.

You can have multiple marker layers output from a pattern matching run. You can specify up to 278 concurrent marker layers. Exceeding this amount will cause an increase in runtime. Marker layers can be used in the same way as a derived SVRF layer; for example, they can be used as an input layer to an SVRF operation.

Pattern matching results appear on the marker layer to highlight a matched pattern. The marker is placed in the same orientation as the detected match. For TEM and BCM patterns, you can add any number of marker types for each match. The marker types are described in "Marker Types for TEM and XBAR Patterns" on page 42 and "Marker Types for BCM Patterns" on page 44.

You can add markers to a pattern to show where a match was found. Markers only need to be defined for individual patterns if custom markers are desired or for computed markers needed for compile-time marker overrides. Markers defined in a pattern override compile-time markers with the same name. Only custom markers must be associated with a pattern. All other pattern marker types can be chosen at compile time. A recommended usage is to define custom markers for patterns and use compile-time markers to output the typical bbox, bbox10, drawn, and matched type markers.

The marker layers output by a pattern matching run are determined by the markers in the pattern library and the settings in the compile command used to create the DMACRO used in pattern matching run. See compile and "DMACRO and CMACRO for Pattern Matching" on page 79. When naming marker layers, make sure the marker name is not the same as any SVRF command name or keyword; see "Avoiding Keyword - Name Conflicts in the Calibre Run" on page 18.

Marker Types for TEM and XBAR Patterns

The possible marker types for TEM and XBAR patterns are BBox10, BBox, Custom, Drawn, Empty, and Matched. These markers return results for each matched pattern.

For additional information on how to define these markers, refer to "Adding Markers to a Pattern" on page 125.

- BBox10

  A BBox10 output marker returns a marker that is approximately ten percent of the minimum bounding box for the pattern, including the pattern extent. For example, if the pattern extent is 1.6 um x 1.6 um, the returned default marker is 0.16 um x 0.16 um. This is the default marker type for TEM and XBAR patterns.

  Caution

  Because BBox10 is calculated at approximately 10% of the pattern's bounding box, it is possible the marker falls between database units, resulting in snapping. The snapping direction may be orientation dependent. An example where this may be problematic is when using the SVRF Push operation on the marker layer; a 1 dbu difference could cause only part of the marker layer to push. It is recommended that you use one of the other marker types (including Custom) in this case.

- BBox10(*layername*)

  BBox10(*layername*) returns a marker that is approximately ten percent of the minimum bounding box for the specified layer, including the layer extent. If a layer extent is not defined, the pattern extent is used.

- BBox

  A BBox output marker returns a marker that is the minimum bounding box for the pattern, including the pattern extent.

- BBox(*layername*)

  A BBox(*layername*) output marker returns a marker that is the minimum bounding box for the specified layer, including the layer extent. If a layer extent is not defined, the pattern extent is used.

- Custom

  A Custom marker has a user-defined shape and location. When a Custom marker is defined for a TEM or XBAR pattern, it must fall within the pattern’s default extent. Any number of customized shapes are possible with custom markers, and non-Manhattan shapes can be used.

- Drawn(*layername*)

  A Drawn output marker returns the original input shapes of *layername* defined in the pattern library when a match is found. The *layername* parameter denotes a pattern layer name. For example, if the input pattern layer is named “metal3”, then a marker set to drawn(metal3) outputs the exact shapes defined in the pattern for metal3.

- Empty

  An Empty marker generates no output on the given marker layer for a pattern match. There can be more than one empty marker defined for a pattern, and all markers can be

Empty as well. An Empty marker can be useful when there are multiple output layers, and one of the outputs needs to be empty for use in later SVRF operations.

- Matched(*layername*)

  A Matched output marker returns the original input shapes of *layername* defined in the pattern library after the application of constraints. The *layername* parameter denotes a pattern layer name. For example, if the input pattern layer is named “metal4”, then a marker set to matched(metal4) outputs layout shapes that match the pattern.

  If region layers are used, the matched output polygons are clipped by the region layer; that is the matched output is match_out = layout_layer NOT region_layer.

Related Topics

Topological Edge Match (TEM) Patterns

Crossbar (XBAR) Patterns

Marker Types for BCM Patterns

The possible marker types for BCM patterns are Custom, Empty, Matched, BBox, and BBox10.

- Custom

  A Custom marker has a user-defined shape and location. They are rectangular, Manhattan geometries that extend the full matched run length and must be parallel to the pattern edges (wire edges). There can be any number of Custom marker rectangles, and they can overlap.

  Custom markers in BCM patterns are fixed relative to the pattern extent. Custom markers cannot be used in BCM1 (accordion extent) patterns that have constraints, because the constraints act to change the pattern extent.

  Note

  If a Custom marker is not a rectangle, a rectangular bounding box of the marker shape, stretched in the run length to the ends of the match, is output. Sometimes, a Custom marker does not stretch to precisely the entire pattern run length, meeting both edges of the pattern in the run length direction. In such a case, the marker is modified on output to stretch in the run length direction to the ends of the match. Modified Custom marker patterns are not changed when viewed in the GUI, however a warning is issued.

- Empty

  Generates no output on the given marker layer for a pattern match. There can be more than one empty marker defined for a pattern, and all markers can be Empty as well. An Empty marker can be useful when there are multiple output layers, and one of the outputs needs to be empty for use in later SVRF operations.

- Matched(*layername*)

Returns the geometries in the input that match the pattern. The *layername* parameter denotes a pattern layer name.

- BBox

  Returns a marker that is the minimum bounding box for the pattern, including the pattern extent.

- BBox(*layername*)

  For BCM patterns, the BBox(*layername*) marker returns the same shape as the BBox marker.

- BBox10

  Returns a marker that is approximately ten percent of the minimum bounding box for the pattern, including the pattern extent. For example, if the pattern extent is 1.6 um x 1.6 um, the returned default marker is 0.16 um x 0.16 um. This is the default marker type for BCM patterns.

  Caution

  Since bbox10 is calculated at approximately 10% of the pattern's bounding box, it is possible the marker falls between database units, resulting in snapping. The snapping direction may be orientation dependent. An example where this may be problematic is when using the SVRF Push operation on the marker layer and a 1 dbu difference could cause only part of the marker layer to push. It is recommended that you use one of the other marker types (including Custom) in this case.

- BBox10(*layername*)

  For BCM patterns, the BBox10(*layername*) marker returns the same shape as the BBox10 marker.

Related Topics

Barcode Match (BCM) Patterns

Count Markers

You can add count markers to the results from a pattern matching run. Count markers are small, non-overlapping markers for each matched result. They can be used to determine a count of matched results. Count markers are added when you compile a pattern library.

These sections are included:

- Geometry and Placement of Count Markers
- Adding Count Markers
- Using Count Markers

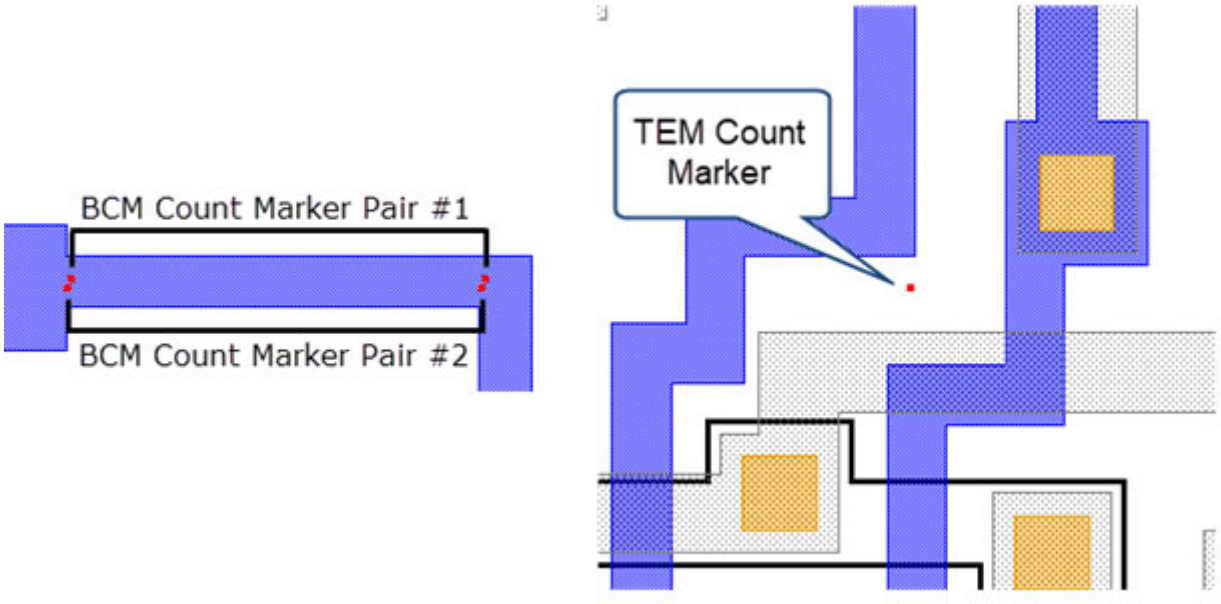
Geometry and Placement of Count Markers

- **TEM and XBAR Patterns** — One 2 x 2 dbu square marker in the center of the match.
- **BCM Patterns** — Two 2 x 2 dbu square markers, where one marker is at each end of the run length of the match.

For all patterns, if a second pattern matches in a similar position, the count marker(s) are shifted slightly to avoid overlap and merging of the marker shapes in the resulting output layer.

The following figure shows count markers for both TEM and BCM patterns. Two BCM pattern matches were found in the image on the left, resulting in two pairs of count markers.

Figure 2-15. Count Marker Geometry and Placement



Adding Count Markers

Count markers are added when you compile a pattern library (PMDB) into a DMACRO.

Tip

It is recommended that each pattern in the PMDB include a property that contains a unique identifier for the pattern. See “Identifying the Source Pattern” on page 18. The property is attached to the all marker output and enables you to determine the pattern that corresponds to the matched result.

You can use either the Calibre Pattern Matching GUI or the pdl_lib_mgr compile utility to specify count markers:

- When compiling a library using the GUI, select **Tools > Compile Library**, select the **Markers** tab, and check the “Enable unique count marker” checkbox.

- When running pattern matching from the GUI, in the Run Matching dialog box, click the **Compile Options** tab, click the options button (), and check “Enable unique count marker” on the **Markers** tab.

  See “Running Pattern Matching From a Calibre Layout Viewer” on page 103 for a link to a video that uses count markers.

- With the pdl_lib_mgr compile utility, specify the gen_count_layer option.

The count marker layer is named match_count.

Using Count Markers

Count markers are handled as any other marker layer when writing the pattern matching rule file. For example, see “Using a DMACRO in a Rule File for a Pattern Matching Run” on page 185. You can include the statement DFM Pattern Match Report within a rule check in order to produce a report which includes the number of matches per pattern.

You can highlight count markers using Calibre RVE as with other marker output. For the following example, the property “pid” was attached to each pattern with a unique pattern index. The pattern marker output is a bbox10 shape. The PMDB was compiled with this command:

```
pdl_lib_mgr compile input input1.pmdb output dmacro.svrf gen_count_layer
```

The following is an excerpt from the pattern matching rule file:

```
LAYER poly 13
INCLUDE "dmacro.svrf"
CMACRO pm:input1 poly " " match  count
match {COPY match}
count {COPY count}
CHECK {
  DFM RDB match "./pat_match.rdb" ALL CELLS CHECKNAME match
  DFM RDB count "./pat_match.rdb" ALL CELLS CHECKNAME count
  DFM PATTERN MATCH REPORT count PM_report.txt ALL
}
```

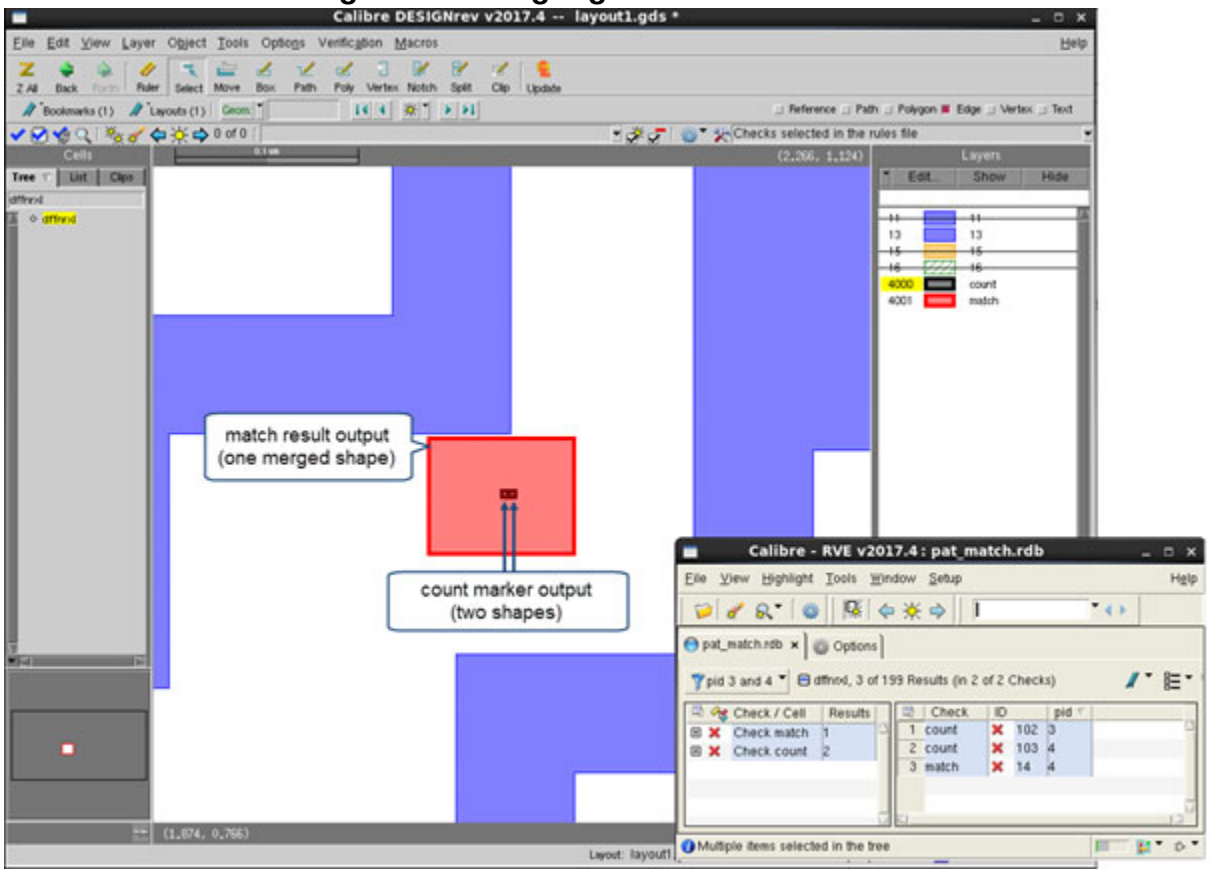
A match to the patterns with pid=3 and pid=4 is found in the same location. The resulting count marker output shapes (the count layer) are unique and do not overlap, resulting in two results for the two matches. The pattern marker output shapes (the match layer) are merged, resulting in one result for the two matches.

The *pat_match.rdb* results file loaded in Calibre RVE is shown in Figure 2-16, along with the highlight of the match and count layers for pid=3 and pid=4. A property filter is applied in Calibre RVE so that only the results of interest are displayed.

When marker shapes with different property values are merged, the largest value of the property is placed on the merged shape. In this example the match result shape has pid=4, as shown in the Calibre RVE display.

By highlighting both the match and count layer results, it is apparent that two pattern matches were found in the same location. The pid property on each count marker indicates which pattern produced the match.

Figure 2-16. Highlighted Count Markers

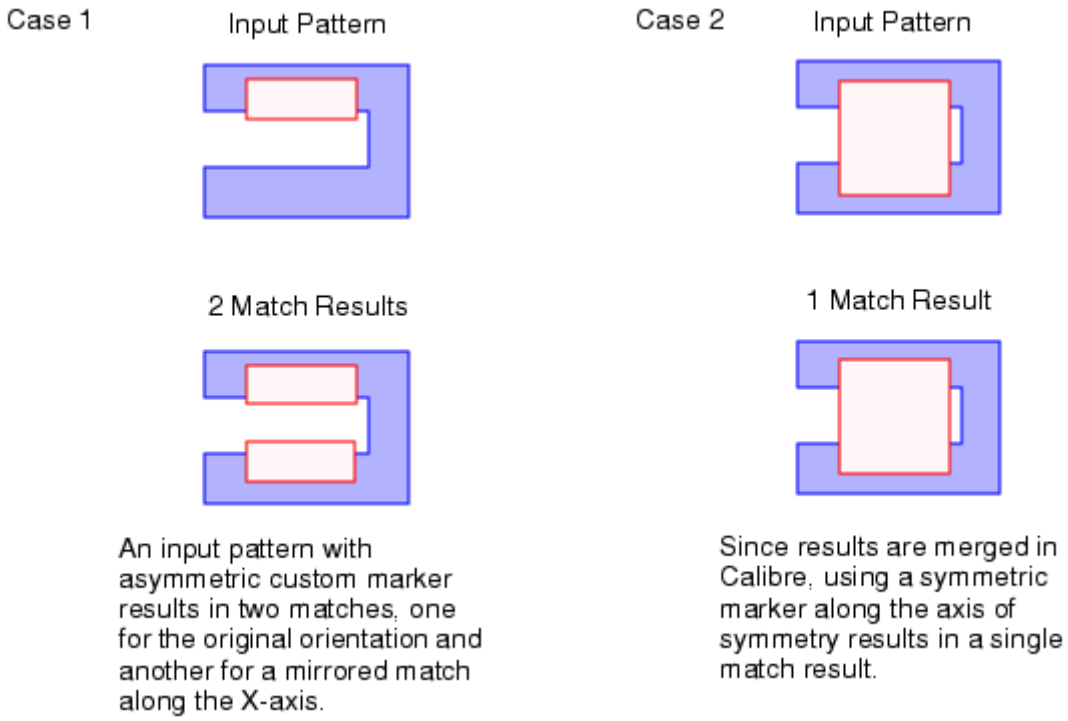


Marker Symmetry and Match Results

For symmetric patterns with an asymmetric marker, pattern matching may return two or more results, signifying more than one match for the same shape because the tool finds all possible rotations and mirrors of a pattern.

To avoid over-counting of returned matches, choose a symmetric marker with the same symmetry as the pattern, as shown in Figure 2-17. For example, if a pattern is symmetric about the X-axis, choose a marker symmetric about the X-axis.

Figure 2-17. Marker Symmetry and Match Results



Region Layers

Region layers have special matching properties. A region layer is associated with a specific pattern layer and enables the definition of special areas within a pattern.

The region types are summarized in the following table.

Table 2-3. Region Types

Region Type	Summary Definition
Don’t Care Regions	An area that is ignored during the matching process.
Critical Regions	An area that requires a minimum of one polygon to exist within the region shape for a successful match during the matching process.
Keep Out Regions	An area that must not have any polygons inside it. Keep Out Regions are available for historical reasons, and are not necessary in the current pattern matching implementation.

A pattern can contain all three types of regions. Regions are layer specific, therefore multilayer patterns can specify Don't Care, Critical, and Keep Out regions for each pattern layer.

The following conditions apply to regions:

- Regions are only supported for TEM patterns.
- Regions can only contain Manhattan shapes. Regions may overlap vertices and edges, parts of polygons, or entire polygons of the layer geometries. They cannot have constraints directly applied to them.
- Regions outside the extent are checked.
- Regions that overlap or abut a pattern edge invalidate the use of that edge as an anchor. The edge is not guaranteed to exist, and the tool behavior is identical to that with a virtual edge.
- There is no precedence of regions. Each region requires that its criteria are met in order to match, regardless of what regions it overlaps with. See “Overlapping Regions” on page 54 for examples.
- Every polygon for a region type must meet its region condition for there to be a pattern match. If any region polygon fails its condition, there is no pattern match.

Note

Take care when adding regions so that a pattern continues to satisfy minimum pattern requirements. It is possible that a legal pattern without regions fails when regions are added. For example, covering all real edges in a pattern with a Don’t Care region causes the tool to generate an error.

- If matched markers are used and a region layer overlaps the pattern layer, the matched marker output is clipped by the region shape; that is the matched output is match_out = layout_layer NOT region_layer.

Region layers can be added with the following methods:

- GUI — "Adding Regions to a Pattern" on page 127
- Pattern capture from the GUI — "Capturing Patterns From a Layout" on page 118
- DFM Pattern Capture operation — Use the LAYER_REGION keyword.

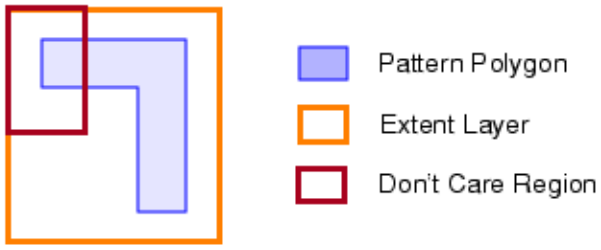
Don't Care Regions

Don't Care regions are unique layers that define a region that is ignored during the matching process. Don't Care regions are associated with a particular pattern layer.

Don't Care regions allow any or no polygons on the associated pattern layer to exist within the region. The Don't Care regions are not considered during the match process for the associated pattern layer.

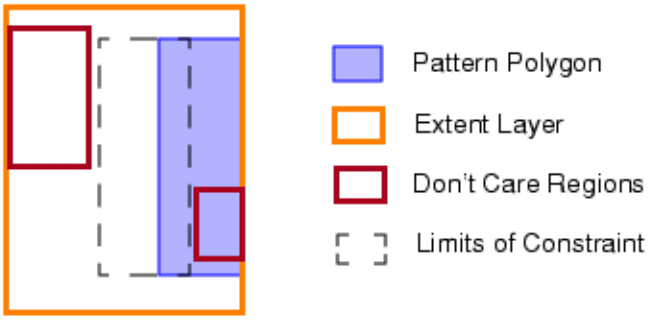
Figure 2-18 shows an example of a Don't Care region where geometries on the pattern layer in the upper left portion of the extent are ignored in matching pattern results:

Figure 2-18. Example of Don't Care Region



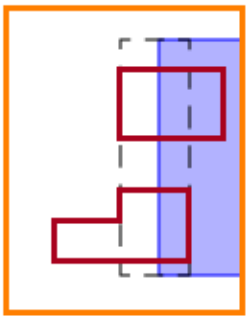
Regions completely outside of a constraint range do not affect the constraint, as in Figure 2-19.

Figure 2-19. Don't Care Region Without Overlap of Constraints



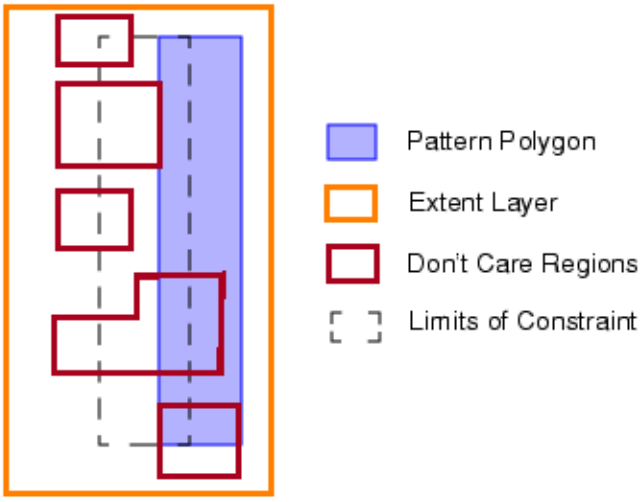
Regions that overlap a constraint range must consistently cover the same area across the entire edge movement area, as in Figure 2-20.

Figure 2-20. Don't Care Region Overlapping a Constraint



Regions that overlap a moving pattern edge (constraint range) but *do not* consistently cover the same area across the entire edge movement area are not supported, as in Figure 2-21.

Figure 2-21. Bad Don't Care Regions



If matched marker output is used, the matched output polygons are clipped by the region layer; that is the matched output is match_out = layout_layer NOT region_layer.

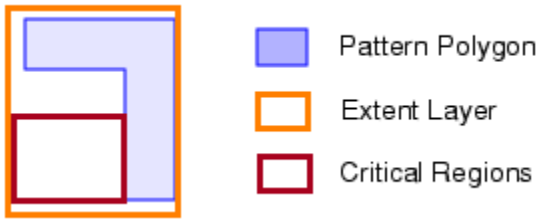
Critical Regions

Critical regions are unique layers that require a minimum of one polygon, of any shape, to exist within the region layer for a successful match during the matching process. Critical regions are associated with a particular pattern layer.

The pattern does not match if any Critical region area does not contain at least one layout shape on the associated layer. Regions can abut or overlap pattern layers. A layout shape that abuts a Critical region does not satisfy the critical region criteria. Pattern shapes within a Critical region that are on the associated pattern layer are ignored.

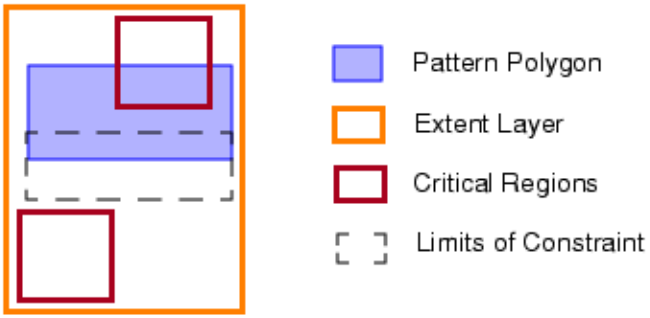
Figure 2-22 shows an example of a Critical region where a shape in the lower left part of the extent is required in matching pattern results:

Figure 2-22. Example of a Critical Region



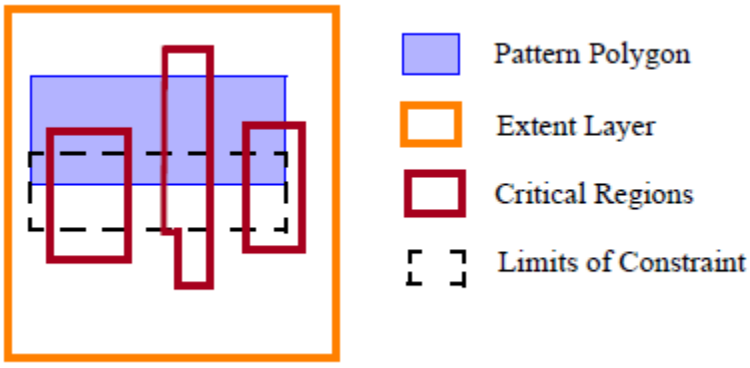
Critical regions completely outside of a constraint range do not affect the constraint, as in Figure 2-23.

Figure 2-23. Regions Outside of Constraint Range



Critical regions overlapping a moving pattern edge (constraint range) must consistently cover the same area across the entire edge movement area, as in Figure 2-24. Although they must completely cover the constraint region, the Critical region is always satisfied in the example below, because the Critical region overlaps the pattern shape.

Figure 2-24. Regions Overlapping a Constraint Range



If matched marker output is used, the matched output polygons are clipped by the region layer; that is, the matched output is match_out = layout_layer NOT region_layer.

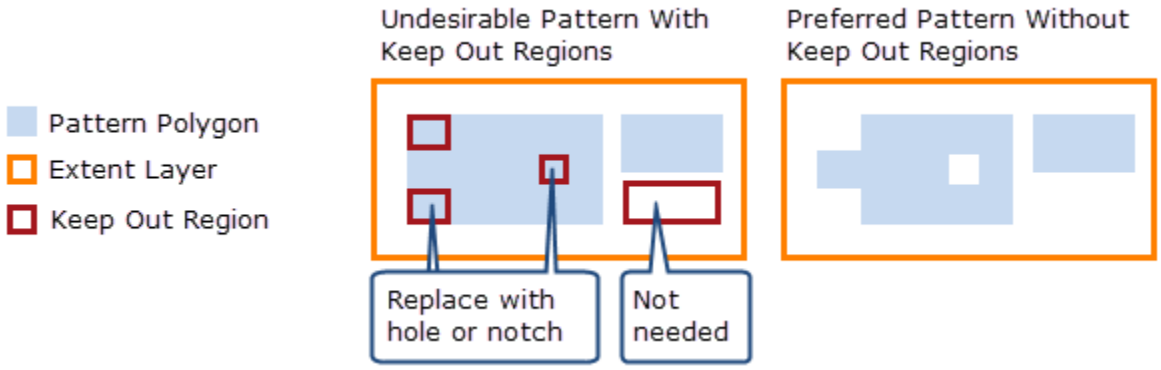
Keep Out Regions

A Keep Out region is an area that must not have any polygons inside it.

Keep Out regions are available for historical reasons, but are not necessary in the current pattern matching implementation. Previously, it was not possible to draw shapes with holes using the pattern matching GUI, and Keep Out regions were used to create a pattern shape with a hole.

Any empty region within a pattern extent must remain empty when matched, so a Keep Out region is not needed in a pattern area with no shapes. The following example shows how to create a pattern without Keep Out regions.

Figure 2-25. Keep Out Region



In general, more processing is required to handle regions, so patterns without regions have better performance.

As with all regions, Keep Out regions that overlap a moving pattern edge (constraint range) but do not consistently cover the same area across the entire edge movement area are not supported. See the figures with Don’t Care Regions and Critical Regions. Nested Keep Out regions are not supported.

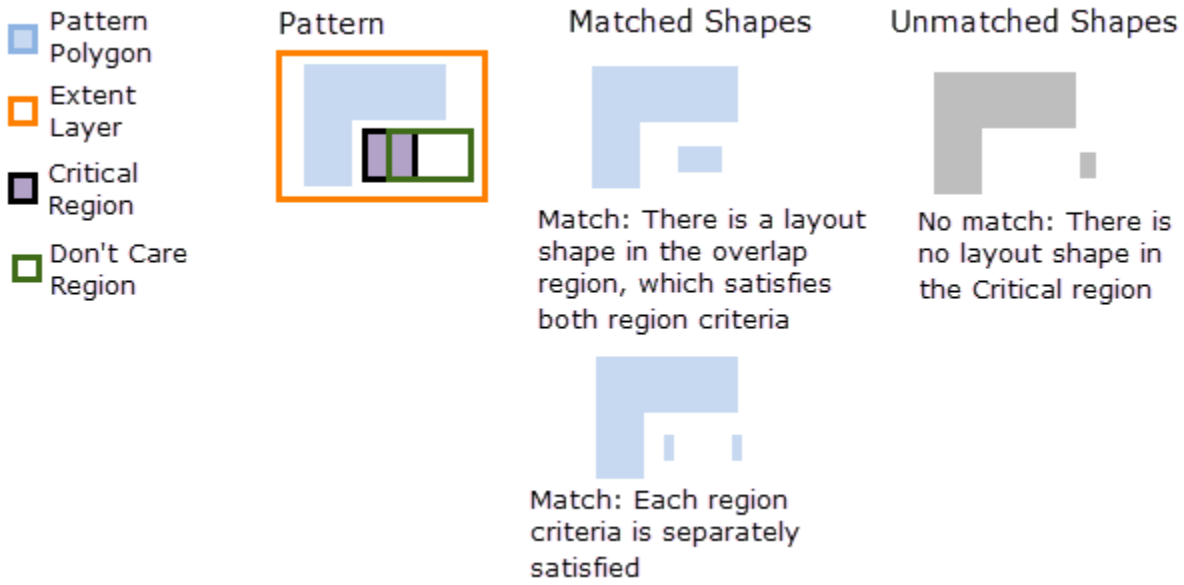
If matched marker output is used, the matched output polygons are clipped by the region layer; that is, the matched output is match_out = layout_layer NOT region_layer.

Overlapping Regions

When regions overlap, the criteria for each region must be met.

The following figure shows an example with overlapping Critical and Don’t Care regions.

Figure 2-26. Region Overlap



In effect, the overlap of a Critical and Don’t Care region functions as Critical region. If the overlap is part of a larger Critical region, the Critical region can be satisfied by a shape anywhere in the Critical region.

Keep Out regions are not recommended—see “Keep Out Regions” on page 54. However, if Keep Out Regions are used and overlap with a Don’t Care or Critical region, there can be no layout shape in the overlap region. This is because *both* region criteria must be satisfied. Critical and Keep Out regions cannot be completely coincident, because both conditions cannot be satisfied.

Don’t Care Pattern Layers

Geometries on Don’t Care pattern layers are ignored during the matching process. Don’t Care layers can be added to a library so that two or more libraries have the same number and order of layers, which is required by the merge utility.

Don’t Care pattern layers are specified with these methods:

Method	Summary
With the GUI	Select the layer in the **Layers** tab and click the **Set/unset Don’t Care button** ( ). See “Adding Pattern Layers to a Library” on page 128 for instructions on adding a Don’t Care layer to a library.
With DFM Pattern Capture	Specify the keyword DC with LAYER_TARGET to specify the layer as a Don’t Care pattern layer.

Method	Summary
With pdl_lib_mgr convert	Use the option add_layer to add a Don't Care or empty pattern layer to a library.

Note

Don't Care pattern layers are *not* the same as Don't Care regions. See "Don't Care Regions" on page 51.

Full and Empty Pattern Layers

Multilayer patterns can include full and empty pattern layers, but at least one layer that is not full, empty, or a Don't Care layer is required.

- **Full layer** — The layer has one pattern geometry that coincides with the pattern extent.
- **Empty layer** — The layer has no drawn shapes.

Constraints

Constraints enable you to specify the distance an edge can shift and still be considered a match. Constraints allow for variations in the pattern when Calibre searches for pattern matches. Constraints are specified in the Calibre Pattern Matching GUI.

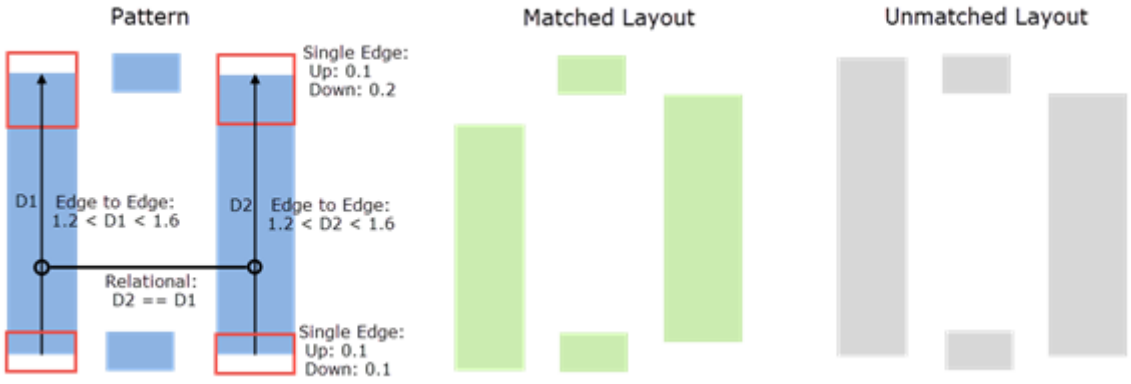
These are the possible constraint types and the pattern types they can be used in:

Table 2-4. Summary of Constraint Types

Constraint	Valid Pattern Types	Summary
Global (cglobal)	TEM Single-layer BCM	Specifies the allowed movement for all edges. For TEM patterns, a cglobal constraint can be applied per layer.
Single Edge	TEM BCM2 (fixed extent)	Specifies the allowed movement of one edge.
Edge to Edge	TEM BCM1 and BCM2	Specifies the allowed separation between two parallel edges.
Relational	TEM BCM1 and BCM2	Specifies a relation (such as equality) between the separation distances of two edge to edge constraints.

Constraints cannot be specified for angled edges or edges adjacent to an angled edge. Constraints cannot be specified in crossbar (XBAR) patterns.

The following figure illustrates single edge, edge to edge, and relational constraints for a TEM pattern. The top and bottom edges of the two large polygon are allowed to move due to single edge constraints. Two edge to edge constraints specify the allowed length (D1 and D2) of the polygons. A relational constraint states that the lengths D1 and D2 must be equal. A matched layout and an unmatched layout are shown in the figure.



Global Constraints

A global constraint specifies an allowed movement for all real edges in a pattern. Global constraints can be applied to TEM patterns and single-layer BCM patterns. For TEM patterns, you can apply a global constraint to the whole pattern or to selected pattern layers.

The global edge constraint is applied to all edges not coincident with the extent. Edges coincident with the default extent are virtual edges and may not be constrained the same way as real edges. A global constraint value (cglobal) should be no greater than 1/4 of the minimum feature size (width or spacing) in the pattern.

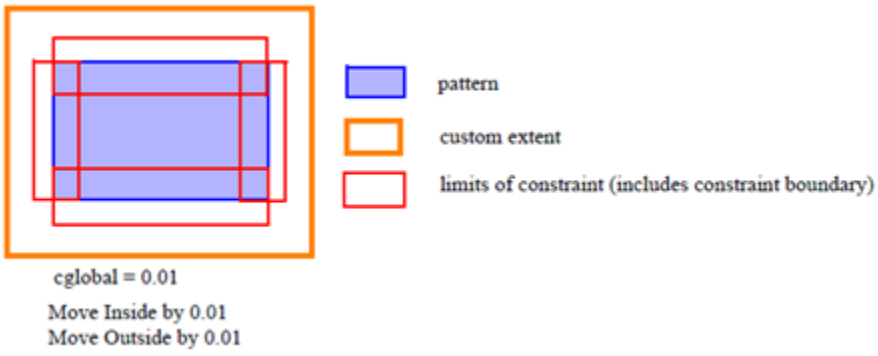
A pattern with overlapping real vertices cannot have a global constraint. See “Vertices in Patterns” on page 25.

For TEM Patterns, you can apply other edge constraints to a pattern with a cglobal constraint—the other edge constraints override the cglobal value.

See “Adding Global Constraints (cglobal)” on page 136 for information using the Calibre Pattern Matching GUI to add a cglobal constraint.

The following figure shows a cglobal constraint for a pattern with one layer.

Figure 2-27. Cglobal Constraint



Related Topics

Adding Global Constraints (cglobal)

Edge to Edge Constraints in TEM Patterns

Edge to edge constraints specify the allowed separation between two parallel edges. Edge to edge constraints can be specified between a pattern edge and another pattern edge, between a pattern edge and a custom extent edge, or between a pattern edge and a custom marker edge.

Note

By default, edge to edge constraints measure separation from the first (or reference) edge to the second edge and take direction into account. Default (or directional) TEM edge to edge constraints are discussed here.

Non-directional edge to edge constraints are possible in TEM patterns and are useful in certain cases. See “Non-Directional Edge to Edge Constraints in TEM Patterns” on page 61.

These conventions are used in the figures:

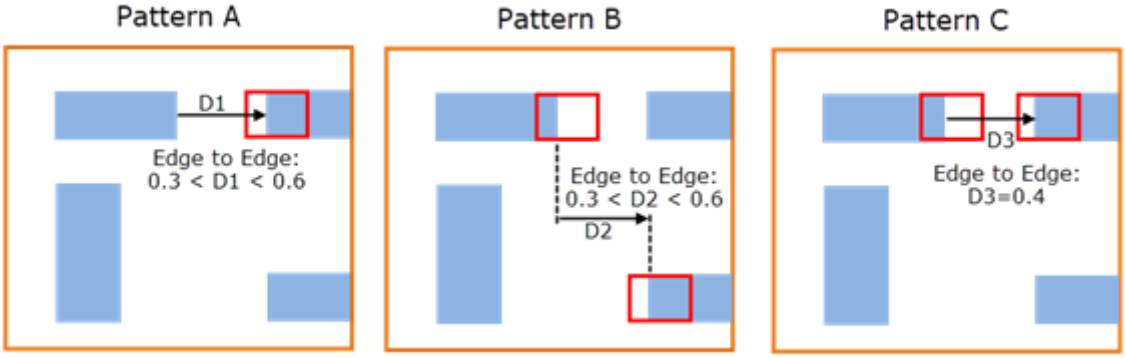
The separation between two parallel edges is measured as the minimum separation distance between the edges. If the parallel edges do not have a common projection, the minimum separation between the extension of the edges is measured, as shown for Pattern B in Figure 2-28.

If an edge to edge constraint is defined between two fixed pattern edges (edges without constraints), the edge to edge constraint can be used in forming a relational constraint, but it does not provide any other functional purpose.

Pattern Edge to Pattern Edge

The following figure shows three different edge to edge constraints between pattern edges.

Figure 2-28. TEM Edge to Edge Constraints Between Pattern Edges



Pattern A: There is only one single edge constraint. The edge with the single edge constraint can move as allowed by its constraint. The edge to edge constraint specifies a constraint on the distance D1.

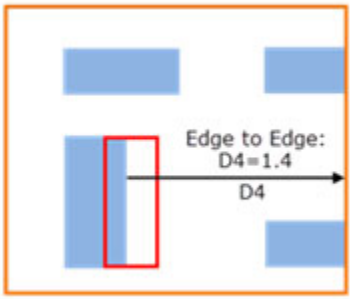
Pattern B: There are two single edge constraints. The edge to edge constraint specifies a range for the distance D2 between the two edges.

Pattern C: There are two single edge constraints. The edge to edge constraint specifies a fixed distance D3 between the two edges.

Pattern Edge to Custom Extent

Edge to edge constraints can be specified between a pattern edge and an edge of a custom extent. The pattern edge must have a single edge constraint. The edge to edge constraint must be an equality (==) constraint, specifying that the custom extent edge maintains a fixed distance from the pattern edge. See “Dynamic Extents” on page 39 for more information.

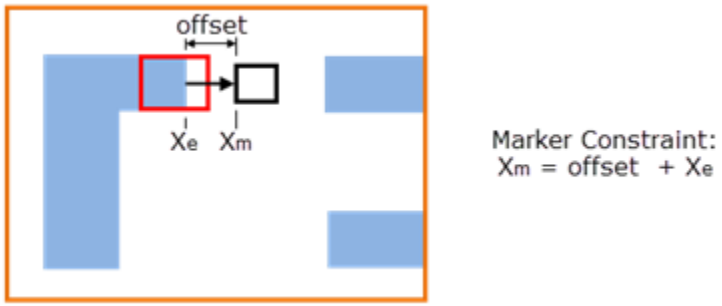
Figure 2-29. TEM Edge to Edge Constraint to a Custom Extent Edge



Pattern Edge to Custom Marker

Edge to edge constraints can be specified between a custom marker edge and one or two pattern edges. The pattern edge(s) must have a single edge constraint. The following figure shows a marker constraint to one pattern edge.

Figure 2-30. TEM Edge to Edge Constraint to a Custom Marker Edge



See "Marker Constraints" on page 64 for more information and examples.

Related Topics

Adding a TEM Edge to Edge Constraint

Non-Directional Edge to Edge Constraints in TEM Patterns

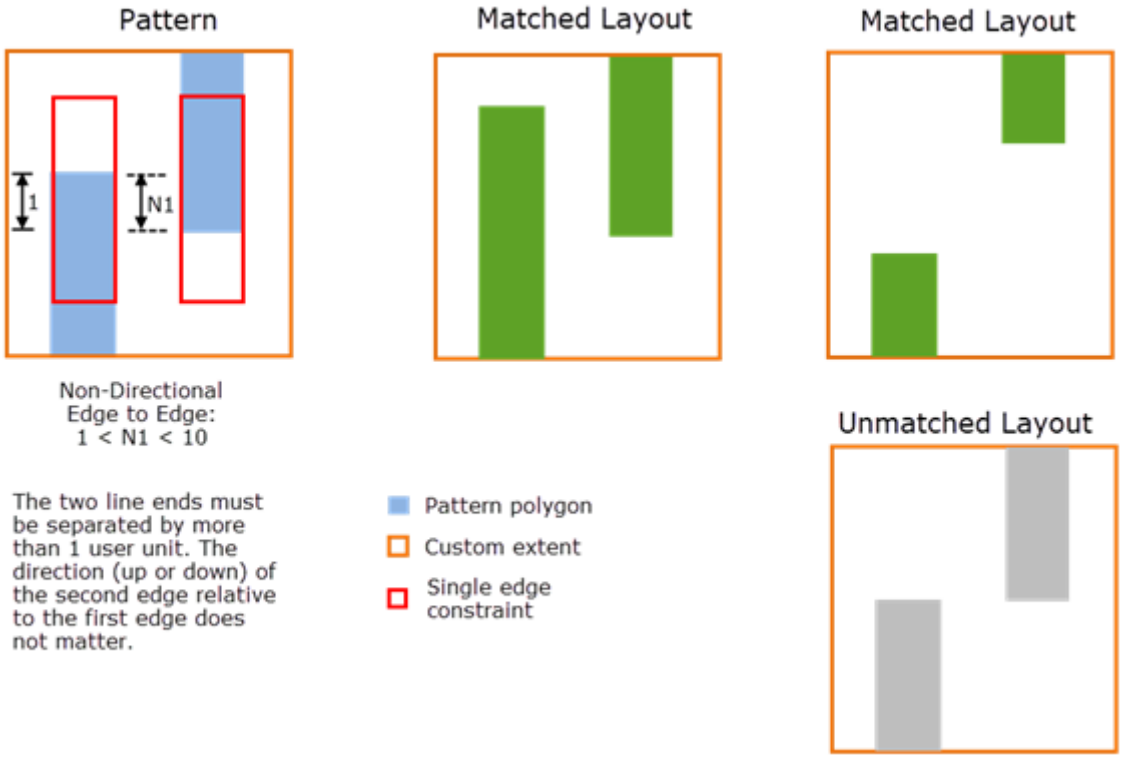
Non-directional edge to edge constraints are useful in patterns in which the relative position (left/right or up/down) of two moving pattern edges does not matter, but there is a constraint on the absolute value of the edge separation.

Non-directional edge to edge constraints are only valid in TEM patterns. Both edges must have single edge constraints.

Most constraints can be defined without the use of a non-directional constraint. A non-directional constraint should only be used when necessary, as they generally provide poorer performance than the standard directional constraint.

The following figure illustrates an edge to edge constraint that must be specified as a non-directional. It is not possible with a directional edge to edge constraint to exclude separations less than 1 and still match both of the matched configurations shown in the figure.

Figure 2-31. Non-Directional Edge to Edge Constraint



Related Topics

Adding a TEM Edge to Edge Constraint

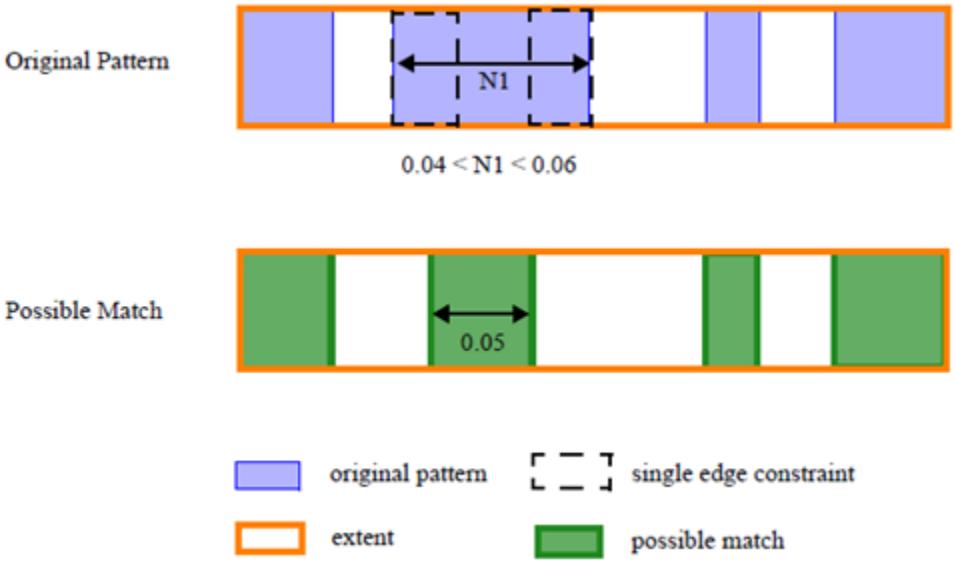
Edge to Edge Constraints in Fixed Extent BCM Patterns (BCM2)

Edge to edge constraints in BCM2 patterns specify the allowed separation of two edges, where at least one of the edges must have a single edge constraint. The pattern extent remains fixed as the edges move.

The allowed movement for an edge is specified with the single edge constraint, while the edge to edge constraint places a constraint on the distance between two edges. Edge to edge constraints cannot be specified between a pattern edge and the pattern extent. Virtual edges at the pattern extent cannot have constraints.

The following image shows a BMC2 pattern with two single edge constraints. An edge to edge constraint N1 is specified between the two edges that can move. Without the edge to edge constraint, the two single edge constraints allow a minimum wire width of 0.02. The edge to edge constraint specifies a minimum edge separation of 0.04, and a maximum separation of 0.06.

Figure 2-32. BCM2 Fixed Extent Pattern With Edge to Edge Constraint



Related Topics

Adding a BCM2 (Fixed Extent) Edge to Edge Constraint

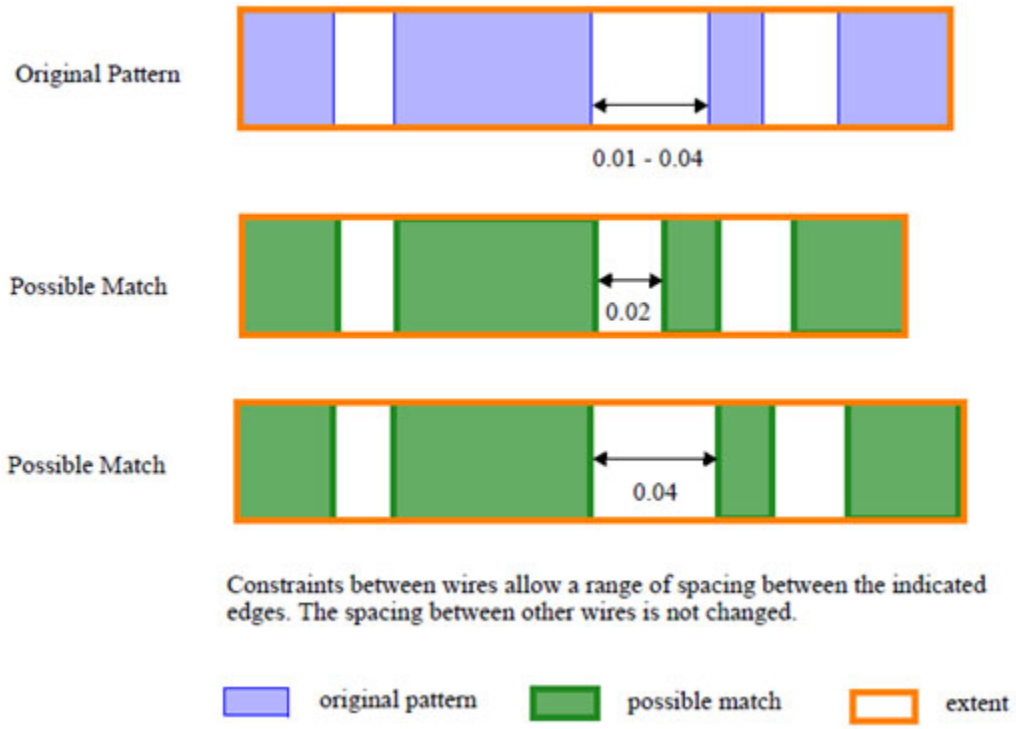
Edge to Edge Constraints in Accordion Extent BCM Patterns (BCM1)

Edge to edge constraints in BCM1 patterns specify the allowed separation of two edges. The pattern extent shrinks or expands as the edges move.

Single edge constraints are not used in BCM1 patterns. The allowed movement of edges is specified with edge to edge constraints. Edge to edge constraints can be specified between any two pattern edges. Edge to edge constraints cannot be specified between a pattern edge and the pattern extent. Virtual edges at the pattern extent cannot have constraints.

The following image shows a BCM1 pattern with an edge to edge constraint specifying a range of allowed spacings between two pattern edges. The pattern extent changes as the spacing changes.

Figure 2-33. BCM1 Accordion Extent Pattern With Edge to Edge Constraint



Related Topics

Adding a BCM1 (Accordion Extent) Edge to Edge Constraint

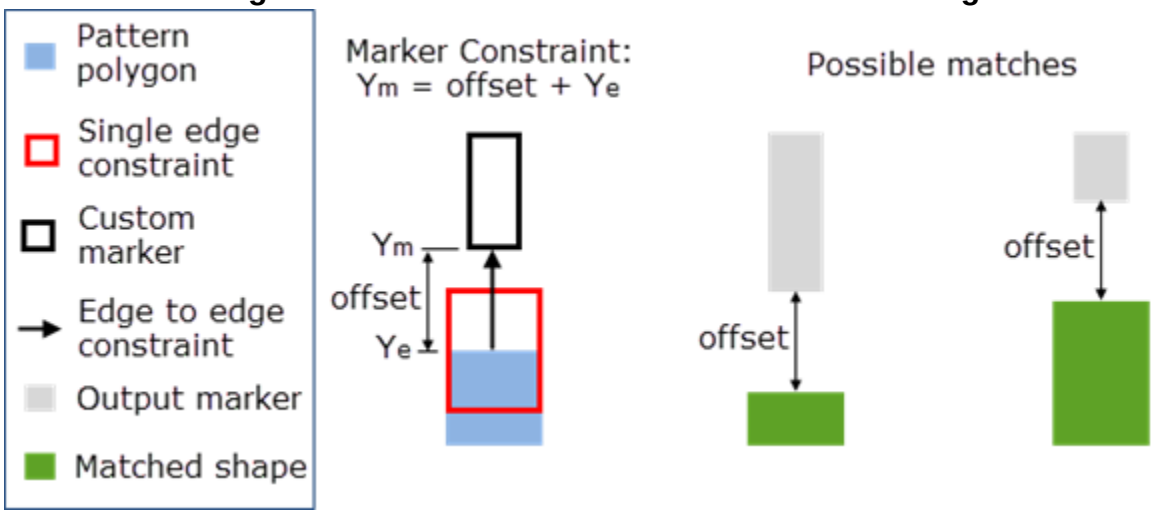
Adding Constraints to a Multilayer BCM1 (Accordion Extent) Pattern

Marker Constraints

Marker constraints specify the location of a custom marker edge in relation to a pattern edge that can move. Marker constraints are only allowed in TEM patterns. They are specified as an edge to edge constraint from a custom marker edge to one or two constrained pattern edges.

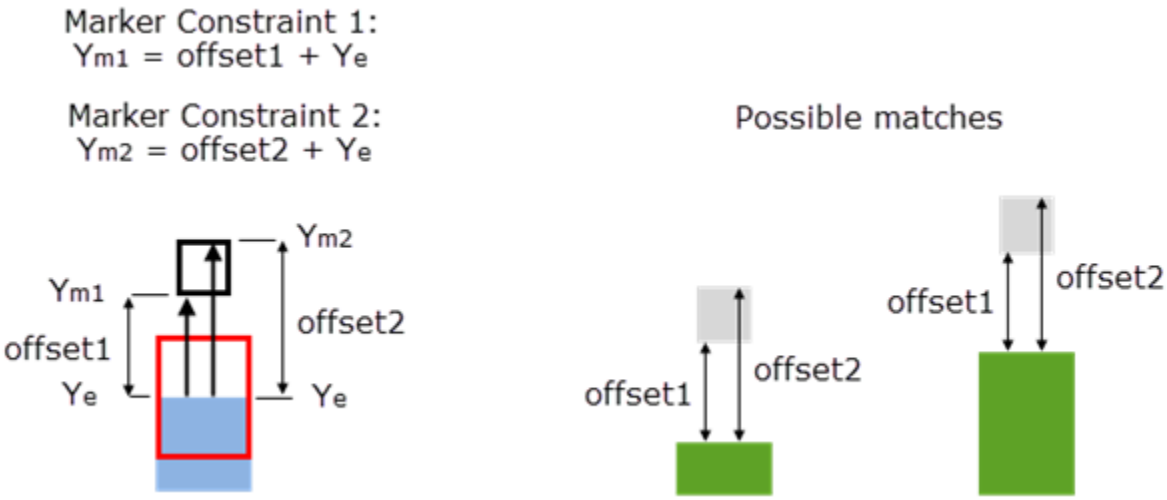
The following image shows a pattern shape with a single edge constraint and a custom marker shape. An edge to edge constraint specifies a fixed distance between the top pattern edge (which can move) and the lower marker edge. The top marker edge remains fixed.

Figure 2-34. Marker Constraint to One Pattern Edge



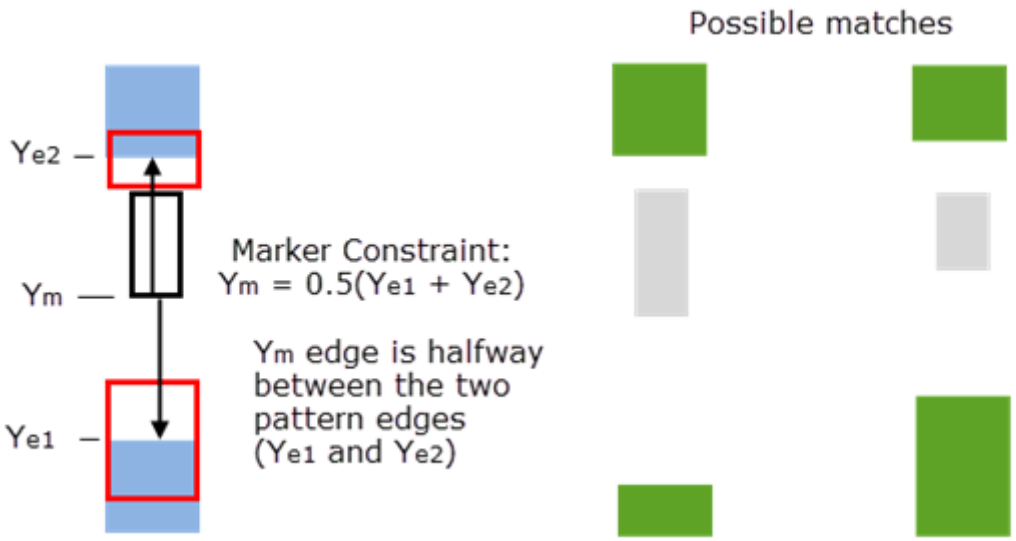
The following image has an additional marker constraint specified to the top customer marker edge, so that the marker width in the Y-direction remains constant.

Figure 2-35. Two Marker Constraints to Define Fixed Width Marker



You can also specify the relative position of a marker edge between two pattern edges that can move. For example, you can specify that the marker edge be halfway between two pattern edges that move. This is shown in the following figure.

Figure 2-36. Marker Constraint to Two Pattern Edges



The above example shows the marker constraint equation for a marker edge centered between two pattern edges. The more general equation is the following:

$$Y_m = \text{offset} + b_1 * Y_{e1} + b_2 * Y_{e2}$$

where Y_m is the position of the marker edge, offset is a fixed offset, b_1 and b_2 are coefficients, and Y_{e1} and Y_{e2} are the pattern edge positions. When creating marker constraints to two pattern edges, the GUI constructs a marker constraint equation that satisfies the initial edge positions. You can edit the two coefficients (b_1 and b_2).

Related Topics

Adding a Marker Constraint

Relational Constraints

Relational constraints specify a relation between two edge to edge measurements. For example, two polygons may have variable widths, but require the same width for a successful pattern match.

At least two edge to edge constraints must exist in the pattern in order to create a relational constraint. See these topics:

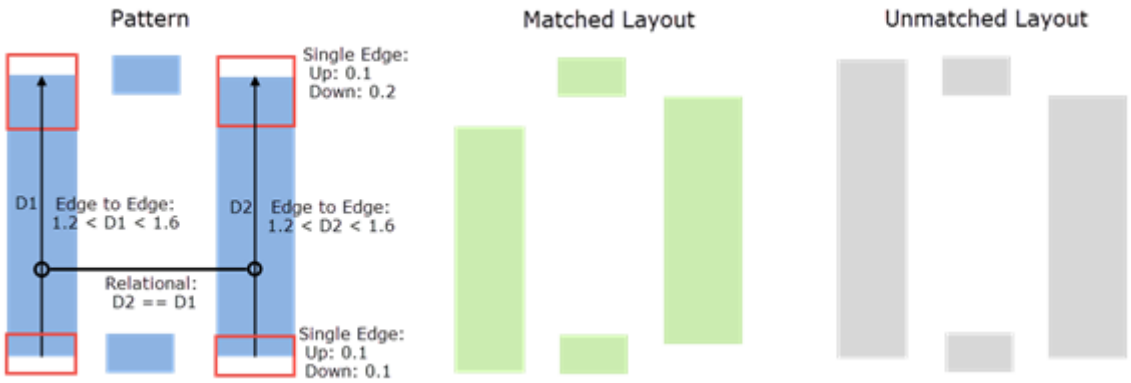
- “Edge to Edge Constraints in TEM Patterns” on page 59
- “Non-Directional Edge to Edge Constraints in TEM Patterns” on page 61
- “Edge to Edge Constraints in Accordion Extent BCM Patterns (BCM1)” on page 63
- “Edge to Edge Constraints in Fixed Extent BCM Patterns (BCM2)” on page 62

Relational constraints support the following comparisons:

==	Equal	!=	Not equal
<	Less than	>	Greater than
<=	Less than or equal	>=	Greater than or equal

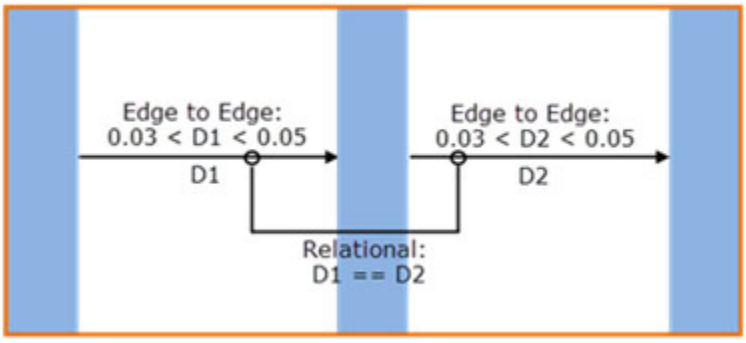
The following figure illustrates single edge, edge to edge, and relational constraints for a TEM pattern. The top and bottom edges of the two long polygons are allowed to move due to single edge constraints. Two edge to edge constraints specify the allowed length (D1 and D2) of the polygons. A relational constraint states that the lengths D1 and D2 must be equal. A matched layout and an unmatched layout are shown in the figure.

Figure 2-37. TEM Relational Constraint



In the following BCM1 pattern, the two edge to edge constraints specify the allowed spacing between wires. The relational constraint specifies that the spacing must be equal.

Figure 2-38. BCM Relational Constraint



Relational constraints take this form:

(+/-)*separation_1 operator* (+/-)*separation_2*

- *separation_n* — The separation of the two edges in the edge to edge constraint, as measured in the matched layout, measured from the first edge to the second edge. When specifying a relational constraint in the GUI, you can specify "-" to use the negation of the measured separation.

  In Figure 2-38, D1 and D2 correspond to *separation_1* and *separation_2*.

- *operator* — One of the relational operators. (==, !=, <, <=, >, >=)

The measured separation can be positive or negative, depending on the relative placement of the first and second edges. Care should be taking when creating relational constraints using non-directional edge to edge constraints, because a non-directional edge to edge constraint can match to a positive or negative separation.

Tip

To determine the first and second edge of an edge to edge constraint, hide the single edge constraints in the GUI and view the connector of the edge to edge constraint. The first edge has a diamond-shaped connected. See Figure 3-6 on page 144 in the topic "Adding a TEM Edge to Edge Constraint."

Related Topics

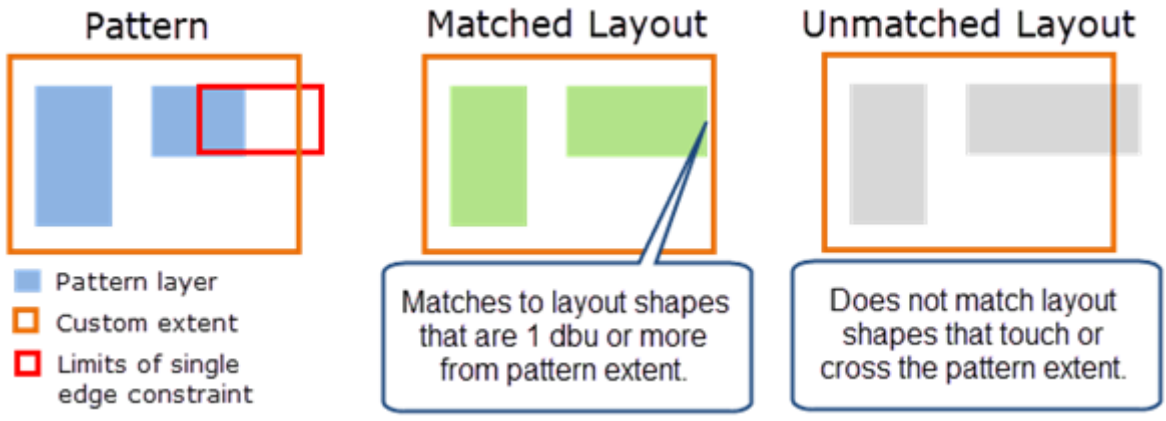
Adding a Relational Constraint

Behavior of Constrained Edges at the Pattern Extent

Pattern constraints that extend past the pattern extent can cause matches to be missed.

It is possible to define a constraint that results in a polygon extending to or past the pattern extent, as in the following figure. Constraints of this type only match up to 1 dbu inside of the extent, and the tool issues a warning that matches touching or crossing the extent will be missed.

Figure 2-39. Single Edge Constraint Outside Extent Boundary



Pattern Information: Attributes, Properties, Keys, Comments, and Orientation

An attribute can be given to a pattern or group of patterns prior to using the pattern matching tool. A key is an alphanumeric string attribute that can be used to filter out patterns from the pattern library. Patterns can also be filtered by pattern orientation. Pattern comments can be helpful in providing information to other users about a particular pattern.

Properties

Properties can be assigned to patterns to enhance the output capabilities of Calibre Pattern Matching.

The list of properties is defined at the library level of a pattern library. There are two types of properties:

Table 2-5. Property Types

Type	Definition
Custom property	The property value is assigned per pattern, but the property type is fixed for each property. The property type can be floating point or integer, enabling storage of 32-bit floating point numbers (up to 16,777,216) or 32-bit signed-integer values (up to 2,147,483,647). Property names can contain only alphanumeric characters (a-z, A-Z, and 0-9) and underscores (_). They cannot contain spaces.
Output property	Output properties are added at the library level. The property value is assigned for the matched result during a pattern matching run and reports a property describing how the match was made. The output properties are match_orient, match_rotation, and match_type. See “Reserved Property Names” on page 71.

During a pattern matching run, Calibre attaches all properties for a matched pattern to the output marker shapes for the match by default. The properties are attached as DFM properties, which can be read and manipulated by operations such as DFM Property. The output layer from a pattern matching run is often saved to a results database with the DFM RDB operation. You can use the CMACRO option -property to limit the properties that are attached to the output marker shapes.

If output marker shapes from a pattern matching run overlap, they are merged. If the value for an attached property is not the same on the merged shapes, the largest property value is used. You can use count markers to avoid merging; see “Count Markers” on page 45.

If a pattern has a defined property but no value is assigned, a value of -1 is assigned to the property on the output marker shape for the pattern match. This default value can be changed with the CMACRO option -invalid_prop_value; see “CMACRO for Calibre Pattern Matching” on page 177.

Properties can be added to a library with these methods:

- **With the GUI** — See “Adding Custom and Output Properties and to a Library” on page 158. You can add both custom properties and output properties.
- **During a capture operation** — For custom properties, see the PATTERN_PROP keyword for DFM Pattern Capture when running batch Calibre. For output properties, see the MATCH_ORIENT, MATCH_ROTATION, and MATCH_TYPE keywords.

Also see "Capturing Patterns From a Layout" on page 118 when capturing from an open layout.

- **With a Tcl API command** — See pmatch::add_properties in the *Calibre Pattern Matching Reference Manual*.

Reserved Property Names

Reserved property names used internally by the Calibre Pattern Matching tool. The property value is assigned during a pattern matching run.

The following table summarizes required input and possible return values for each property.

Table 2-6. Special Property Names

Property Name	Returned Value
match_orient	0-7
match_rotation	0,1,2, or 3
match_type	0, 1, or 2
orient_detail	reserved
pattern_orient	reserved

Note

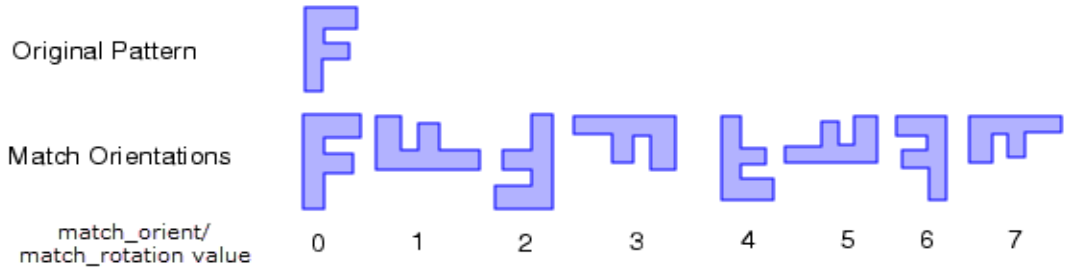
For match_orient, match_type, and match_rotation *no* input values are set. These properties are enabled in the Library Attributes window instead of the Properties pane for patterns. These properties are also referred to as output properties. See "Adding Custom and Output Properties and to a Library" on page 158.

The use of match_orient and match_rotation can cause slower performance and increased flattening of the results when the pattern library is used in a pattern matching run.

The Returned Value is assigned during a pattern matching run. The property is attached to the output marker shapes. The properties are defined as follows:

- **match_orient** — A number that identifies how the match result is oriented in the layout. The possible returned values are from zero to seven. Figure 2-40 shows the match orientation for each value. The output marker for each match result is returned in the same orientation as the match.
- **match_rotation** — A number that identifies how the match result is rotated in the layout. The possible returned values are from zero to three. Figure 2-40 shows the match rotation for each value. The output marker for each match result is returned in the same rotation as the match. This property is mutually exclusive with match_orient property.

Figure 2-40. match_orient and match_rotation Values



Note

A pattern can be drawn with any number of angled edges, however the placement of that pattern in the design must be at a 0, 90, 180, or 270 degree rotation or mirroring.

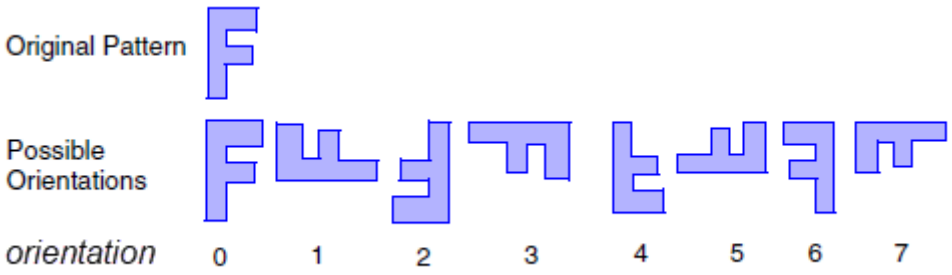
- **match_type** — An integer that identifies whether constraints were used when matching the layout location. The allowed values and meaning are described in the following table.

Table 2-7. Returned match_type Values

Returned match_type Value	Description
0	The match was to a pattern with no constraints.
1	The match was to a pattern with constraints and one or more constraints were used when making the match.
2	The match was to a pattern with constraints but none of the constraints were needed when making the match.

The match_type property only applies to TEM patterns. BCM patterns do not support this property.

- **orient_detail** — A bit value that gives the allowed orientations of a match to the pattern. The value contains the setting (0 or 1) for the eight possible pattern orientations. A pattern that can match in all orientations has a value of 11111111. A pattern that can match only in orientation 0 (as drawn) has a value of 10000000.



Attributes

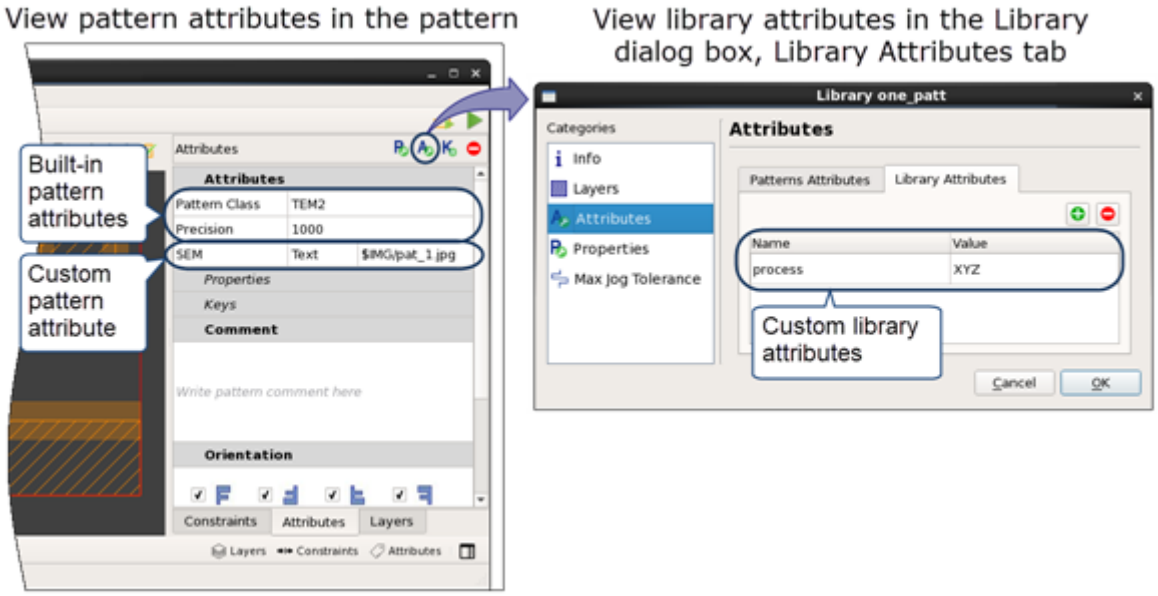
An attribute is string or numeric data associated with the pattern library. Attributes are not used in a pattern matching run and are not attached to pattern matching results.

Pattern libraries may have these types of attributes:

Table 2-8. Attribute Types

Type	Definition
Built-in pattern attribute	A tool-defined attribute that provides information about the pattern, such as the pattern type (TEM, BCM, XBAR) and precision.
Custom pattern attribute	A user-defined attribute with a value that is assigned per pattern.
Custom library attribute	A user-defined attribute with a value that is assigned for the library.

Figure 2-41. Viewing Attributes



Attributes are added with these methods:

- **In the GUI** — See “Adding Custom Attributes to a Library” on page 161 and “Defining Custom Attribute Values in a Pattern” on page 163.
- **During a capture operation** — See the PATTERN_ATTR keyword for DFM Pattern Capture when running batch Calibre.
- **With a Tcl API command** — See pmatch::add_int_attr, pmatch::add_text_attribute, and pmatch::set_library_attribute in the *Calibre Pattern Matching Reference Manual*.

Pattern Keys

Keys are optional pattern attributes that allow you to select a subset of patterns to pass to the pattern matching tool for matching.

If the pattern matching CMACRO command specifies the -keys option, then only patterns that have all the specified keys are used in the pattern matching run.

Suppose a pattern library has three patterns with these keys defined:

- Pat1 key = cat
- Pat2 key = dog
- Pat3 key = cat, dog

The following patterns are selected at runtime, depending on the CMACRO option:

CMACRO option	Selected Patterns
-keys cat	Pat1 and Pat3
-keys dog	Pat2 and Pat3
-keys {cat dog}	Pat3

Key names can contain only alphanumeric characters (a-z, A-Z, and 0-9) and underscores (_). They cannot contain spaces.

Keys are defined in a pattern in these ways:

- **In the GUI** — See “Attaching Keys to a Pattern” on page 165.
- **During a capture operation** — See the PATTERN_KEY keyword for DFM Pattern Capture when running batch Calibre. Also see “Capturing Patterns From a Layout” on page 118 when capturing from an open layout.
- **With a Tcl API command** — See pmatch::add_keys in the *Calibre Pattern Matching Reference Manual*.

Pattern Comments

Pattern comments are optional pattern attributes that allow you to add and save text along with your pattern.

Pattern comments are useful for passing information about a pattern to other pattern users. A pattern comment can be made up of any keyboard characters.

Comments are only used in the Calibre Pattern Matching GUI; they are not used during pattern matching. See “Adding Comments to a Pattern” on page 167.

When pattern libraries are merged, if duplicate patterns have different comments, the comments are concatenated in the pattern saved to the merged library.

Related Topics

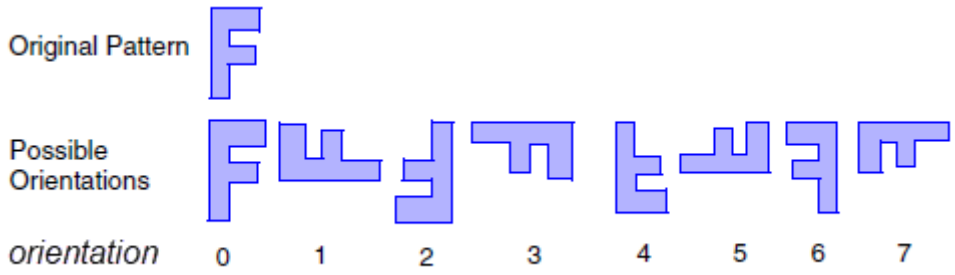
Adding Comments to a Pattern

Pattern Orientation

Pattern orientation settings are optional pattern attributes that define the allowed orientations of a pattern match.

The following figure illustrates the eight possible orientations for a pattern:

Figure 2-42. Orientations



You can specify pattern orientation with these methods:

- In the GUI — "Defining Orientation for a Pattern" on page 166
- With the DFM Pattern Capture operation — See the PATTERN_ORIENT keyword
- With Tcl API commands — pmatch::set_pattern_orient and pmatch::add_pattern_orient

Keep in mind the following as you use orientation-aware pattern capture:

- Orientation-aware pattern capture and matching operations may increase run times. This is indicated by the tool through warning messages.
- The accuracy of orientation-aware pattern capture and matching operations may be impacted if you specify layout cloning statements in your SVRF rule file.

Caution

If you use orientation-aware pattern matching, do not specify Layout Clone Transformed Placements NO or Layout Clone Rotated Placements NO in your SVRF rule files. Doing so invalidates the orientation results.

Also, be aware that certain operations that work on cell names may not work on their cloned counterparts.

Related Topics

Defining Orientation for a Pattern

Jog Tolerance in Pattern Matching

You can specify a jog tolerance for all patterns in a library. Jogs within the specified tolerance are ignored in both the pattern and the layout.

Jog tolerance is controlled by the parameter maxjog. A jog or series of jogs is ignored if the total jog displacement is less than maxjog, as shown in the following figure:

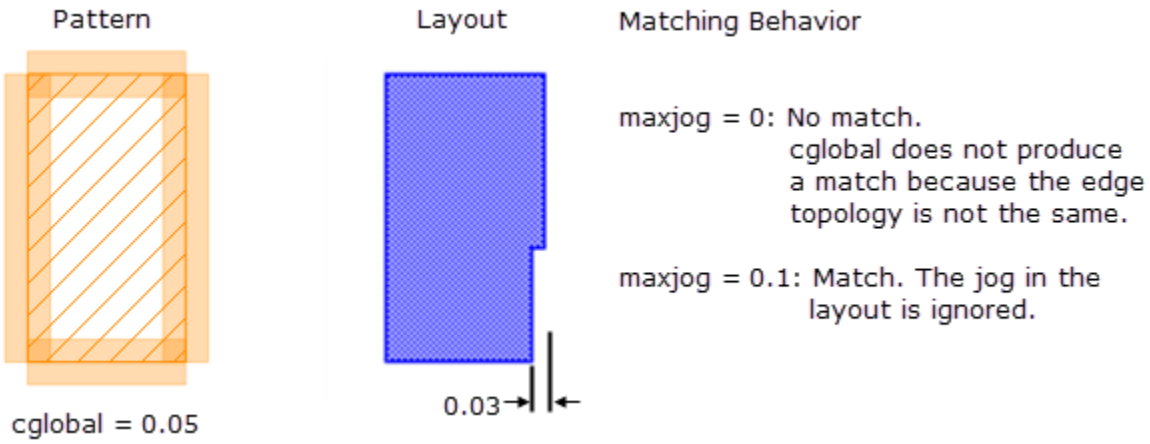
You can specify the maxjog parameter with these methods:

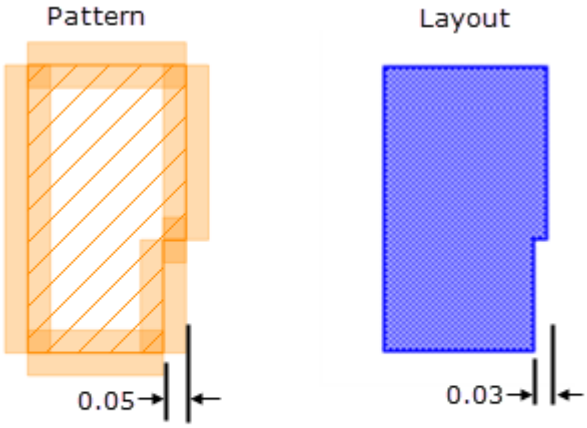
- In the Calibre Pattern Matching GUI as a library attribute. You can also disable jog tolerance for individual edges. See “Specifying Jog Tolerance for Patterns in a Library” on page 96.
- With the maxjog compile option. See “compile” on page 226.
- With the CMACRO runtime option -maxjog. See “CMACRO for Calibre Pattern Matching” on page 177.

The jog tolerance value (maxjog) must be less than or equal to twice the global constraint value:

```
maxjog <= 2*cglobal
```

The following series of figures provide some examples of jog handling.



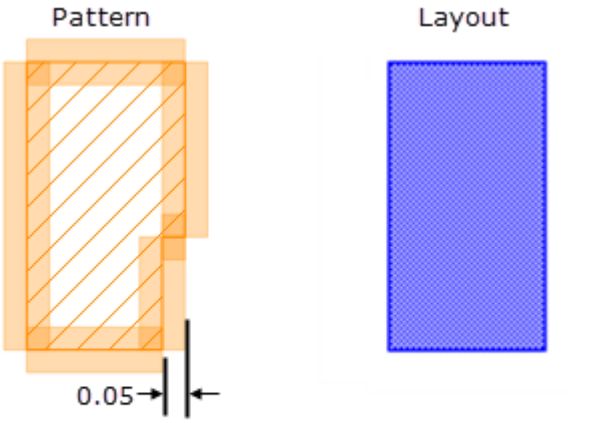


cglobal = 0.05
Pattern edge has a jog of 0.05

Matching Behavior

maxjog = 0: Match.
cglobal results in a match because the edge topology is the same.

maxjog = 0.1: Match.

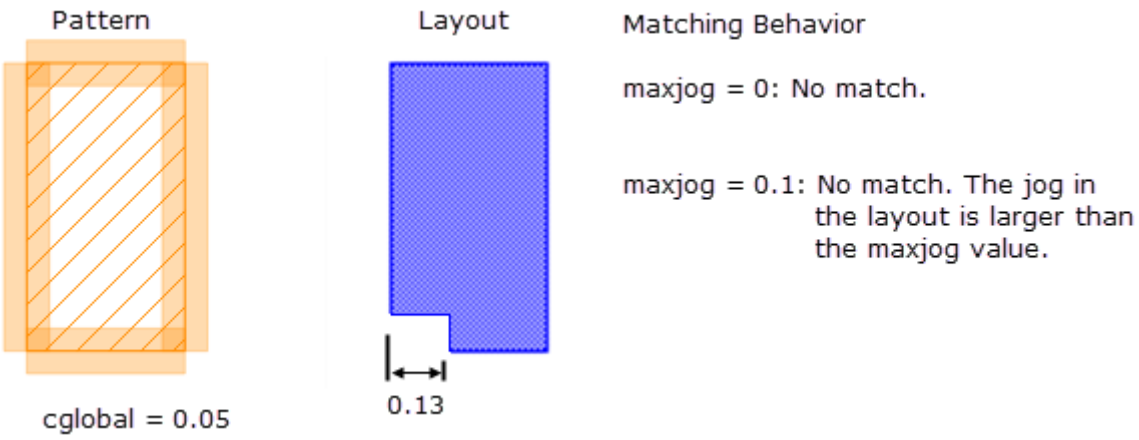


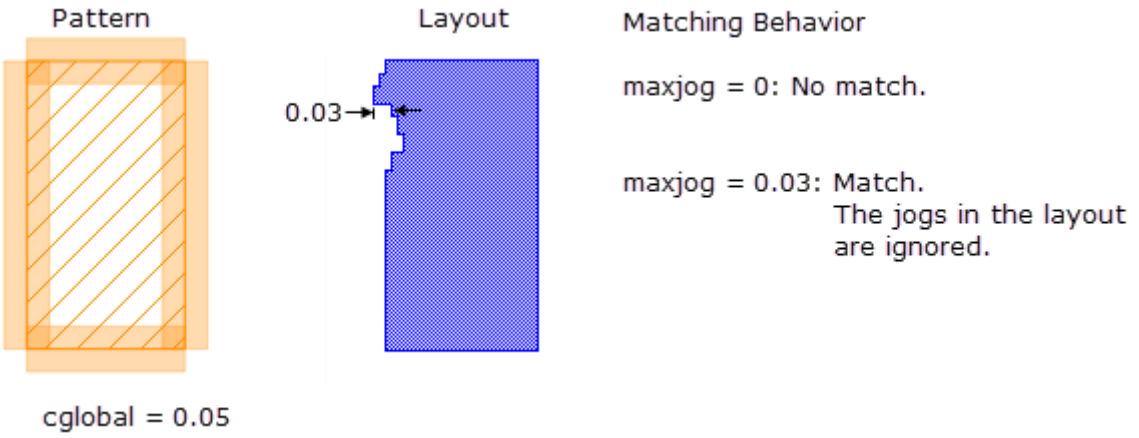
cglobal = 0.05
Pattern edge has a jog of 0.05

Matching Behavior

maxjog = 0: No match.
cglobal does not produce a match because edge topology is not the same.

maxjog = 0.1: Match. The jog in the pattern is ignored.





Precision Handling in Pattern Matching

Most precision differences between a pattern, the layout being matched, and the rule file are handled automatically during a pattern matching run. If the precisions differ, the pattern coordinates are automatically adjusted so that the matching behavior is correct.

In some cases, however, adjusting the pattern coordinates results in truncation of the pattern geometry. In particular, this can happen when the layout precision is lower than the pattern precision. If truncation of a pattern occurs due to precision handling during a pattern matching run, an error is issued for the affected pattern and the pattern is ignored.

DMACRO and CMACRO for Pattern Matching

A DMACRO file defines the patterns in a pattern library in a format that can be included in the SVRF rule file. The CMACRO command runs pattern matching on the DMACRO library and generates output layers corresponding to the output markers in the pattern library.

Once you have a pattern library, it is compiled to create a DMACRO. The compile step can be done with two different methods:

- From the Calibre Pattern Matching GUI — see "Compiling a Pattern Library with the Calibre Pattern Matching GUI" on page 105
- From the command line — see the compile command for the pdl_lib_mgr utility

The compiled DMACRO pattern library is specified in the rule file with an Include statement.

The CMACRO command runs pattern matching on the compiled DMACRO pattern library. The CMACRO command can include various options that control the pattern matching process. See "CMACRO for Calibre Pattern Matching" on page 177.

The following topic demonstrates how to use the DMACRO library and the CMACRO command in a rule file for a pattern matching run:

- "Using a DMACRO in a Rule File for a Pattern Matching Run" on page 185

Related Topics

Pattern Libraries

Chapter 3
Using the Calibre Pattern Matching GUI

The Calibre Pattern Matching GUI provides a way to build and edit a pattern library for a Calibre Pattern Matching run. You can also run certain library utilities, such as merge and compile, from the GUI.

Calibre Pattern Matching GUI Basics

The Calibre Pattern Matching GUI provides an interface for viewing and customizing pattern libraries. This section discusses invocation, basic setup, and GUI components.

Invoking the Calibre Pattern Matching GUI

You invoke the Calibre Pattern Matching GUI from the command line or from a Calibre layout viewer.

Prerequisites

- The requirements discussed in "Calibre Pattern Matching Product Requirements" on page 16 are met.

  In particular, make sure that a writable temporary directory is available for use by the GUI. The directory */usr/tmp* is used by default. To specify an alternate directory, set the environment variable CALIBRE_PMATCH_UI_TMP_DIR to the directory path.

Procedure

1. Use one of these options to open the Calibre Pattern Matching GUI:

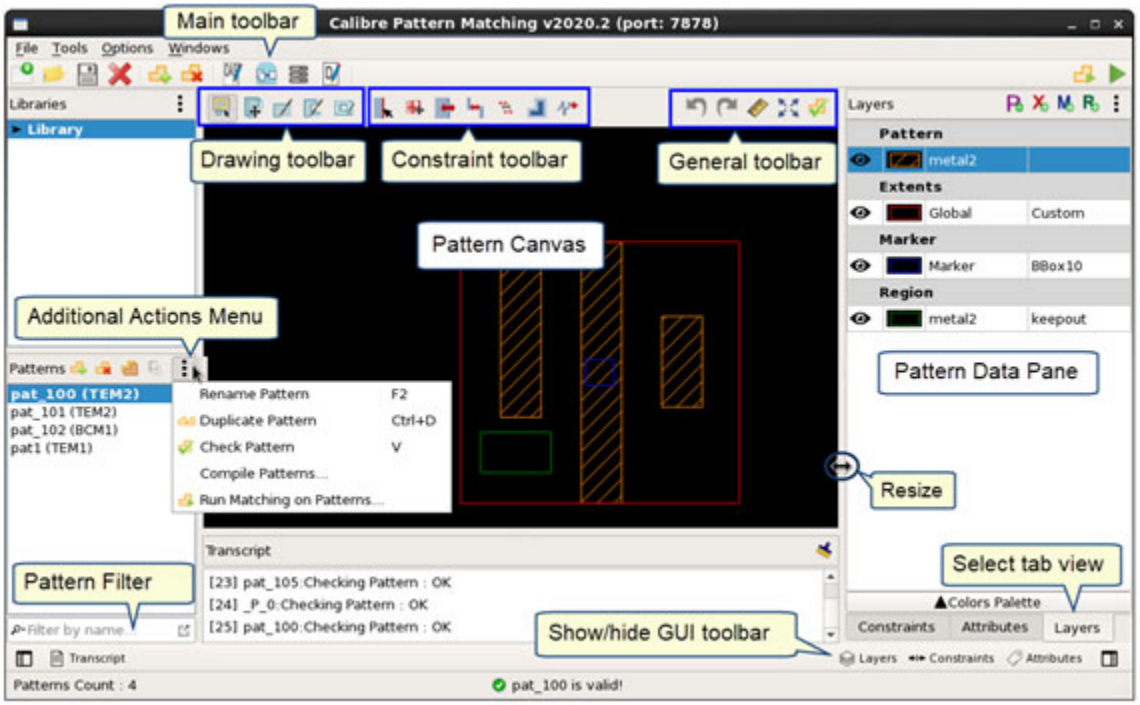
To invoke from …	Do this …
Command line	`calibrepm [pattern_library_path]` where *pattern_library_path* is an optional argument specifying the path to a pattern matching database (PMDB) to open.
Calibre layout viewer	From Calibre DESIGNrev or other Calibre layout viewer: **Verification > Pattern Matching > Open GUI**

2. Do one of the following:

   - Click the **Open Library** icon ( ) to open a pattern library.
   - See "Creating a New Pattern Library" on page 94 to create a new pattern library.

   The following image shows a view of the GUI with a pattern selected.

Figure 3-1. Calibre Pattern Matching GUI



Note

The Pattern Matching GUI is able to load extremely large patterns (more than 25K edges), but does not draw the pattern geometries. You can perform actions that do not require use of the pattern canvas, such as editing pattern attributes, keys, and properties. You can also add non-custom marker and extent layers. The GUI displays a message stating that the pattern size exceeds the drawing limit and provides the option to write the pattern to an OASIS file for viewing.

3. See “Setting GUI Preferences” on page 88 to set grid preferences, snapping behavior, and ruler preferences.

Related Topics

Calibre Pattern Matching GUI

Working with Libraries

Working with Patterns

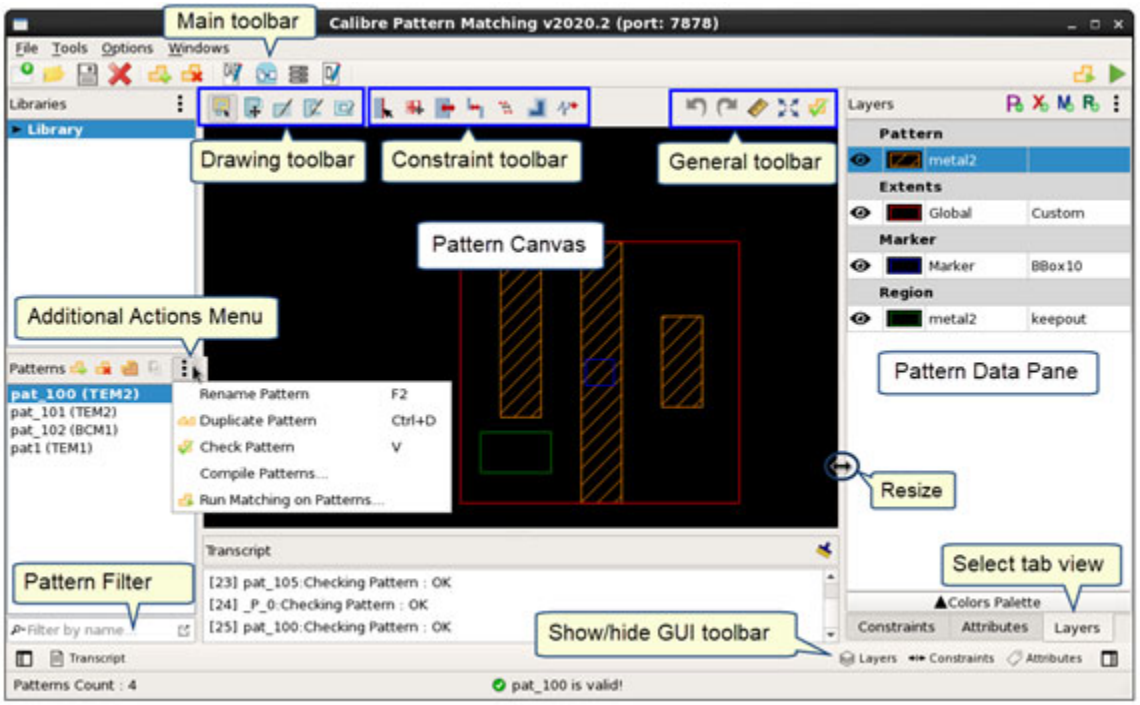
Calibre Pattern Matching GUI

Access with:

- "calibrepm [*pmdb*]" from the command line
- **Verification > Pattern Matching > Open GUI** from a Calibre layout viewer

The Calibre Pattern Matching GUI enables you to view and edit patterns in a library. You can also run pattern library utilities, such as merge and compile.

Figure 3-2. Calibre Pattern Matching GUI Main Window



Objects

GUI Object	Description
Main toolbar	Includes quick access buttons for common actions. Hover over a button for a tooltip.
Pattern Filter	Filters the pattern list based on pattern names, properties, and keys. "Filtering a Pattern Library" on page 97
Pattern Canvas	Displays the pattern. The drawing and constraint toolbars include tools for creating and editing patterns. Choose **Options > Preferences** to set editor and ruler options. "Drawing Shapes with the Pattern Matching GUI" on page 85

GUI Object	Description
Pattern Data Pane	Displays the layer, constraint, or attribute data for a pattern, depending on the tab selected at the bottom of the pane. "Working with Layers" on page 125 "Working with Constraints" on page 136 "Working with Properties and Attributes" on page 158
Show/hide toolbar	Includes buttons to show and hide GUI panes. The **Windows** menu also includes settings for pane display.
Resize	Hover the mouse between GUI panes to resize the panes.

Related Topics

Invoking the Calibre Pattern Matching GUI

Drawing Shapes with the Pattern Matching GUI

You use the drawing toolbar to draw pattern shapes, custom markers, and custom layer extents.

Prerequisites

- A pattern library is open in the GUI. See "Invoking the Calibre Pattern Matching GUI" on page 82.

Procedure

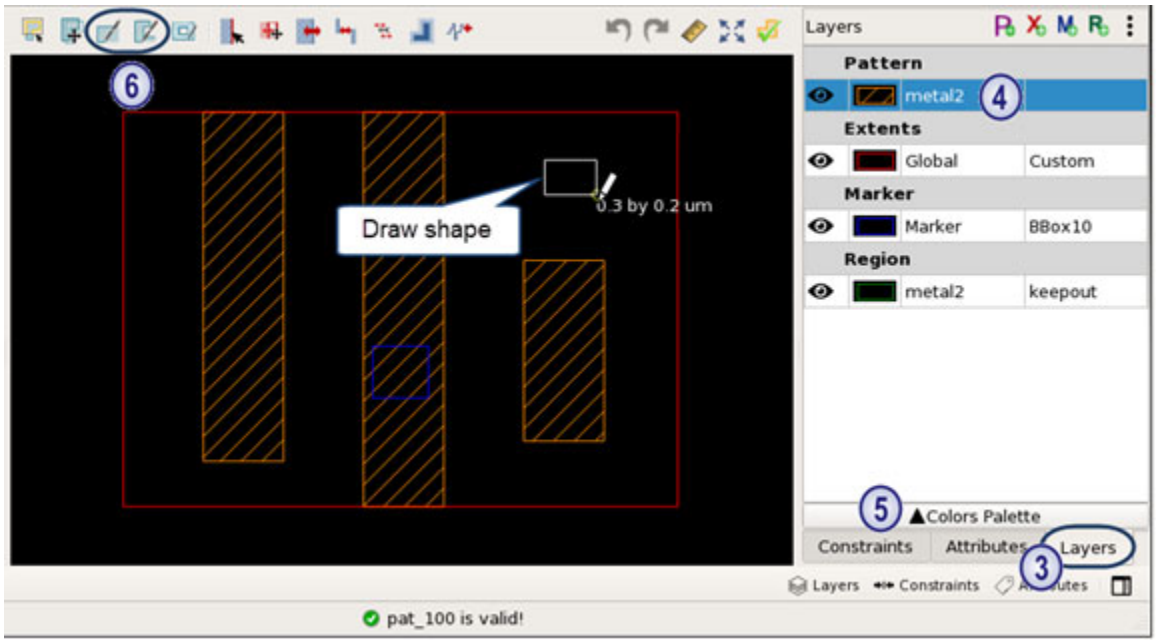
1. (Optional) Choose **Options** > **Preferences** > **Editor Preferences** and set grid and background color preferences.
2. Select a pattern in the pattern list.
3. View the **Layers** tab in the panel to the right of the pattern canvas.
4. Select the layer you want to draw the object on.

   The layer can be a pattern layer, a custom extent layer, a custom marker layer, or a region layer. If you need to add a layer, click the appropriate button in the **Layers** tab toolbar (, , , or)
5. (Optional) To change the color and fill of the selected layer, click "Colors Palette" at the bottom of the **Layers** tab.
6. With the desired layer still selected, click the **Create Rectangle** ( ) or **Create Polygon** ( ) button and draw a shape.

Note

To draw only Manhattan shapes when drawing polygons, press and hold the Ctrl key while drawing. To set the drawing mode for polygons, choose **Options > Preferences > Editor Preferences** and choose Arbitrary or Manhattan.

The following image shows drawing a rectangle for the metal2 pattern layer.



Note

Take care with adding non-Manhattan shapes, which may have their length or width snap differently if an edge is off-grid. In addition, non-Manhattan pattern polygon edges should not intersect with pattern extents.

7. (Optional) Do the following to add a hole to a shape:
   a. Click the **Create Hole** button ( ).
   b. Click polygon you want to place the hole in.
   c. Draw the hole. You can only draw a rectangular hole. To create a non-rectangular hole, draw two or more adjoining or overlapping holes.
8. If needed, click the **Move** button ( ) to make adjustments to the shapes.

   To move a polygon, edge, or vertex, click and drag the object. When moving vertexes, the edge connected to the vertex is also moved.
9. Click the **Check Pattern** button ( ) to check that the pattern is legal.

Related Topics

Invoking the Calibre Pattern Matching GUI

Creating a New Pattern

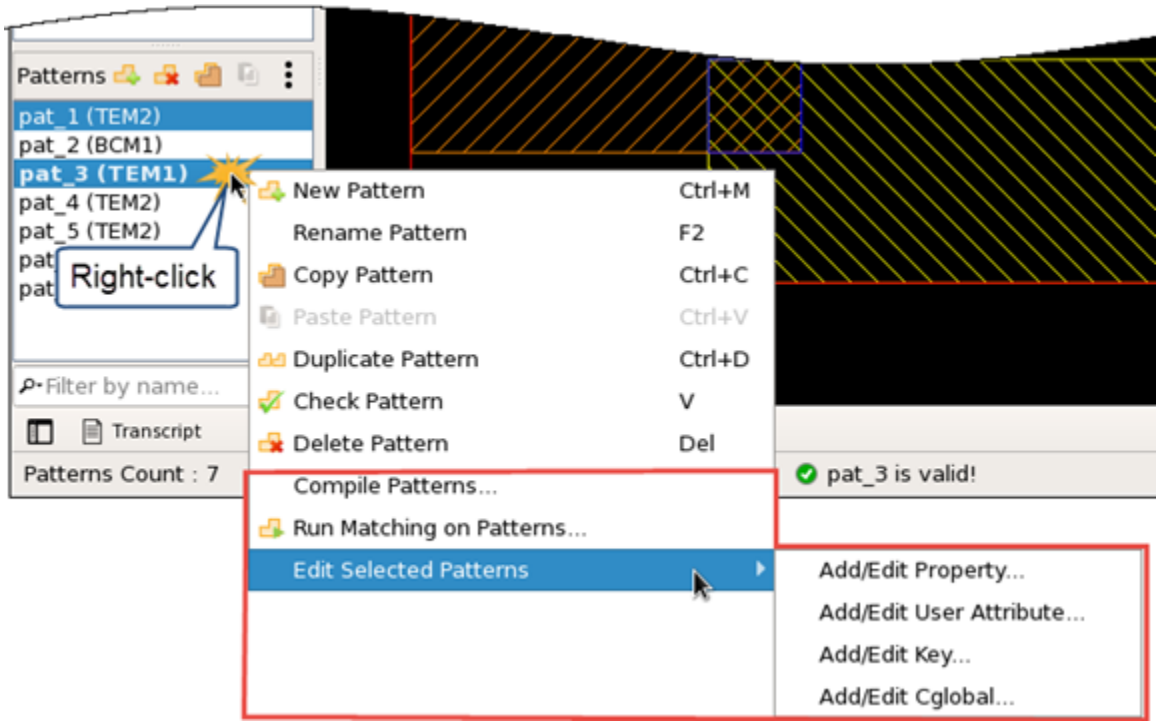
Adding A Boolean Layer Derivation Result to a Pattern

Working with Constraints

Operating on Multiple Selected Patterns

For some actions you can select multiple patterns in the pattern list and apply the same operation to the selected patterns.

To do this, first select the patterns in the pattern list, then right-click. The right-click menu includes selections to compile, run pattern matching, and edit the patterns.



You can also add the same single edge constraint to multiple edges; see "Adding a Single Edge Constraint to Multiple Edges" on page 140.

Related Topics

Defining Custom Property Values in a Pattern

Attaching Keys to a Pattern

Defining Custom Attribute Values in a Pattern

Adding Global Constraints (cglobal)

Running Pattern Matching From the GUI

Compiling a Pattern Library with the Calibre Pattern Matching GUI

Setting GUI Preferences

You can set editor preferences such as the grid style and background color. You can also set ruler preferences and license timeout. Preferences are saved when you close the GUI and restored when you start your next session.

Prerequisites

- The Calibre Pattern Matching GUI is open. See "Invoking the Calibre Pattern Matching GUI" on page 82.

Procedure

1. Choose **Options > Preferences > Editor Preferences** and set options as desired in the Editor Preferences dialog box.
   - **Grid Spacing** — Select the grid spacing and units.
   - **Grid Style** — Select no grid, points, or lines.
   - **Color** — Select the background and grid color.
   - **Polygon Tool**
     - **Arbitrary** — Allows drawing polygons with edges at arbitrary angles.
     - **Manhattan** — Forces polygons to be Manhattan shapes, with edges orthogonal to the database axes.

       Note

       You can also force drawing of Manhattan shapes for individual polygons—press and hold the Ctrl key while drawing the polygon. The works even when Arbitrary is selected as the drawing mode.
   - **Display constraint labels in canvas** — As described.
   - **Enable alternate canvas panning** — Enables the G, H, Y, and J keys for panning left, up, down, and right in the pattern canvas.

     This is useful if the arrow keys for panning in the pattern canvas are not functional. This happens on platforms that do not support the XKEYBOARD protocol extension.
2. Choose **Options > Preferences > Ruler Preferences** to open the Ruler Preferences dialog box.

- **Direction** — Choose whether rulers measure in an arbitrary direction, along Manhattan lines, or along a multiple of 45 degrees.
- **Snapping** — Choose whether to snap ruler endpoints to the grid, snap to a pattern vertex or edge, or to turn snapping off.
- **Units** — Choose the units for ruler measurements.
- **Allow multiple rulers** — Check to enable multiple rulers. If unchecked, any previous rulers are deleted before making a new measurement.

Tip

Use Shift-k to delete all rulers. Use Alt-k to delete the most recent ruler.

Note: Some X-window clients, such as Exceed on Demand, map the right-hand Alt key to the X window, and map the left-hand Alt key to the Windows system.

3. Choose **Options > Preferences > License Timeout** to configure the license timeout.

Related Topics

Using Custom Constraint Labels

Keyboard Shortcuts (Hotkeys) in the Calibre Pattern Matching GUI

Most toolbar button in the Calibre Pattern Matching GUI have a keyboard shortcut, or hotkey.

Note

You must use the right-hand Alt key on some X-window clients, such as Exceed on Demand. On some clients the right-hand Alt key is mapped to the X window, while the left-hand Alt key is mapped to the Windows system.

The keyboard arrow keys and the Delete key are not functional on platforms that do not support the XKEYBOARD protocol extension. The GUI provides alternate keyboard hotkeys to make navigation easier on these platforms; these are noted with “alternate”, to indicate the hotkey works on such platforms.

Tip

Some buttons are not visible by default. Click the corresponding More Options icon ( ) to view additional selections.

Hotkey function depends on the proper GUI area being in focus. The hotkey definitions are organized by function.

- “Utilities and Tools Hotkeys” on page 90

Utilities and Tools Hotkeys

Button	Definition	Hotkey
	Generate OASIS	Ctrl-Shift-o
	Merge libraries	Ctrl-Shift-m
	Cluster library	Ctrl-Shift-c
	Compile library	Ctrl-Shift-d
	Unfold library	Ctrl-Shift-u
	Run pattern matching	Ctrl-Shift-t

Library Hotkeys

Button	Definition	Hotkey
	Create new library	Ctrl-n
	Open library	Ctrl-o
	Save library	Ctrl-s
	Close library	Ctrl-w
	Open library attributes	Ctrl-t
	Reload library	Ctrl-r

Pattern Hotkeys

Button	Definition	Hotkey
	Create new pattern	Ctrl-m

Button	Definition	Hotkey
	Delete pattern	Delete key
	Copy pattern	Ctrl-c
	Paste pattern	Ctrl-v
(none)	Rename Pattern	F2
	Duplicate pattern	Ctrl-d
	Check Pattern	v
	Run pattern matching on patterns	
Pattern Scrolling — In the header of the pattern list, choose > **Pattern Scrolling** to display the scrolling toolbar. The hotkeys are functional regardless of the toolbar visibility.		
	Select first pattern	a (alternate hotkey)
	Select previous pattern	w (alternate hotkey)
	Select next pattern	s (alternate hotkey)
	Select last pattern	d (alternate hotkey)

Pattern Drawing Hotkeys

Button	Definition	Hotkey
Drawing Shapes		
	Select items	s
	Move items	m
	Create rectangle	r
	Create polygon	p Press and hold Ctrl while drawing to draw Manhattan shapes.
	Create notch	n
Drawing Constraints		
	Select edges	Ctrl-b

Button	Definition	Hotkey
	Edit constraints	Shift-m
	Single edge constraint	Shift-s
	Edge to edge constraint	Shift-d
	Relational constraint	Shift-r
	Cglobal constraint	Shift-c
	Set fixed width extent (BCM patterns only)	Shift-e
	Toggle jog behavior	Shift-j
General		
	Delete item	Delete key or d (alternate hotkey)
	Undo	u
	Redo	Shift-u
	Ruler	k Shift-k to delete all rulers Alt-k to delete most recent ruler
	Zoom all (fit)	f
	Check pattern (validate)	v

Alternate Pattern Canvas Panning Hotkeys

You can enable alternate hotkeys for panning as follows:

1. Choose **Options > Preferences > Editor Preferences**, or press e on the keyboard.
2. Under Canvas Panning, check “Enable alternate canvas panning.”

This enables the G, H, Y, and J keys for panning left, up, down, and right.

Pattern Data Pane Hotkeys

Button	Description	Hotkey
Layers Tab		
	Add Pattern Layer	Alt-Shift-l
	Add Layer Extent	Alt-Shift-x
	Add Marker Layer	Alt-Shift-m
	Add Region Layer	Alt-Shift-r
	Group layers by relevance	Alt-Shift-v
Attributes Tab		
	Manage Properties	Alt-Shift-p
	Manage Attributes	Alt-Shift-a
	Add Key	Alt-Shift-k

Working with Libraries

You can create a pattern library (PMDB) using the Calibre Pattern Matching GUI. You can also perform certain operations on the library, such as merging, compiling, and running pattern matching.

Creating a New Pattern Library

You can create a new pattern matching database (PMDB) from the Calibre Pattern Matching GUI.

Prerequisites

- You have met the "Calibre Pattern Matching Product Requirements" on page 16.

Procedure

1. Invoke the GUI from the command line:

   ```
   calibrepm
   ```

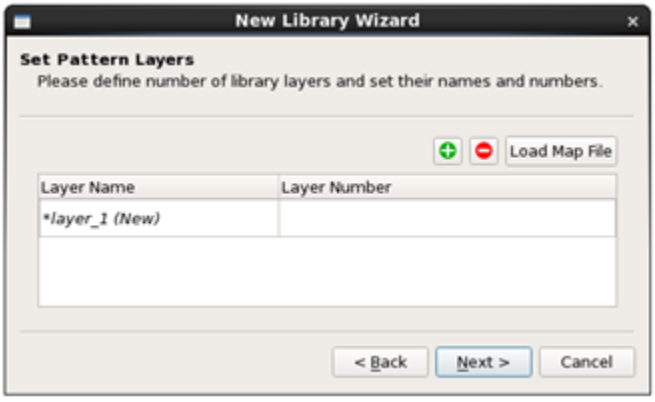
2. Choose **File > New Library** or click the **New Library** button () in the toolbar.

   The New Library Wizard opens.

3. Fill in the required information:

   a. Define Library Path step — Enter the library name.

b. Set Pattern Layers step — Add layer definitions manually or click the **Load Map File** button to populate the layers with a layer map file. See "Loading a Layer Mapping File for a Pattern Library" on page 133 for information.



Caution

If pattern libraries might be merged at a later time, it is important to use a consistent set of layers in your libraries. When merging libraries, the tool only uses the layer definitions from the first input library. It is important that layer names, order, and number of layers are identical for all the input libraries. See "Merging Pattern Libraries Using the GUI" on page 99.

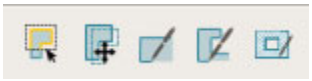
c. Create Empty Pattern step — Leave the "Create pattern" checkbox checked. The following steps assume a pattern is selected in the pattern list.

d. Click **Finish** to close the wizard.

The library is created with an empty pattern.

4. (Optional) Select **Options** > **Preferences** > **Editor Preferences** to specify grid options and canvas background color.

5. Add pattern geometries using the drawing action buttons above the pattern canvas.



Add at least one pattern shape. A library must have at least one non-empty pattern in order to save the library. See "Creating a New Pattern" on page 115 for more details about creating a pattern.

6. (Optional) Add marker, extent, and region layers using the toolbar buttons on the **Layers** tab.

(Add Marker Layer) **(Add Layer Extent)** **(Add Region Layer)**

See "Working with Layers" on page 125.

7. (Optional) Click the **Attributes** tab and add attributes, properties, keys, comments, and orientations.

   See "Working with Properties and Attributes" on page 158.

8. (Optional) Specify jog tolerance for the library. See "Specifying Jog Tolerance for Patterns in a Library" on page 96.

9. Click the **Save Library** button () or choose **File > Save Library** to save the library.

   Errors are reported in the Transcript pane.

Results

A new pattern library.

Related Topics

Creating a New Pattern

Specifying Jog Tolerance for Patterns in a Library

Jog tolerance is specified at the library level and applies to all pattern edges by default. You can select individual TEM pattern edges and disable jog tolerance for that edge. When jog tolerance is specified for an edge, jogs in the pattern or layout that are within the tolerance are ignored during pattern matching.

Prerequisites

- A pattern library is open in the Calibre Pattern Matching GUI. See "Invoking the Calibre Pattern Matching GUI" on page 82.

Procedure

1. Choose **File > Library Attributes** or click the **Library Attributes** button () to open the Library dialog box
2. Click the **Max jog Tolerance** category.
3. Select "Set Library Value."
4. Specify a jog tolerance in user units in the "Value" field.
5. Click **OK**.

   A message is displayed to confirm that you want to save the library options. Click **Yes** to proceed.

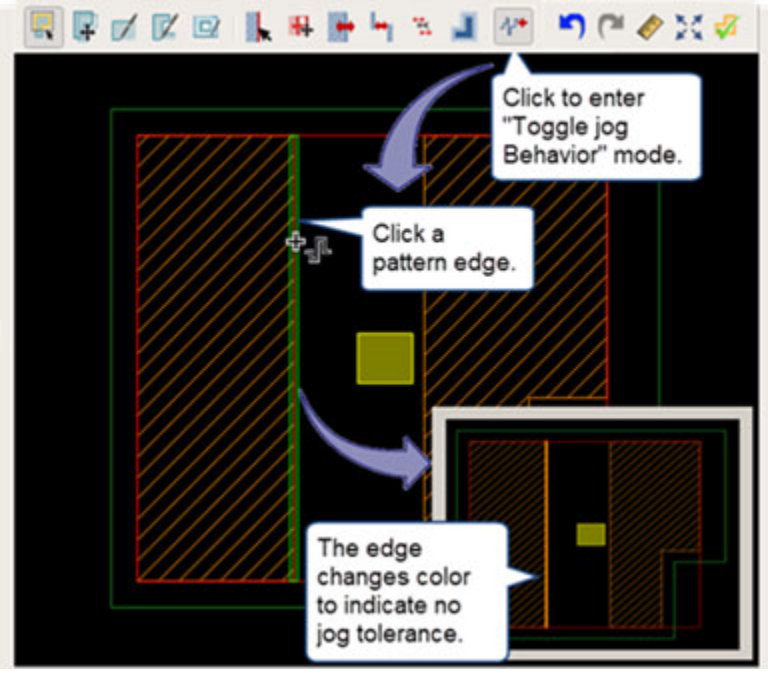
   The specified jog tolerance is applied to all pattern edges.

6. (Optional) Do the following to remove jog tolerance from selected TEM pattern edges:
   a. Select a TEM pattern in the pattern list.

      You cannot toggle jog behavior for edges in BCM patterns.

   b. Click the button to enter the Toggle jog Behavior mode, which is indicated with a mouse cursor.

   c. Click a pattern edge to toggle the jog tolerance application for the edge.



The pattern edge changes color, as shown in the preceding figure. In addition, the entry "Jog Tags" is added to the **Constraints** tab on the right side of the GUI, under "jog Removal Tags."

Related Topics

Jog Tolerance in Pattern Matching

Filtering a Pattern Library

You can filter a library based on pattern names, properties, and keys. You can save the filtered patterns in a new library.

Prerequisites

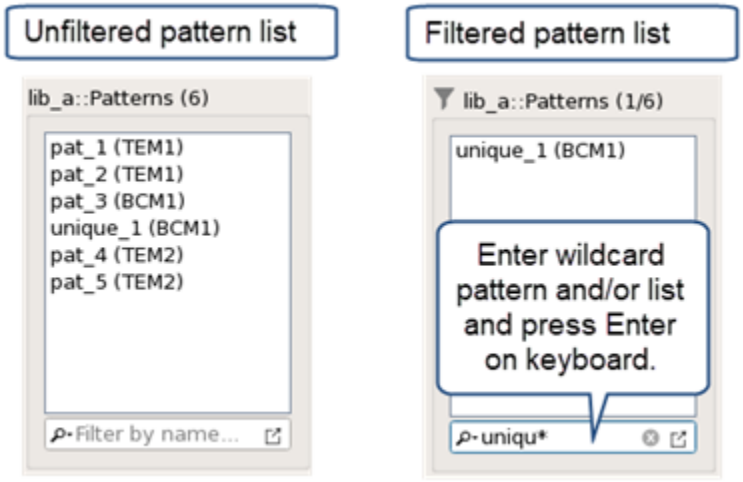
- A pattern library is open in the Calibre Pattern Matching GUI. See "Invoking the Calibre Pattern Matching GUI" on page 82.

- The Explorer panel is visible. If it is not visible, choose **Windows > Explorer**.

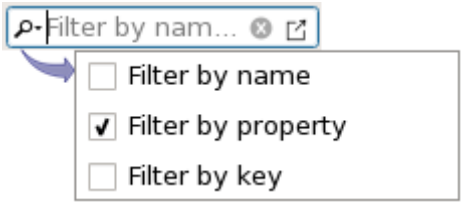
Procedure

1. To filter by pattern name, enter a filter pattern with a wildcard character (*) or a list of pattern names in the filter entry box below the pattern list:

Figure 3-3. Filter Pattern Library



2. To filter by properties or keys, click the filter button and select the filter type:



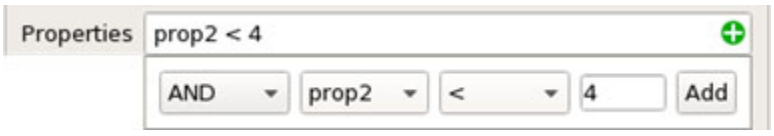
   Hover over the filter field to see the allowed syntax. Most comparison operators are supported for property filtering. Boolean “and” and “or” are supported for both property and key filtering.

3. For advanced filtering, such as filtering by both properties and keys, do the following:

   a. Click the button in the filter entry field to open the advanced filter window.

      You can drag the filter window within the transcript pane to get a tabbed view or a side by side view.

   b. Click the button to display controls for creating a filter expression:



   c. Click **Filter** to apply the filter parameters.

4. (Optional) Choose **File > Save Filtered Patterns as** to save the filtered patterns in a new library.

Merging Pattern Libraries Using the GUI

You can merge two or more pattern libraries into one library using the Calibre Pattern Matching GUI. Merging removes duplicate patterns. By default, patterns are considered duplicate if the pattern shape, constraints, layer extents, regions, and markers are identical.

For more information regarding pattern and library precedence and additional options for merging libraries, see pdl_lib_mgr "merge."

Caution

When merging libraries, the tool only uses the layer definitions from the first input library. It is highly important that layer names, order, and number of layers are identical for all the input libraries. An error is reported if the number of layers differ between input libraries and the merge does not take place.

When layers definitions differ, patterns defined in subsequent libraries use the layer definitions from the first library after the merge operation, resulting in patterns that do not apply to the intended layer and incorrect pattern matches. See "About Merging Pattern Libraries" on page 23.

Prerequisites

- The Calibre Pattern Matching GUI is open. See "Invoking the Calibre Pattern Matching GUI" on page 82
- Two or more pattern libraries.
- The layer definitions are identical between the libraries, including layer order.

  See "Adding Pattern Layers to a Library" on page 128. You can add Don't Care pattern layers as needed in order make the layer definitions match between libraries.

Procedure

1. Click the **Merge Libraries** button () or choose **File > Merge Libraries** to open the Merge Libraries dialog box.
2. Enter the names of the libraries to merge in the "Input libraries" section.
3. Select options. The corresponding merge utility option is given in parentheses.
   - Ignore markers — Ignore markers when comparing patterns (ignore_markers)
   - Ignore regions — Ignore regions when comparing patterns (ignore_regions)

- Ignore layer extents — Ignore per-layer extent differences when comparing patterns (ignore_layer_extents)
- Log file path — Create a log file (log_file)

4. Enter the name of the merged output library in the “Output file” section.
5. (Optional) Specify Run Options.

   “Run turbo” specifies the -turbo argument for the merge utility, which enables multithreaded parallel processing with the specified number of processors.
6. Click **Run** to merge the libraries.

Results

A merged pattern library file. View the log file if the operation did not succeed.

If any of the ignore options are selected, patterns that differ only in the indicated item (markers, regions, or layer extents) are considered duplicates, and only the first such pattern is saved to the merged library.

See the pdl_lib_mgr “merge” command reference for details on how custom properties and pattern orientation are handled.

Running Pattern Matching From the GUI

You can run pattern matching on a layout using the Calibre Pattern Matching GUI. The tool builds the appropriate rule file and executes a Calibre nmDRC run to find pattern matches. The rule file is saved, enabling you to make edits if desired. You can invoke Calibre RVE at the end of the run to view the matches.

You can run on the whole library, or on selected patterns.

Prerequisites

- A pattern library is open in the Calibre Pattern Matching GUI. See “Invoking the Calibre Pattern Matching GUI” on page 82.
- You have a layout in GDS or OASIS format that is suitable for testing the pattern library.

Procedure

1. Select the pattern library name in the library list.
2. (Optional) To run only on selected patterns, select the patterns using Shift- and Ctrl-click in the pattern list.

Tip

You can apply a search filter to the patterns in the library, then select the resulting patterns. See "Filtering a Pattern Library" on page 97.

3. Do one of the following to open the Run Pattern Matching dialog box:

To …	Do This …
Run on the whole library	Click the  button (**Run Pattern Matching**)
Run on selected patterns	Click the  button (**Run Matching on Patterns**)

4. Specify the run directory.
5. Specify Layout Options:
   a. Select the layout source in the dropdown list.
      - **Active Layout** — Use a layout open in a Calibre layout viewer. The GUI automatically connects to an open layout viewer instance. Click **Connect** if you want to connect to a different layout viewer instance.

        Tip

        To display the layout viewer socket number, choose **Verification > RVE/CI Setup**. If the socket server is not running, click **Start Server**.

      - **Layout File** — Use a layout file.

   b. Review the layer table. If the layer table does not have the correct correspondence between the pattern library layers and the layout layers, double-click in a layout layers cell to correct it.
   c. (Optional) Check "Layout Area" to run on a specified region of the layout. Do one of the following to specify the layout region:
      - **Specify manually** — Specify the cell name and the lower left and upper right coordinates of the bounding box.
      - **Specify in the viewer** — Click the **Select Area** button and draw the capture region in the layout viewer. The cell name and coordinates are entered in the dialog box. (Only available when running on an active layout.)
6. (Optional) To specify compile options, click the **Advanced** button and choose the **Compile Options** tab.
   - **Compile before run** — Compiles the selected library or patterns.

Click the  button to choose compile options. See "Compiling a Pattern Library with the Calibre Pattern Matching GUI" on page 105 for an explanation of the options.

- **Use previously compiled file** — Runs with the file you specify in the DMACRO file field. Not available when running on selected patterns.

7. (Optional) To specify general Calibre run options, click the **Advanced** button and choose the **Run Options** tab.
   - **Generate Rules (ONLY)** — Create the pattern matching rule file only.
   - **Run Calibre** — Start a Calibre nmDRC run from the command line.

     "Run Hier" and "Run Turbo" specify the corresponding command-line arguments.
   - **Launch Calibre Interactive** — Open Calibre Interactive nmDRC to start the run.

     You can adjust options using the Calibre Interactive GUI before starting the run.
   - **Enable results reporting in cell space** — Add the DRC CELL NAME YES CELL SPACE XFORM ALL statement to the rule file used for the pattern matching run.

     When this option is checked, the Calibre RVE menu selection **Highlight > Highlight in Context** is available; by default it is not available.
8. Click **Run**.

Results

A transcript pane is displayed listing the command line used (if a DRC run was performed), transcript output, and the output files created.

The following output is generated, where <*lib*> is replaced with the library name:

- pm_test_<*lib*>.svrf — The pattern matching rule file

If "Compile before run" is checked:

- <*lib*>.svrf — The compiled library

If a DRC run was performed, these files are also created:

- pm_test_<*lib*>.drc.rdb — An ASCII format DRC results database
- pm_test_<*lib*>.dfm.rdb — A DFM RDB
- pm_test_<*lib*>.drc.report — A DRC summary report
- pm_test_<*lib*>.log — The transcript output

If "Launch Calibre Interactive" was checked, this file is also created:

- pm_test_<*lib*>_runset — A Calibre Interactive runset

  Other files used internally by Calibre Interactive are also created.

Click the folder icon ( ) to open any file. Text files are opened in a text viewer. RDB files are opened in Calibre RVE for DRC. See "Using Calibre RVE to View Match Results" on page 187.

Related Topics

DMACRO and CMACRO for Pattern Matching

Highlighting DRC Results [Calibre RVE User's Manual]

Running Pattern Matching From a Calibre Layout Viewer

You can open the Run Pattern Matching GUI from a Calibre layout viewer and run pattern matching on the open design. The tool builds the appropriate rule file and executes a Calibre nmDRC run to find pattern matches. The rule file is saved, enabling you to make edits if desired. You can invoke Calibre RVE at the end of the run to view the matches.

Prerequisites

- A layout in GDS or OASIS format is open in a Calibre layout viewer.
- A previously created pattern library, which is suitable for running on the open layout.

Video

The video "Pattern Match With Count Markers" demonstrates how to run pattern matching on an open design. The video also demonstrates how to add count markers as a compile option. Count markers provide an accurate count of matches and enable you to know if pattern matches overlap.

Procedure

1. Choose **Verification > Pattern Matching > Run Match** in the layout viewer.
2. Specify the pattern library in the Open Library File dialog box.

   The Run Matching dialog box opens.
3. Edit the Run Directory path, if necessary, in the Run Matching dialog box.

4. Specify Layout Options:

   a. Select the layout source in the dropdown list:

      - **Active Layout** — Use the open layout. The GUI automatically connects to the layout viewer instance that it was invoked from. Click **Connect** if you want to connect to a different layout viewer instance.
      - **Layout File** — Use a layout file.

   b. Review the layer table. If the layer table does not have the correct the correspondence between the pattern library layers and the layout layers, double-click in a layout layers cell to correct it.

   c. (Optional) Check “Layout Area” to run on a specified region of the layout. Do one of the following to specify the layout region:

      - **Specify manually** — Specify the cell name and the lower left and upper right coordinates of the bounding box.
      - **Specify in the viewer** — Click the **Select Area** button and draw the capture region in the layout viewer. The cell name and coordinates are entered in the dialog box. (Only available when running on an active layout.)

5. (Optional) To specify compile options, click the **Advanced** button and choose the **Compile Options** tab.

   - **Compile before run** — Compiles the selected pattern library.

     Click the  button to choose compile options. See “Compiling a Pattern Library with the Calibre Pattern Matching GUI” on page 105 for an explanation of the options.

   - **Use previously compiled file** — Runs with the compiled pattern library file you specify in the DMACRO file field.

6. (Optional) To specify general Calibre run options, click the **Advanced** button and choose the **Run Options** tab.

   - **Generate Rules (ONLY)** — Create the pattern matching rule file only.
   - **Run Calibre** — Start a Calibre nmDRC run from the command line.

     “Run Hier” and “Run Turbo” specify the corresponding command-line arguments.

   - **Launch Calibre Interactive** — Open Calibre Interactive nmDRC to start the run.

     You can adjust options using the Calibre Interactive GUI before starting the run.

   - **Enable results reporting in cell space** — As described. This adds the DRC CELL NAME YES CELL SPACE XFORM ALL statement to the rule file used for the pattern matching run.

When this option is checked, the Calibre RVE menu selection **Highlight > Highlight in Context** is available; by default it is not available.

7. Click **Run**.

Results

A transcript pane is displayed listing the command line used (if a DRC run was performed), transcript output, and the output files created.

The following output is generated, where <*lib*> is replaced with the library name:

- pm_test_<*lib*>.svrf — The pattern matching rule file

If "Compile before run" is checked:

- <*lib*>.svrf — The compiled library

If a DRC run was performed, these files are also created:

- pm_test_<*lib*>.drc.rdb — An ASCII format DRC results database
- pm_test_<*lib*>.dfm.rdb — A DFM RDB
- pm_test_<*lib*>.drc.report — A DRC summary report
- pm_test_<*lib*>.log — The transcript output

If "Launch Calibre Interactive" was checked, this file is also created:

- pm_test_<*lib*>_runset — A Calibre Interactive runset

  Other files used internally by Calibre Interactive are also created.

Click the folder icon ( ) to open any file. Text files are opened in a text viewer. RDB files are opened in Calibre RVE for DRC. See "Using Calibre RVE to View Match Results" on page 187.

Related Topics

DMACRO and CMACRO for Pattern Matching

Count Markers

Highlighting DRC Results [Calibre RVE User's Manual]

Compiling a Pattern Library with the Calibre Pattern Matching GUI

You can use the Calibre Pattern Matching GUI to compile patterns into a DMACRO file for use in a Calibre pattern matching run.You can compile all patterns in the library or selected patterns.

The GUI uses the pdl_lib_mgr compile utility.

Note

The DMACRO format changed in releases 2019.2 and 2021.2. DMACRO files created with Calibre 2021.2 cannot be used by earlier versions of Calibre. DMACRO files created from 2019.2 to 2021.1 cannot be used in Calibre versions 2019.1 and earlier.

DMACRO files created prior to 2021.2 can be used in later versions of Calibre.

Prerequisites

- A pattern library is open in the Calibre Pattern Matching GUI. See "Invoking the Calibre Pattern Matching GUI" on page 82.

Procedure

1. Select the pattern library name in the library list.
2. (Optional) If you want to compile only selected patterns, use Shift- and Ctrl-click to select patterns in the pattern list.

   Tip

   You can apply a search filter to the patterns in the library, then select the resulting patterns. See "Filtering a Pattern Library" on page 97.

3. Do one of the following:

   - To compile the library, click the **Compile Library** button ( ) or choose **Tools > Compile Library**.
   - To compile selected patterns, right-click on the selected patterns from Step 2 and choose **Compile Patterns**.

4. Enter the name for the compiled DMACRO file in the "SVRF Path" field.
5. Select the **Markers** tab. (It is selected by default.)

   a. (Optional) Check "Enable unique count marker" to generate a count marker for matched patterns. See "Count Markers" on page 45 for details.

      Note

      If the patterns being compiled do not all have the same set of markers, you must enable "Generate all markers" in Step b and/or specify compile-time markers in Step c.

   b. (Optional) Check "Generate all markers" to add the gen_all_markers option to the compile command.

   c. (Optional) Specify compile-time markers in the Marker table.

Compile-time markers can do two things: specify which of the existing markers in the PMDB to output to the DMACRO and add new markers to the DMACRO. See "Compile-Time Markers" on page 230 and "Example 3: marker and gen_all_markers" on page 232 within the pdl_lib_mgr compile command reference for complete information.

Do the following to specify a compile-time marker:

i. Click the **Add Marker** () button.

ii. Select an existing marker name in the Marker dropdown list or type in a new marker name.

iii. Select the marker type in the Type dropdown list.

You can add multiple compile-time markers. The markers in the Marker table correspond to the compile-time markers specified with the marker argument set in the compile command.

6. (Optional) Click the **Options** tab to set compile and CMACRO options.

   - **Exclude error patterns on compilation** — Specifies the exclude_error_patterns option for the compile command.

     Unchecked — The compilation fails if any pattern has errors.

     Checked — Patterns with errors are not saved to the DMACRO and compilation succeeds.

   - **Enable runtime CMACRO options**

     Unchecked — The CMACRO command in the rule file cannot specify an *options_string*. The CMACRO options specified in the option table are used.

     Checked — The CMACRO command in the rule file can specify an *options_string*. This enables you to change CMACRO options at runtime. Specify a password if desired. If a password is specified, it must be provided with the -password argument in the CMACRO *options_string* at runtime. Runtime options override the options in the GUI option table.

   - **Option table** — You can set several arguments for the CMACRO *options_string* in the table. See "CMACRO for Calibre Pattern Matching" on page 177 for information on the options. The initial view of the table displays the default values that are used.

7. (Optional) Specify Run Options.

   "Run turbo" specifies the -turbo argument for the compile utility, which enables multithreaded parallel processing with the specified number of processors.

8. Click **Run** to compile the pattern library.

Results

A transcript pane is displayed in the dialog box. The transcript includes the compile command line and transcript output.

An encrypted DMACRO file is saved to the file specified in the "SVRF Path" field. See "DMACRO File for Calibre Pattern Matching" on page 174.

The name of the compiled DMACRO file is displayed at the end of the transcript pane. Click the  button to open the Run Pattern Matching dialog box. See "Running Pattern Matching From the GUI" on page 100.

See this topic for instructions on using the DMACRO file in an SVRF rule file: "Using a DMACRO in a Rule File for a Pattern Matching Run" on page 185.

Related Topics

Compiling a Pattern Library With the compile Utility

Exporting an OASIS File

You can export a pattern library to a graphical OASIS file using the Calibre Pattern Matching GUI. The OASIS file can be used to visualize patterns in layout format. The OASIS layout can also be used in a pattern matching run to validate the pattern library.

You can select the pattern layers that are output to the OASIS file. By default, all polygon information from pattern shapes, constraints, regions, markers, and extents is output. You can select the layers that are output.

See the pdl_lib_mgr write_oasis utility for complete information on the command options and the generated output.

Prerequisites

- A pattern library is open in the Calibre Pattern Matching GUI. See "Invoking the Calibre Pattern Matching GUI" on page 82.

  If you want to group patterns in the output OASIS file by predefined clusters, the library should be the output of a cluster operation. See the pdl_lib_mgr cluster utility or "Defining Pattern Clusters in a Library" on page 110.

Procedure

1. Select the pattern library name in the library list.
2. Choose **Tools** > **Generate OASIS** or click the **Generate OASIS** button () to open the Generate OASIS dialog box.
3. Enter the name of the output file in the "OASIS Path" field.

4. Click each of the **OASIS Layers**, **Global Extent**, **Markers**, and **Regions** tabs, and select whether to output the corresponding pattern layer.

   The GUI specifies an output OASIS layer number—you can change this if desired.

5. Click the **Options** tab and specify the following options:

   - **Precision** — The OASIS file precision. By default this is set to the precision of the first pattern.

     Caution

     If the patterns in your library have different precisions, set the output precision to the lowest common multiple of precision values in your library to ensure that information is not lost.

   - **Space** — The space between patterns in microns. The default is the maximum width or height across all patterns. Set to 0 for the minimum spacing.
   - **Text Height** — The text height in microns, which is used to determine the placement coordinate (x, y) of text below the pattern. The default text height is one quarter of the pattern to pattern spacing.
   - **Pattern** — Select whether to output all patterns or a specific pattern.
   - **Columns** — The number of columns in the output. By default, patterns are output with roughly the same number of rows and columns.

6. (Optional) Choose clustering options in the area below the tab display.

   - **Cluster patterns** — Check this option to cluster the output patterns based on a clustering property in the input library. Each pattern cluster is placed in a separate OASIS cell.
   - **Cluster property name** — Check this option and provide a property name if your input library does not use the default cluster property name of “cluster”.
   - **Minimum cluster number** — Check this option if you want to specify a minimum number of patterns in a cluster. The default value is 1.

7. (Optional) Specify Run Options.

   “Run turbo” specifies the -turbo argument for the write_oasis utility, which enables multithreaded parallel processing with the specified number of processors.

8. Click **Run** to generate the OASIS file.

Results

An OASIS layout file. The transcript of the write_oasis utility is displayed.

Defining Pattern Clusters in a Library

You can group similar patterns in a pattern library into a cluster using the Calibre Pattern Matching GUI. Clustering patterns is helpful for generalizing patterns; that is, grouping similar patterns together and determining minimum and maximum constraints that may apply to a group of patterns. You can also save a library in which each group of clustered patterns is represented by a pattern with constraints, thus reducing the number of patterns in the library.

You can also apply clustering to your pattern library with the pdl_lib_mgr cluster utility.

Tip

Some options in the Cluster Dialog dialog box include a help ? button, which displays a small diagram demonstrating the effect of the option.

Prerequisites

- A pattern library is open in the Calibre Pattern Matching GUI. See “Invoking the Calibre Pattern Matching GUI” on page 82.

Procedure

1. Select the pattern library to cluster in the pattern library list.
2. Choose **Tools > Cluster Library** to open the Cluster Library dialog box, or click the  button.
3. Specify the output files and formats in the Output Files section.
   - **Path** — The filename for the clustered library. You can specify an absolute or relative path.
   - **Generate OASIS file** — (Optional) Generate an OASIS database consisting of pattern shapes for each pattern, where each cluster is placed in its own OASIS cell. The write_oasis utility with the cluster option is used to generate the OASIS database.
   - **Compress library path** — (Optional) Generate a compressed library in addition to the clustered library. A compressed library contains one “cluster pattern” for each cluster group, where the cluster pattern includes constraints such that each pattern in the cluster is represented. Each cluster pattern includes a property named Cluster_size that contains the number of patterns represented by the cluster pattern.
4. Specify the property name for the cluster property. The default is “cluster”.

   The cluster property is added to all patterns that are part of a cluster, with an integer value corresponding to the cluster number. Patterns that are not part of a cluster do not receive a cluster property.
5. Select options on the **Pattern Movement** tab in the Clustering Options area.

- **None** — Do not use “Allowed shift” or “Center halo.”
- **Allowed shift** — Allow the pattern center to shift by the specified “Shift Value.”
- **Center halo** — Specify a center halo region. Exact matching is used in the center halo region. Any options specified on the **Edge Movement** tab apply to the region outside the center halo.

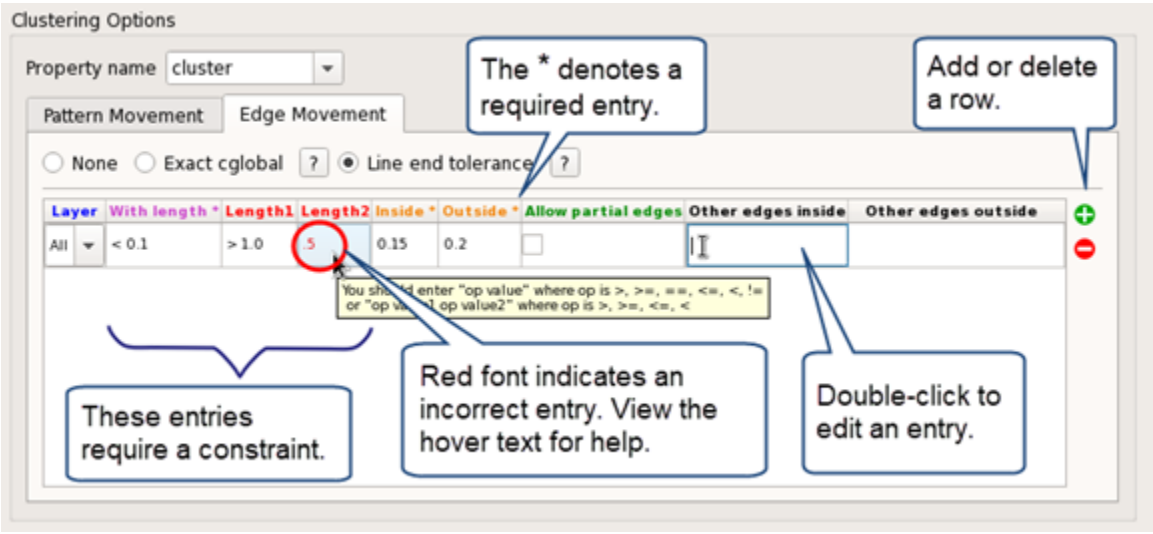
  Halo Radius — Specifies the size of the center halo region, which is a square region centered on the pattern center and with a width of twice the “Halo Radius” value.

  Ignore outer halo — Specifies to only use the center halo region to determine cluster matches (corresponds to the crop option for the cluster utility). This option disables selections on the **Edge Movement** tab.

6. Select options on the **Edge Movement** tab in the Clustering Options area.
   - **None** — Do not allow edge movement.
   - **Exact cglobal** — Allow edges of exact patterns to move by the specified cglobal value when determining a cluster match. The cglobal value should be less than 1/4 of the minimum feature size (width or spacing) in the pattern.
   - **Line end tolerance** — Allow movement of line ends when determining a cluster match.

     The parameters correspond to the line_ends option of the cluster utility.

Figure 3-4. Cluster Library Dialog Box



  - **Layer** — The layer to which the line-end tolerance applies. “All” denotes all layers.
  - **With length** — A constraint specifying the line-end length required in order for the line-end tolerance to be applied.

- **Length1** — A constraint specifying the length of one side of the line end. If Length2 is not specified, the constraint applies to only one side of the line end.
- **Length2** — A constraint specifying the length of the second side of the line end.
- **Inside** — The allowed movement to the inside.
- **Outside** — The allowed movement to the outside.
- **Allow partial edges** — Include line ends with partial edges (edges with one or two vertices that lie on the pattern extent and at least one adjacent edge that is a virtual edge).
- **Other edges inside** — The allowed movement to the inside for edges that do not satisfy the line-end criteria (With length, Length1, and Length2).
- **Other edges outside** — The allowed movement to the outside for edges that do not satisfy the line-end criteria.

7. Specify Run Options.
   - **Generate cluster script only** — Saves the cluster command line to the specified file.
   - **Run turbo (-turbo)** — Specifies the -turbo argument for the cluster utility, which enables multithreaded parallel processing with the specified number of processors.
8. Click **Run** to apply clustering to the library.

Results

A transcript pane is displayed in the dialog box. The transcript includes the compile command line and transcript output.

The generated output files are listed; click the  icon to open a file.

Unfolding Patterns in a Library Using the GUI

You can unfold the patterns in a library using the Calibre Pattern Matching GUI. Unfolding outputs a series of patterns without constraints. The output patterns are the result of applying the pattern constraints in various permutations.

See the pdl_lib_mgr unfold utility for complete information on the command options and the generated output.

Prerequisites

- A pattern library is open in the Calibre Pattern Matching GUI. See “Invoking the Calibre Pattern Matching GUI” on page 82.

Procedure

1. Select the library name in the library list.
2. Do one of the following:

To …	Do this …
Unfold all patterns	Click the **Unfold Library** button (). This opens the Unfold Library dialog box.
Unfold one pattern	Right-click on the pattern in the pattern list and choose **Unfold Pattern**. This opens the Unfold Selected Pattern dialog box.

3. (Optional) Enter the name of the output file in the Path field of the dialog box.

   The output library is automatically named *unfolded_<input_library>.pmdb*.

   Tip

   If you are unfolding a specific pattern, you may want to add the pattern name to the output library name.

4. (Optional) Specify unfold options:

Option	Description
Pattern limit	Specifies the maximum number of permutations that are created for *each* pattern. Default: 100.
Index property	Specifies the name of a numeric property that is added to each unfolded pattern. The property value is unique for each pattern. The property value starts at one with the first pattern that is output and is incremented by one for each subsequent output pattern. If a library contains a property of the same name, the value is overwritten.
Unfold step	Specifies to unfold patterns based on a step size in microns. • Use linear unfolding — Use the linear_step_size option in unfold rather then the default step_size option. See the option descriptions with the unfold utility.

5. (Optional) Specify Run Options.

   “Run turbo” specifies the -turbo argument for the unfold utility, which enables multithreaded parallel processing with the specified number of processors.

6. Click **Run** to generate the unfolded library.

Results

The unfolded library is opened in the GUI. The transcript of the unfold utility is displayed in the Results tab.

Working with Patterns

You can create and edit patterns using the Calibre Pattern Matching GUI. You can also capture patterns from a layout that is open in a Calibre layout viewer.

Creating a New Pattern

You can use the Pattern Matching GUI to add a new pattern to a pattern library. The drawing tools in the GUI are used to define pattern shapes.

Tip

You can also copy existing patterns. See "Copying Patterns" on page 117.

Prerequisites

- A pattern library is open in the GUI. See "Invoking the Calibre Pattern Matching GUI" on page 82.

Procedure

1. Click the **New Pattern** button () or right-click the pattern library name and select **New Pattern**.
2. Enter a name for the pattern and the precision in the New Pattern dialog box and click **OK**.

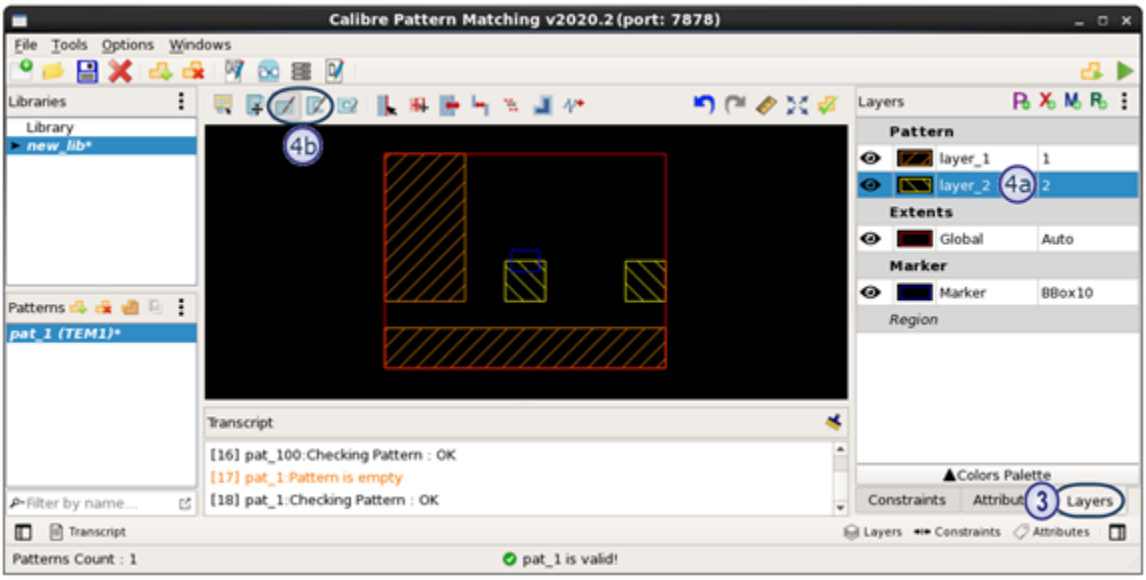
   An empty pattern is displayed in the GUI.
3. View the **Layers** tab in the right panel.
4. Create the pattern.
   a. Select the pattern layer in the **Layers** tab to the right of the pattern canvas.
   b. Click the **Create Rectangle** () or **Create Polygon** () button and draw a shape.
   c. Use the other tools as needed to adjust the pattern shapes.

Note

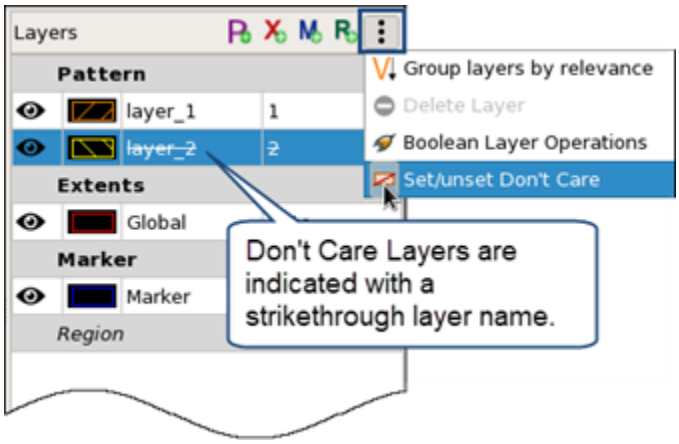
Take care with adding non-Manhattan shapes, which may have their length or width snap differently if an edge is off-grid. In addition, non-Manhattan pattern polygon edges should not intersect with pattern extents.

To draw only Manhattan shapes when drawing polygons, press and hold the Ctrl key while drawing. To set the drawing mode for polygons, choose **Options > Preferences > Editor Preferences** and choose Arbitrary or Manhattan.

5. Click the **Check Pattern** button ( ) to check that the pattern is valid.



6. (Optional) If any pattern layers are Don't Care layers, select the layer in the **Layers** tab and choose > **Set/unset Don't Care**.



See "Don't Care Pattern Layers" on page 55.

7. If you created a BCM pattern, the default pattern type is accordion extent (BCM1). Do the following to change the type:
   a. Click the **Constraints** tab in the pattern data pane to the right of the pattern canvas.
   b. Check "Set Fixed Width Extent" to specify a BCM2 pattern.
8. Click the  (**Save Library**) button to save the changes.
9. (Optional) Add constraints, markers, properties, and attributes to the pattern.

   See the following:

   - "Working with Constraints" on page 136
   - "Working with Layers" on page 125
   - "Working with Properties and Attributes" on page 158

Related Topics

Invoking the Calibre Pattern Matching GUI

Creating a New Pattern Library

Adding A Boolean Layer Derivation Result to a Pattern

Copying Patterns

The Calibre Pattern Matching GUI makes it easy to copy and paste patterns.

Prerequisites

- A pattern library is open in the GUI. See "Invoking the Calibre Pattern Matching GUI" on page 82.
- (Optional) A second pattern library is open if you want to copy a pattern from one library to another. The libraries must have the same number of pattern layers and the pattern layers should be in the same order.

Procedure

1. Select the pattern to copy in the pattern list.
2. Do one of the following, depending on what you want to do:

To …	Do this …
Duplicate the pattern in the current library	Click the **Duplicate Pattern** () button.

To …	Do this …
Copy the pattern to a different library	1. Click the **Copy Pattern** ( ) button. 2. Select the library to copy the pattern to. 3. Click the **Paste Pattern** ( ) button.

The pattern is copied, with "_1" appended to the pattern name.

3. Double-click the pattern name to change it.

Capturing Patterns From a Layout

You can capture a patterns from a layout using the Calibre Pattern Matching GUI. You can define the pattern capture area with a drawn rectangle, with polygons in the layout, with a mask layer, or with a hotspot layer.

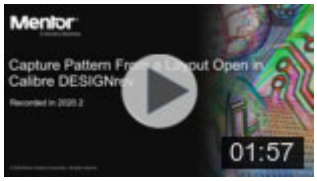
The pattern capture process uses the DFM Pattern Capture operation.

Prerequisites

- A layout that contains polygons is open in one of the Calibre layout viewers (Calibre DESIGNrev, Calibre WORKbench, Calibre LITHOview, or Calibre MDPview).
- A Calibre Pattern Matching license.

Video

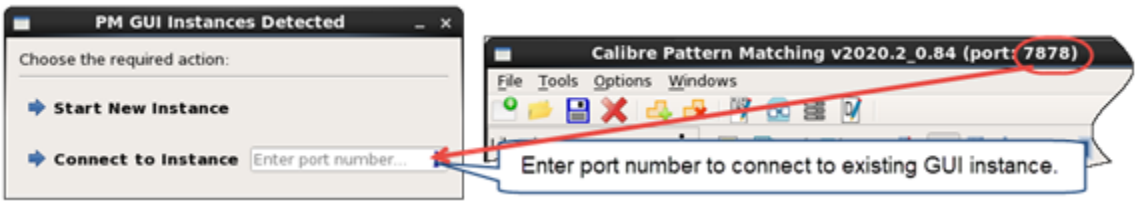
The video "Capture Pattern from a Layout Open in Calibre DESIGNrev" demonstrates the capture process for bounding box capture mode. This mode captures pattern shapes within a rectangular region that you draw in the layout viewer.



Procedure

1. In the layout viewer, choose **Verification** > **Pattern Matching** > **Run Capture** to open the Capture Patterns wizard.

If an instance of the Calibre Pattern Matching GUI is already open, a dialog box opens that prompts you to choose whether to open a new instance or connect to the existing instance.



The dialog box does not appear if the open GUI instance is already connected to the Calibre layout viewer session.

2. Enter a pattern library name in the Output File field.

   If the library does not exist, it is created.

   If the library exists, the captured patterns are added to the library if “Append to an existing library” is checked, otherwise the existing library is overwritten.

   Click **Next**.

3. Choose the Capture Mode using the dropdown selection.

   - **Capture by Bounding Box** — Captures shapes under a rectangle that you draw in the layout viewer. You can also specify the coordinates of the bounding box.
   - **Capture by Mask** — Captures shapes under mask polygons. The polygons correspond to the pattern extent and must be Manhattan shapes. The mask polygons can be specified by a mask layer or by selecting mask polygons in the layout viewer.
   - **Capture by Hotspot** — Captures shapes in a halo region surrounding hotspot polygons. The hotspot polygons can be specified by a hotspot layer or by selecting hotspot polygons in the layout viewer.

     Hotspot capture requires definition of the halo distance. The capture region is a square of width 2*halo centered on the hotspot polygon.

4. In the area at the top of the dialog box, do one of the following, depending on the choice of Capture Mode.

Capture Mode	Actions
Bounding Box	1. Specify the bounding box manually or in the layout viewer: • **Manually** — Specify the cell name and the lower left and upper right coordinates of the bounding box. • **In the viewer** — Click the **Select Area** button and draw the capture region in the layout viewer. The cell name and coordinates are entered in the dialog box. **Tip:** For most cases, view the layout at the top cell level. The capture process clips the pattern for all levels of hierarchy at and below the currently selected cell level. 2. (Optional) Check "Truncate empty spaces in target layers" if you want the pattern extent adjusted to remove empty spaces. Click the ? button in the dialog box for a diagram.
Mask	Do one of the following, depending on whether you are using a mask layer or selecting mask polygons: • **Mask Layer** a. Choose "Select Layer." b. In the dropdown list, choose the layout layer that corresponds to the mask layer. • **Mask Polygons** a. Choose "Select Polygons." b. Select polygons in the layout viewer: In the layout viewer, click the **Select** toolbar button, check the Polygon checkbox, and click one or more polygons to select them. Use Shift-click to select multiple polygons. c. In the dialog box, click **Get Polygons**. The number of selected polygons is reported in the dialog box.

Capture Mode	Actions
Hotspot	Do one of the following, depending on whether you are using a hotspot layer or selecting hotspot polygons: • **Hotspot Layer** a. Choose “Select Layer.” b. Enter a value for the halo in the “Define Halo” textbox. c. In the dropdown list of layers, choose the layout layer that corresponds to the hotspot layer. • **Hotspot Polygons** a. Choose “Select Polygons.” b. Enter a value for the halo in the “Define Halo” textbox. c. Select hotspot polygons in the layout viewer: In the layout viewer, click the **Select** toolbar button, check the Polygon checkbox, and click one or more polygons to select them. Use Shift-click to select multiple polygons. d. In the dialog box, click **Get Polygons**. The number of selected polygons is reported in the dialog box.

5. Specify the Target Layers.

   Polygons on the specified target layers and within the pattern capture area are saved to the captured pattern. The columns in the Target Layer table are defined as follows:

Table 3-1. Target Layers Table Definitions

Column	Definition
Layer Number	The source layer number that is saved in the pattern library.
Layer Name	The layer name that is saved in the pattern library
Layout Layer	The target layer in the layout. Double-click in the cell and choose from the dropdown list.
Don’t Care	When checked, the pattern layer is added as a Don’t Care Pattern Layers, which is ignored during pattern matching.

   The method for filling in the Target Layer table depends on the library:

   - **New library** — One layout layer is entered automatically in the Target Layers table. Construct the target layer list using these actions:
     - Click the  or  button to add or delete a layer. New layers are added at the end of the table.
     - Click the **Get Visible Layers** button to populate the table with the visible layers in the layout viewer.

- Double-click a table cell to edit it.

- **Append to existing library** — The Layer Number and Layer Name columns of the Target Layers table are filled in based on the existing pattern library. Only the Layout Layer and Don't Care columns can be edited.

  Click the **Map Layers** button to fill in the Layout Layer and Don't Care columns based the open layout. The layer number is used first, then the layer name. If no matching layer number or name is found in the layout, the Layout Layer entry is set to Empty and Don't Care is checked.

Note

The pattern can have empty input layers for all but one of your target layers. A layer with no geometries on it can be used to check an empty extent in one of the input layers.

6. (Optional) Click the **Advanced** button to select advanced options.

   The corresponding keyword in the DFM Pattern Capture operation is given in parentheses.

Table 3-2. Advanced Options in the Capture Patterns Wizard

Tab	Comments
General, Match and Hotspot Options	The "Match Options" apply in all capture modes. • **Match Orient** — Enable the library setting "Set match_orient Property." (MATCH_ORIENT) • **Match Rotation** — Enable the library setting "Set match_rotation Property." (MATCH_ROTATION) The "Hotspot Options" apply in "Capture by Hotspot" mode. • **Max Search** — Perform auto-centering with the specified search range. (MAX_SEARCH) • **Max Length** — Segment hotspot shapes with a length greater then the specified value. (MAX_LENGTH)

Table 3-2. Advanced Options in the Capture Patterns Wizard (cont.)

Tab	Comments
General, Pattern Generation Options	• **Pattern Prefix** — Form pattern names as "*prefix*<*n*>", where prefix is the specified string, and <*n*> is the pattern ordinal starting at 1, or the specified "Start Value". (PATTERN_NAME_PREFIX) **Start Value** — Optional starting value for the pattern ordinal. • **Index Property** — Add an integer property with the specified name to each captured pattern. The property contains the pattern index, where numbering starts at 1 or the value given by "Start Value." (INDEX_PROP) • **Cglobal** — Add a cglobal value to the pattern library. The cglobal value should be less than 1/4 of the minimum feature size (width or spacing) in the pattern. (PATTERN_CGLOBAL)
Properties and Keys	The first table contains properties and the second table contains key names. Click the + or − button to add or delete a table row. Double-click a table cell to edit it. (PATTERN_PROP *prop_name value* [FLOAT | INTEGER] PATTERN_KEY key_name)
Markers	Specify a layout layer to use as an additional marker layer. A marker layer cannot be target layer or a capture layer. (LAYER_PATTERN_MARKER *marker_layer marker_name*)
Regions	Specify a region layer. The region layer cannot be a target layer or a capture layer. Region layers are applied to a specific pattern layer. (LAYER_REGION *source type pattern_layer*)
Pattern Orientation	Specify the allowed orientations for a successful match to the patterns in the library. (PATTERN_ORIENT {0 | 1 | 2}) • **All (0)** — Match in any orientation. • **Vertical Orientation (1)** — Match the original orientation or orientations that result from a combination of reflections about the x or y axis. • **Custom (2)** — Match only in the orientation of the captured pattern. The pattern icons indicate what is seen for the "Orientation" setting in the Calibre Pattern Matching GUI, but cannot be changed.

7. Click **Capture**.

   The transcript is displayed. You can view the generated rule file in the transcript.

The Calibre Pattern Matching GUI opens the captured library.

8. (Optional) Click the **Back** button to set up another capture. This is useful in these cases:
   - You are capturing patterns in “Bounding Box” mode or with “Select Polygon” selected, and want to add more patterns to the library. (Make sure “Append to existing library” is checked.)
   - You want to adjust options and re-capture the library. (Make sure “Append to existing library” is not checked.)

Working with Layers

You can manipulate the layers in a pattern using the Calibre Pattern Matching GUI. You can define pattern layers, extent layers, region layers, and marker layers using the GUI.

See "Layers" on page 36 for general information on layers in a pattern library.

Adding Markers to a Pattern

You can add marker layers to a pattern using the Calibre Pattern Matching GUI. When a pattern matching run is performed, the marker shapes are output to show where a match occurs. Patterns can have multiple markers of different types.

See "Marker Layer" on page 42 before deciding what markers to add to a pattern. It is usually more efficient to add computed markers (bbox, bbox10, empty, drawn, and matched) at compile time and only add custom markers to the patterns in the pattern matching database (PMDB). Markers of any type that are defined in a pattern override compile-time markers of the same name.

See these topics for information on adding compile-time markers:

- "Compiling a Pattern Library with the Calibre Pattern Matching GUI" on page 105
- "compile" on page 226 (the pdl_lib_mgr compile utility)

Prerequisites

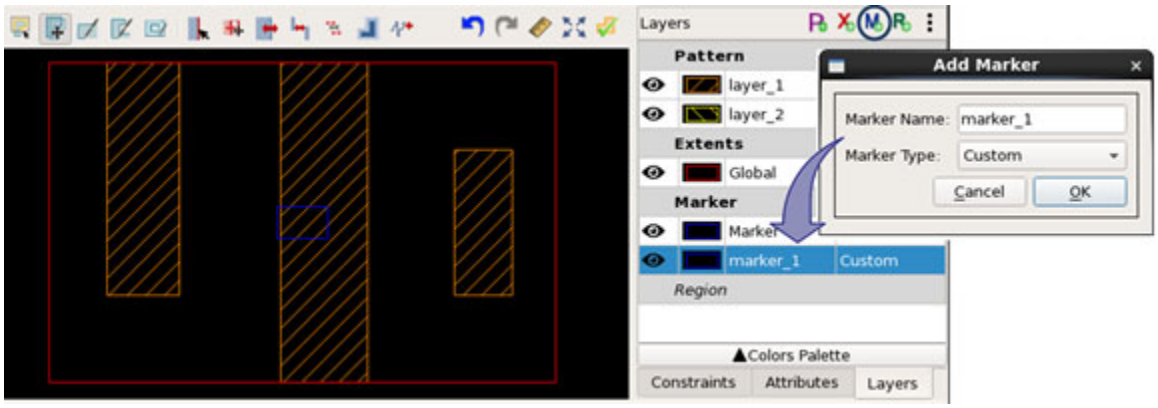
- A pattern library is open in the Calibre Pattern Matching GUI. See "Invoking the Calibre Pattern Matching GUI" on page 82.

Procedure

1. Select a pattern in the pattern list.
2. Click the **Layers** tab at the bottom of the right panel of the GUI.
3. Click the **Add Marker Layer** button ().
4. Enter the marker name and select the marker type in the "Add Marker" dialog box.

5. Click **OK** to close the dialog box.

   The marker is added to the marker list. The following image shows a new marker layer named marker_1 with type Custom.



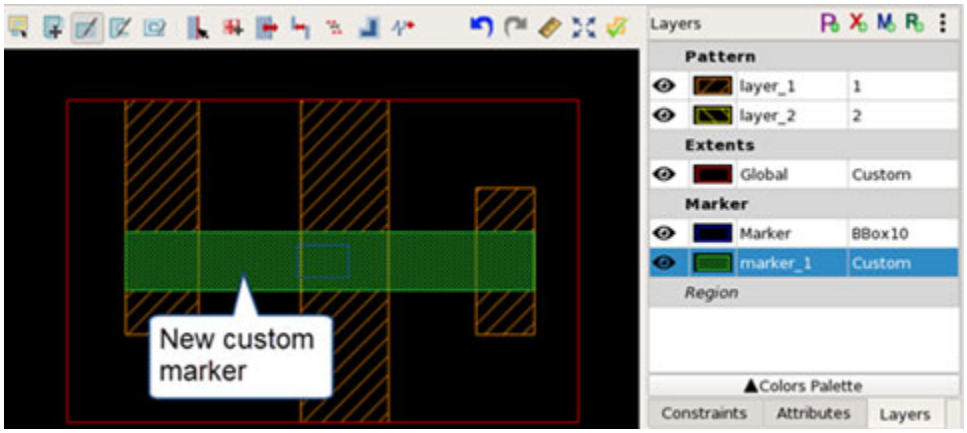
6. (Optional) If you added a Custom marker layer, you need to add a marker shape. The shapes for the other marker types are determined automatically.

   Do the following to draw a custom marker shape:

   a. Click the marker name to select the layer. This is done automatically after adding a marker layer.

   b. Select the rectangle ( ) or polygon ( ) drawing tool and draw the desired marker shape.

   The following image shows a new marker shape, where the fill for the marker_1 layer has been changed using the Colors Palette (located at the bottom of the **Layers** tab).



7. Add more marker layers as needed.

8. Choose **File > Save Library** to save changes to the pattern library.

Related Topics

Marker Layer

Drawing Shapes with the Pattern Matching GUI

Adding A Boolean Layer Derivation Result to a Pattern

Adding Regions to a Pattern

You can add region layers to TEM patterns using the Calibre Pattern Matching GUI. Region layers define areas with special matching properties in a pattern. Region layers are not supported for BCM and XBAR patterns.

Region layers are associated with a particular pattern layer. You may add multiple region layers to a pattern, but only one of each type per pattern layer. See “Region Layers” on page 50 for general information.

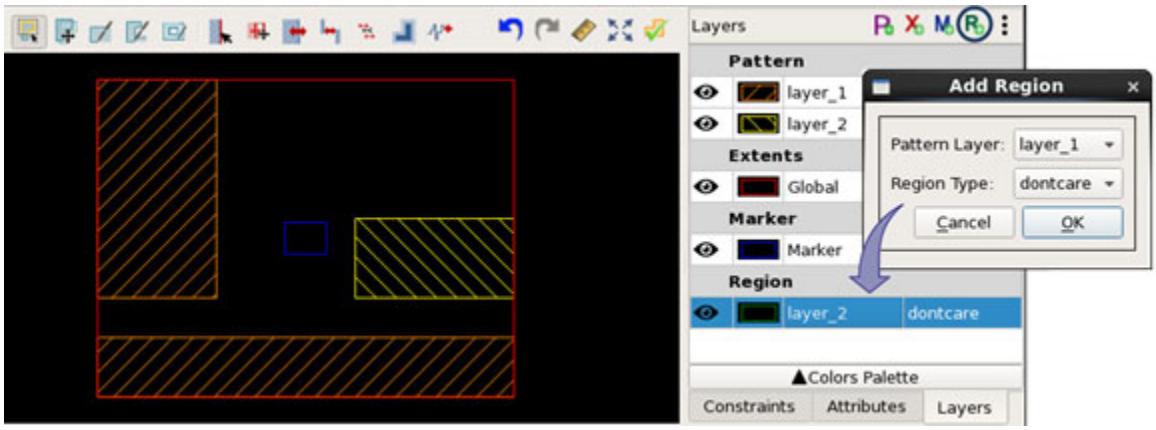
Prerequisites

- A pattern library is open in the Calibre Pattern Matching GUI. See “Invoking the Calibre Pattern Matching GUI” on page 82.

Procedure

1. Select a TEM pattern in the pattern list.
2. Click the **Layers** tab at the bottom of the right panel of the GUI.
3. Click the **Add Region Layer** button ( ) to open the Add Region dialog box.
4. Select the pattern layer the region applies to and the region type.
5. Click **OK** to add the region layer.

   The following image shows a Don’t Care region added for pattern layer layer_2.

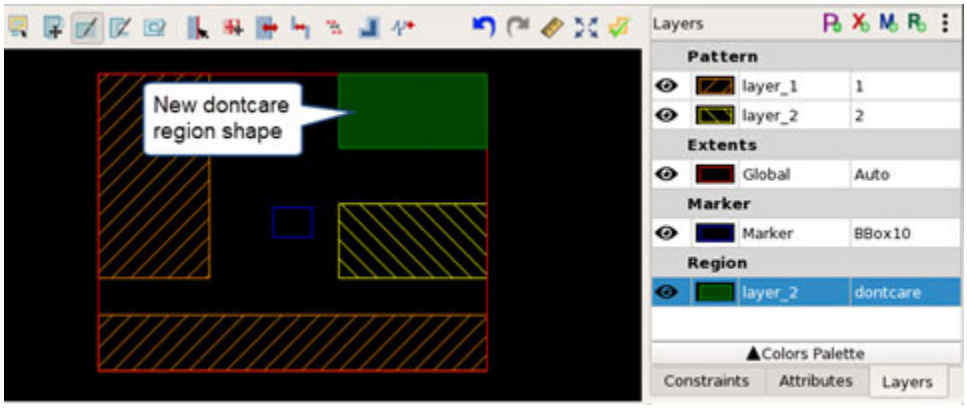


6. Add a region shape to the pattern:

   a. Click the region name to select the region. This is done automatically after adding a region layer.

   b. Select the rectangle ( ) or polygon ( ) drawing tool and draw the desired region shape(s).

   The following image shows a the pattern with a Don't Care region, where the fill for the dontcare layer has been changed using the Colors Palette (located at the bottom of the **Layers** tab.



7. Add more region layers if needed.

8. Choose **File > Save Library** to save changes to the pattern library.

Related Topics

Region Layers

Drawing Shapes with the Pattern Matching GUI

Adding A Boolean Layer Derivation Result to a Pattern

Adding Pattern Layers to a Library

You can add pattern layers to a library with the Calibre Pattern Matching GUI.

Pattern layers can be added to a library in order to make two libraries have the same number of pattern layers, so that the libraries can be merged. If this is the purpose for adding pattern layers, you should be aware of the desired order of the pattern layers. See "About Merging Pattern Libraries" on page 23.

New pattern layers are added as Don't Care Pattern Layers by default. This setting can be changed by selecting the layer on the **Layers** tab and choosing > **Set/unset Don't Care**.

Prerequisites

- A pattern library is open in the Calibre Pattern Matching GUI. See "Invoking the Calibre Pattern Matching GUI" on page 82.

Procedure

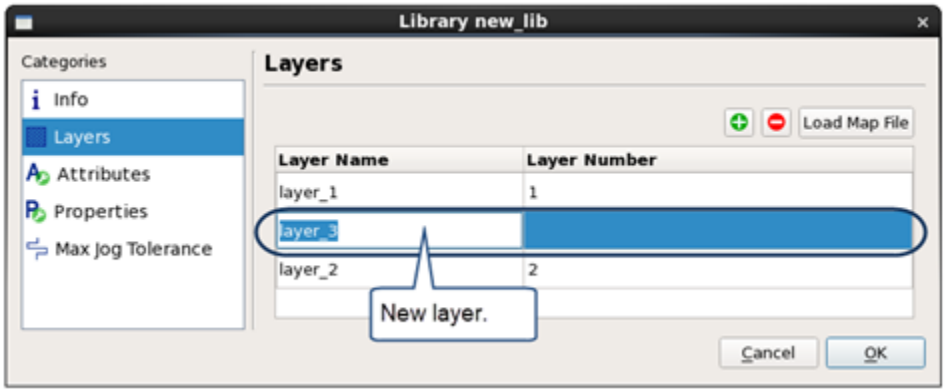
1. Choose **File > Library Attributes** to open the Library dialog box.
2. Select the **Layers** category.

   Tip

   If a pattern is selected, you can click the **Add Pattern Layer** button ( ) on the **Layers** tab instead of doing the first two steps. This opens the Library dialog box to the **Layers** category.

3. In the layer list, click the layer you want to be before the new layer. By default, new layers are added to the end of the layer list.
4. Click the button to add a layer.

   The following image shows the Library dialog box after adding a new layer.

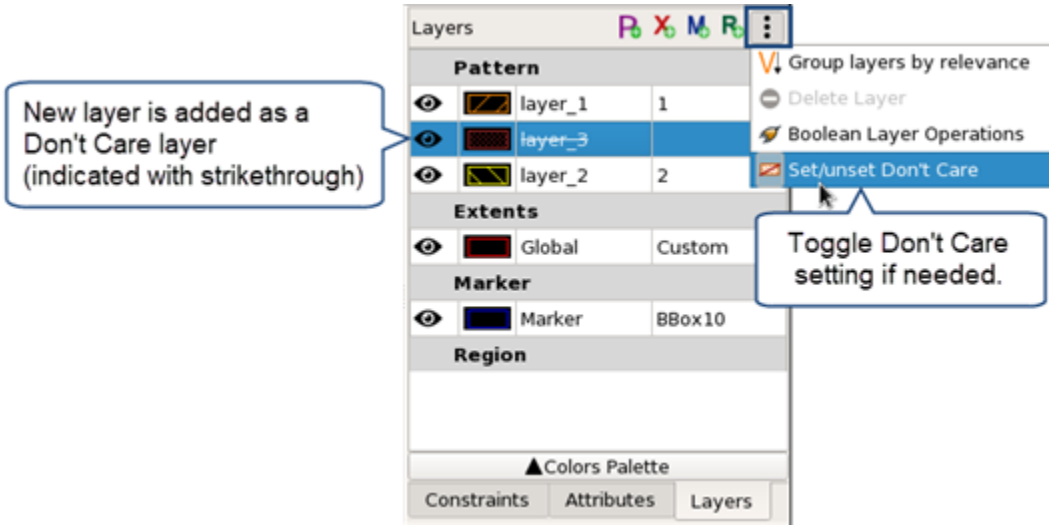


5. Repeat the previous two steps as needed to add more layers.
6. Click **OK** to save the library with the new layer(s).

   A message is displayed to confirm that you want to save the library options. Click **Yes** to proceed. If a pattern is loaded, it is closed.

7. If a pattern is not loaded, select a pattern in the pattern list.

New layers are added as Don't Care pattern layers by default, as shown in the following image of the **Layers** tab.



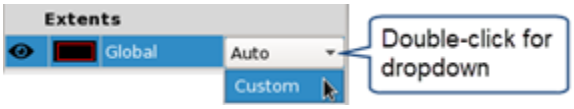
8. If the new layer is not a Don't Care layer, select the layer and choose ⋮ > **Set/unset Don't Care**. Do this for each new layer in each pattern in the library as needed.
9. Choose **File > Save Library** to save changes to the pattern library.

Defining Custom and Per-Layer Extents for a TEM Pattern

You can create a custom or per-layer extent for a TEM pattern using the Calibre Pattern Matching GUI. Custom and per-layer extents enable more control over the region that is matched than is possible with the default extent.

Tip

To change the default global extent to a custom extent, double-click the Auto entry and select Custom in the dropdown box. Use the drawing tools to define the extent boundary.



Prerequisites

- A pattern library is open in the Calibre Pattern Matching GUI. See "Invoking the Calibre Pattern Matching GUI" on page 82.

Procedure

1. Select a TEM pattern in the pattern list.

2. In the **Layers** tab on the right side of the GUI, click the **Add Layer Extent** button ( ).

   This opens the Add Layer Extent dialog box.

3. Do the following in the Add Layer Extent dialog box.

   a. Select the pattern layer the extent applies to.

   b. Select an Extent Type of Custom or Auto.

   The extent layer is added to the Extents list in the **Layers** tab and is automatically selected.

4. If you added a Custom extent, draw the extent shape.

   With the new extent layer still selected, use one of the drawing tools (or) to draw the custom layer extent.

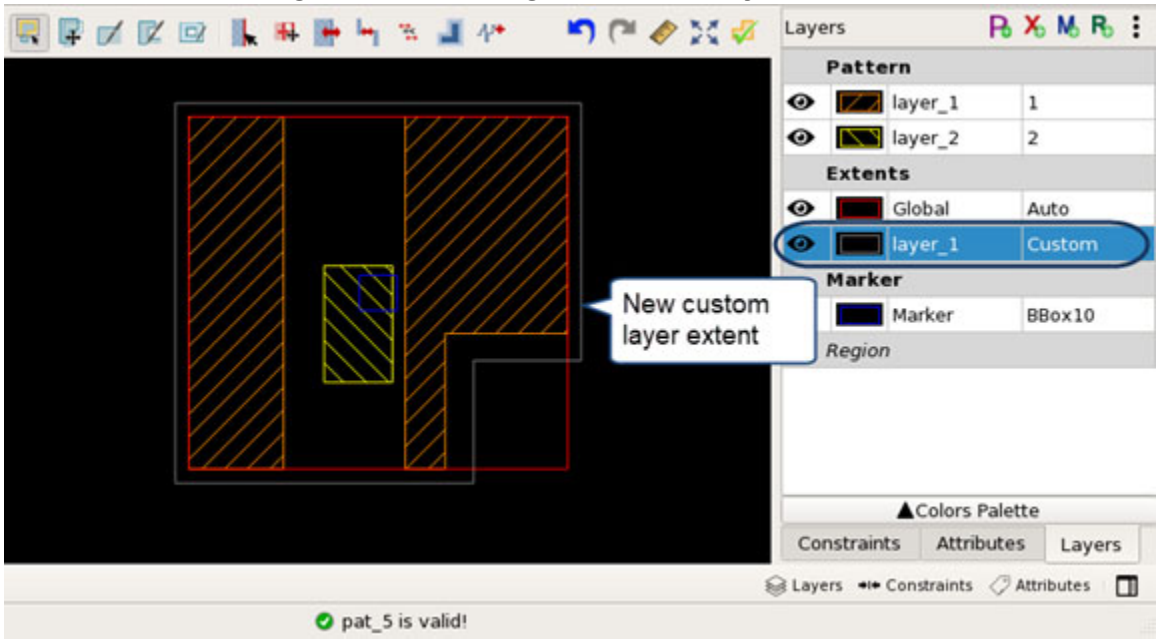
   Defining a custom extent can alter the original pattern, so additional care must be taken to enclose all the desired patterns.

5. Click the **Check Pattern** button ( ) to check that the pattern is valid.

6. Click the (**Save Library**) button to save the changes.

   See the following image.

Figure 3-5. Adding a Custom Layer Extent



Related Topics

Using Custom Constraint Labels

Adding A Boolean Layer Derivation Result to a Pattern

You can add the results of a layer derivation to a pattern layer. The OR, AND, NOT, and XOR layer operations are supported.

Adding the result of a layer derivation is a one-time operation. If the input layers to the layer operation are changed at a later time, the output of the layer operation is *not* updated.

Prerequisites

- A pattern library is open in the Calibre Pattern Matching GUI. See "Invoking the Calibre Pattern Matching GUI" on page 82.

Procedure

1. Select a pattern in the pattern list.
2. Click the **Layers** tab at the bottom of the right panel of the GUI.
3. Choose ⋮ > **Boolean Layer Operations**.

   The Boolean Layer Operations dialog box is displayed.

4. Select the input layers, layer operation, and the output layer.

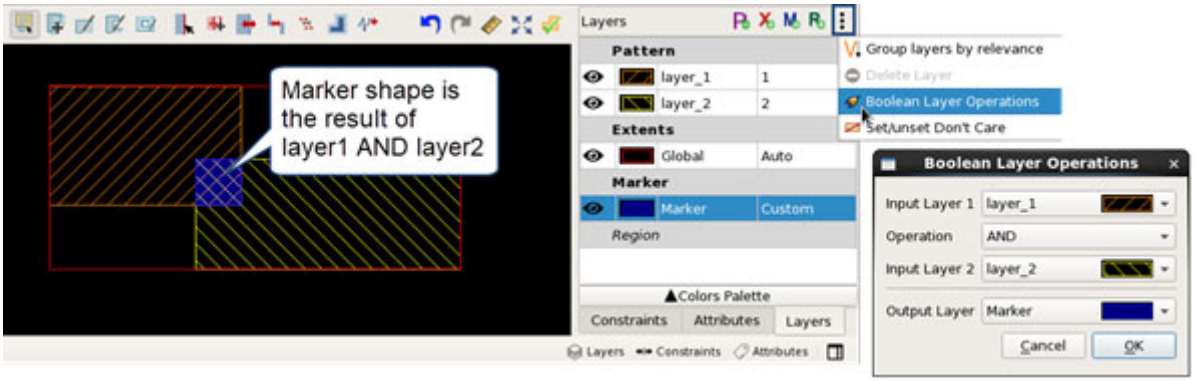
   Input and output layers can be pattern layers, custom marker layers, region layers, and pattern or layer extents. Constraints on the input layers are ignored.

5. Click **OK**.

   The result of the layer derivation is added to the output layer. If the output layer already has drawn polygons, those polygons are retained and the layer derivation result is added to the layer. The output of the layer derivation is not merged with existing polygons on the layer.

Examples

This example shows an AND layer derivation used to create a custom marker.



Loading a Layer Mapping File for a Pattern Library

A layer mapping file is a text file that specifies the correspondence between layer names and layer numbers. You can load a layer mapping file when creating a new pattern library or for an existing library.

Prerequisites

- The Calibre Pattern Matching GUI is open. See "Invoking the Calibre Pattern Matching GUI" on page 82.
- A layer mapping file in the format of the Calibre DESIGNrev layerprops file. See "layerprops File Format" in the *Calibre DESIGNrev Layout Viewers User's Manual*.

  Only the first column (layer number and optional datatype) and fourth column (layer name) are used. The layer name can include alphanumeric characters and underscores only. All six elements are required and there may be only one space between each element. For example:

```
% cat mymap.layerprops
25 pink light_speckle poly 0 1
30.1 green diagonal_right_wide m1_1 0 1
30.2 red carets m1_2 0 1
```

Procedure

1. Do one of the following to get to the **Load Map File** button, depending on whether you are creating a new library or have an existing library.

For …	Do this …
A new library	1. Click the **New Library** button ( ) in the toolbar. 2. Provide the library name and click **Next**.

For …	Do this …
An existing library	1. Click the **Open Library** button ( ) button to open the library. 2. Click the **Library Attributes** button ( ) to open the Library dialog box. 3. Click the **Layers** category.

2. Click the **Load Map File** button, and specify the layer mapping file.

Results

- **New Library** — The layer table is filled in with the information in the layer map file.

  Continue with creating the new library, as described in "Creating a New Pattern Library" on page 94.

- **Existing Library** — The layer table is updated with the information in the layer map file. If the layer mapping file contains more layers than the library, the extra layers are ignored. If the layer mapping file contains fewer layers than the library, the layer table is updated with the layer names and numbers for the lines in the mapping file.

Managing Layer Properties in the Calibre Pattern Matching GUI

The layer properties determine how the Calibre Pattern Matching GUI displays different layers. Layer properties can be set in the GUI and saved to a Calibre DESIGNrev layerprops file. You can load an existing layerprops file to set the layer properties used by the GUI.

The layerprops file defines the layer name, color, and fill pattern. See "layerprops File Format" in the *Calibre DESIGNrev Layout Viewers User's Manual*.

Layer properties apply to all pattern libraries open in the GUI, and to newly created libraries and patterns. When you save layer properties from the GUI or load a layerprops file, the layerprops file is applied the next time you open the Calibre Pattern Matching GUI.

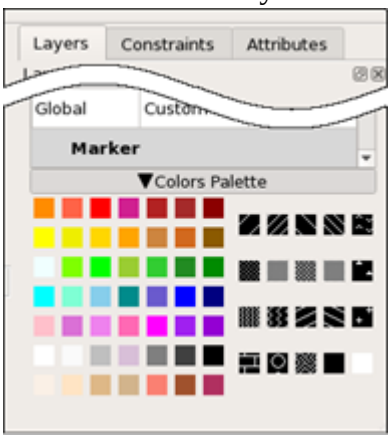
Prerequisites

- (Optional) A Calibre DESIGNrev layerprops file. This is only needed if you want to load an existing layerprops file.
- The Calibre Pattern Matching GUI is open. See "Invoking the Calibre Pattern Matching GUI" on page 82.

  You can load, save, and reset layer properties without a pattern library loaded in the GUI. However, you must load a library and select a pattern in order to set layer properties using the GUI.

Procedure

Do one or more of the following actions to manage layer properties.

To …	Do this …
Load an existing layerprops file.	Choose **Options** > **Load Layer Properties File**.
Save the layer properties defined in the GUI	Choose **Options** > **Save Layer Properties File**.
Reset layer properties to the default	Choose **Options** > **Reset Layer Properties**.
Change layer properties using the GUI	1. Load a pattern library and select a pattern. 2. Select a layer in the **Layers** tab. 3. Expand the “Colors Palette” pane at the bottom of the Layers tab and select a layer color and pattern. 

Working with Constraints

Constraints enable you to specify the distance an edge can shift and still be considered a match. They allow for variations in the pattern when Calibre searches for pattern matches. Constraint are added using the Calibre Pattern Matching GUI.

Adding Global Constraints (cglobal)

Global constraints apply the same single edge constraint to all edges. For TEM patterns, you can apply a global constraint to all layers or to selected pattern layers.

Prerequisites

- A pattern library is open in the Calibre Pattern Matching GUI. See “Invoking the Calibre Pattern Matching GUI” on page 82.
- One or more patterns are selected which meet these requirements:
  - TEM pattern or a single layer BCM pattern.
  - There are no edge-to-edge constraints defined in the pattern. Edge-to-edge constraints must be deleted before adding a global constraint.
  - There are no overlapping real vertices.

Procedure

1. Do one of the following:

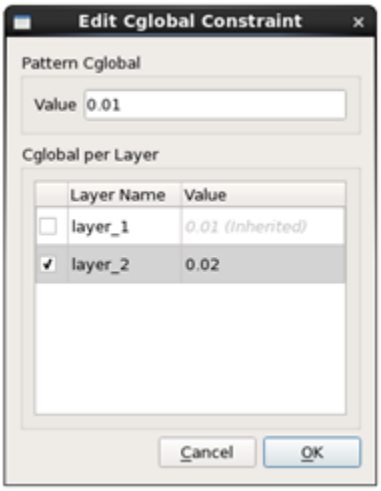
For …	Do this …
Single pattern	Click the **Add Cglobal Constraint** () button above the pattern canvas.

For …	Do this …
Multiple patterns	Right-click on the selected patterns and choose **Edit Selected Patterns > Add/Edit Cglobal**.

2. Fill in the values in the Edit Cglobal Constraint dialog box and click **OK**.

   The cglobal value should be less than 1/4 of the minimum feature size (width or spacing) in the pattern.

   The following image is for a TEM pattern. The pattern cglobal value is 0.01 and a cglobal per layer of 0.02 is applied to layer_2. The layer_1 inherits the pattern cglobal value of 0.01.

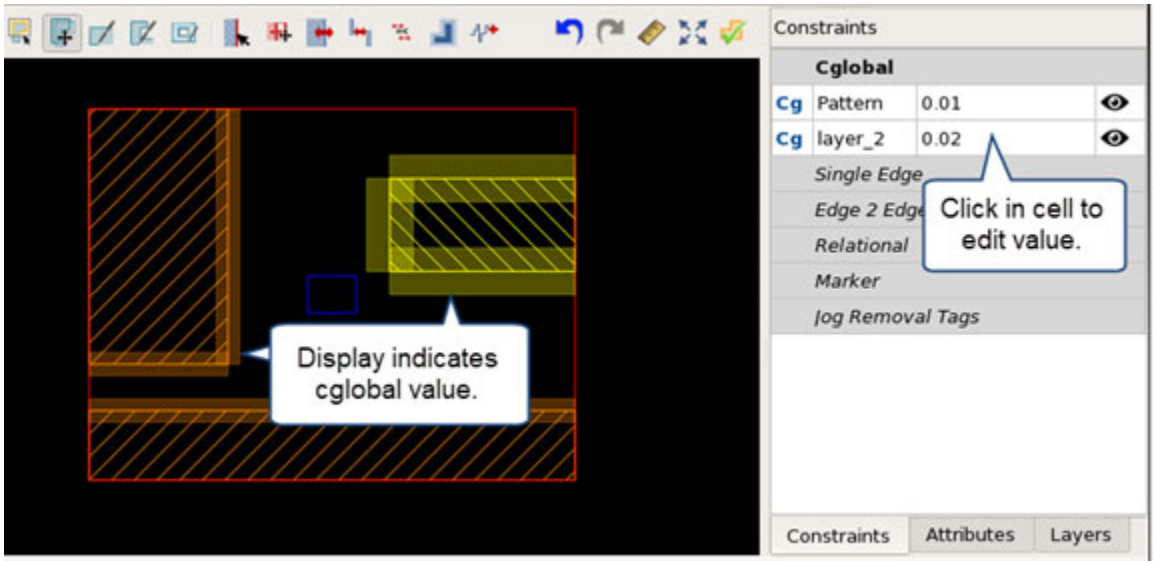


3. Click the **Constraints** tab at the bottom of the right panel to view the cglobal constraint in the pattern data pane.

Results

The pattern is displayed as in the following image. You can edit cglobal constraints by clicking the  button or clicking a cglobal value in the constraint display. For TEM patterns, you can

add constraints to edges with a cglobal value—the new constraint overrides the cglobal setting for the edge.



Adding a Single Edge Constraint

Single edge constraints specify the allowed movement for a single edge and are applied using the Calibre Pattern Matching GUI. You can add single edge constraints to TEM patterns and fixed extent BCM patterns.

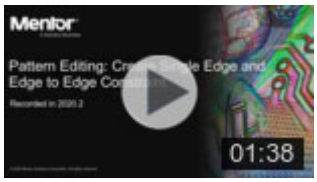
For TEM patterns, you can add a single edge constraint to an edge with a cglobal constraint. The cglobal constraint is removed for that edge but remains for other edges in the pattern.

Prerequisites

- A pattern library is open in the Calibre Pattern Matching GUI. See “Invoking the Calibre Pattern Matching GUI” on page 82.
- One of the following pattern types is selected:
  - A TEM pattern.
  - A fixed extent BCM pattern (BCM2) without a global constraint.
- The selected pattern is validated.

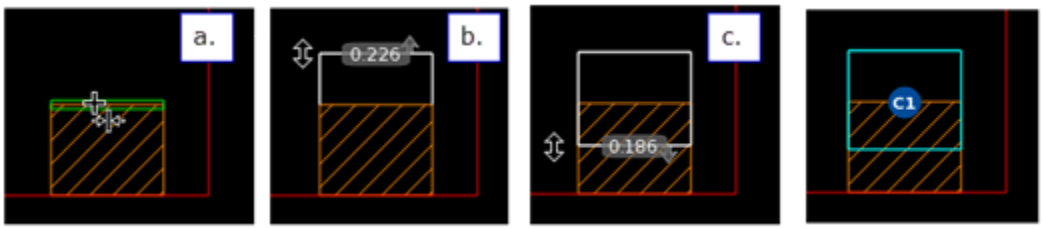
Video

The video “Pattern Editing: Create Single Edge and Edge to Edge Constraint” demonstrates how to add a single edge constraint.

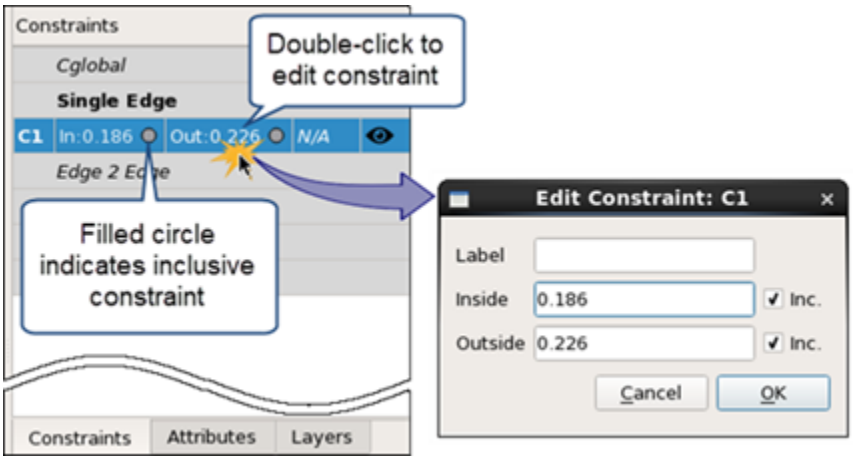


Procedure

1. View the **Constraints** tab in the Pattern Data pane to the right of the pattern canvas.
2. Click the **Add Single Edge Constraint** ( ) button above the pattern canvas.
3. Do the following to add the constraint:
   a. Click the edge to apply the constraint to. Valid edges are outlined in green.
   b. Drag the cursor up (for a horizontal edge) or to the right (for a vertical edge) and click at the allowed edge movement.
   c. Repeat the preceding step for the opposite direction.



The constraint is entered in the list of Single Edge constraints in the **Constraints** tab.



4. If needed, double-click the constraint in the constraint list to edit it.

   Constraints are created with an inclusive range by default. Uncheck the **Inc.** checkbox to exclude the range endpoint. Enter a label if desired.

   Tip

   You can edit the constraint range in the GUI. Click the **Edit Constraints** ( ) button and drag the constraint limits.

5. Choose **File > Save Library** to save changes to the pattern library.

Related Topics

Adding a Single Edge Constraint to Multiple Edges

Behavior of Constrained Edges at the Pattern Extent

Using Custom Constraint Labels

Adding a Single Edge Constraint to Multiple Edges

You can select multiple edges in a pattern and add the same single edge constraint to all of the selected edges.

Prerequisites

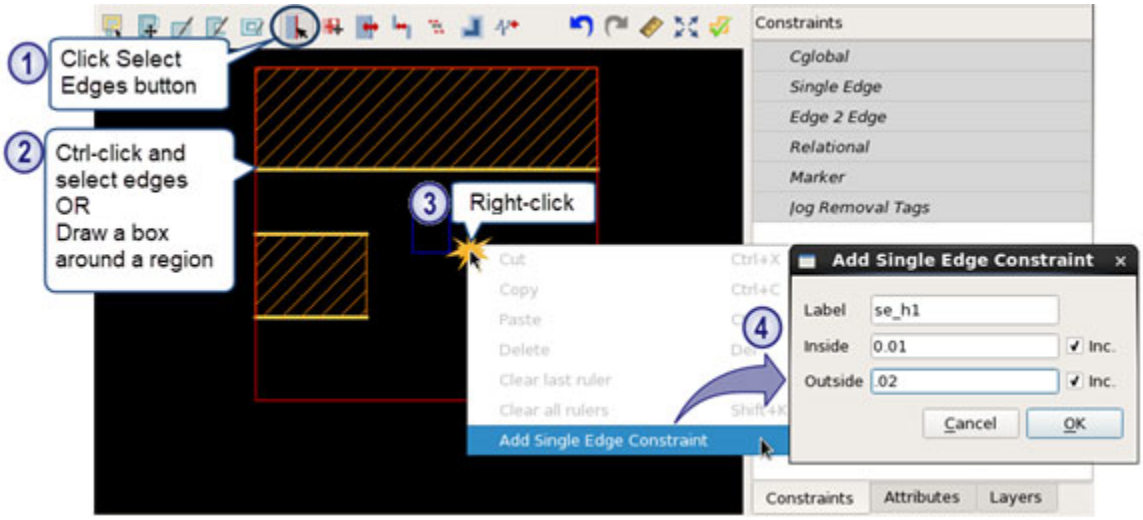
- A pattern library is open in the Calibre Pattern Matching GUI. See “Invoking the Calibre Pattern Matching GUI” on page 82.
- One of the following pattern types is selected:
  - A TEM pattern.
  - A fixed extent BCM pattern (BCM2) without a global constraint.
- The selected pattern is validated.

Procedure

1. Click the **Constraints** tab in the Pattern Data pane to the right of the pattern canvas.
2. Click the **Select Edges, Constraints** button ( ) and use one of these methods to select multiple edges:
   - Ctrl-click to select multiple edges.
   - Drag a box around a pattern region. All edges within the region are selected.

   The selected edges are highlighted.
3. Right-click within the pattern canvas and choose **Add Single Edge Constraint**.

This displays the Add Single Edge Constraint dialog box.



4. Fill in the parameters in the dialog box and click **OK**.

   The constraint is added to each of the valid selected edges.

Related Topics

Adding a Single Edge Constraint

Behavior of Constrained Edges at the Pattern Extent

Using Custom Constraint Labels

Adding a TEM Edge to Edge Constraint

Edge to edge constraints in TEM patterns specify the maximum and minimum separation between two parallel edges, or a fixed separation.

You can create edge to edge constraints between the edge combinations listed in the following table, where a moving pattern edge is a pattern edge with a single edge constraint.

Table 3-3. Possible TEM Edge to Edge Constraints

First Edge	Second Edge	Comments
Moving pattern edge	Moving or fixed pattern edge	Specifies a constraint on the edge to edge separation. See “Edge to Edge Constraints in TEM Patterns” on page 59 and “Non-Directional Edge to Edge Constraints in TEM Patterns” on page 61. A non-directional edge-to-edge constraint requires two moving pattern edges.

Table 3-3. Possible TEM Edge to Edge Constraints (cont.)

First Edge	Second Edge	Comments
Fixed pattern edge	Fixed pattern edge	Specifies a fixed separation between two fixed pattern edges. This type of edge to edge constraint can be used to specify a relation between two edge to edge separations. See “Adding a Relational Constraint” on page 152.

See these topics for other edge to edge constraints:

- “Adding a Marker Constraint” on page 148
- “Specifying a Dynamic Extent for a TEM Pattern” on page 150

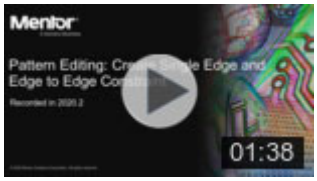
Prerequisites

- A pattern library is open in the Calibre Pattern Matching GUI. See “Invoking the Calibre Pattern Matching GUI” on page 82.
- A TEM pattern is selected in the pattern list.
- At least two parallel edges that meet one of the conditions in Table 3-3.

  See “Adding a Single Edge Constraint” on page 138 for instructions on adding a single edge constraint to a pattern edge.

Video

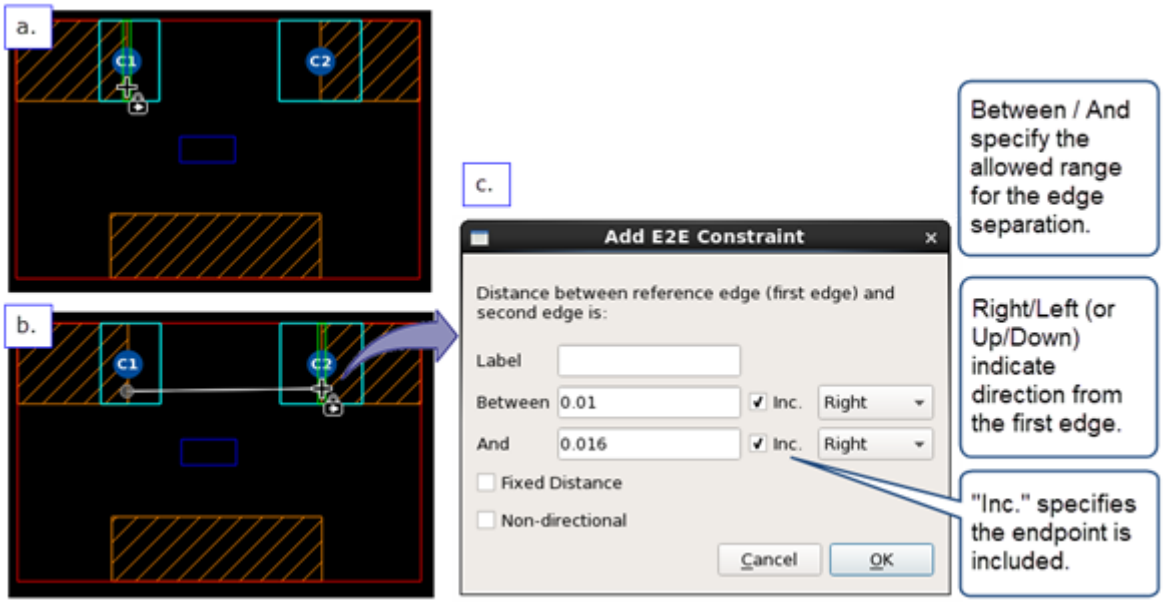
The video “Pattern Editing: Create Single Edge and Edge to Edge Constraint” demonstrates how to add an edge to edge constraint. The two pattern edges have single edge constraints.



Procedure

1. Click the **Constraints** tab in the pattern data pane (to the right of the pattern canvas) to view the constraint list.
2. Click the **Add Edge to Edge Constraint** ( ) button above the pattern canvas.
3. Do the following to add the constraint:
   a. Click the first edge of the edge to edge constraint. Valid edges are outlined in green.
   b. Click the second edge of the edge to edge constraint. Valid edges are outlined in green.

The Add E2E Constraint dialog box opens after clicking the second edge. Adding a constraint between two constrained pattern edges is shown in the following figure.



a. Fill in the “Add E2E Constraint” dialog box.

   The constraint is initialized as a directional, fixed constraint at the current edge separation.

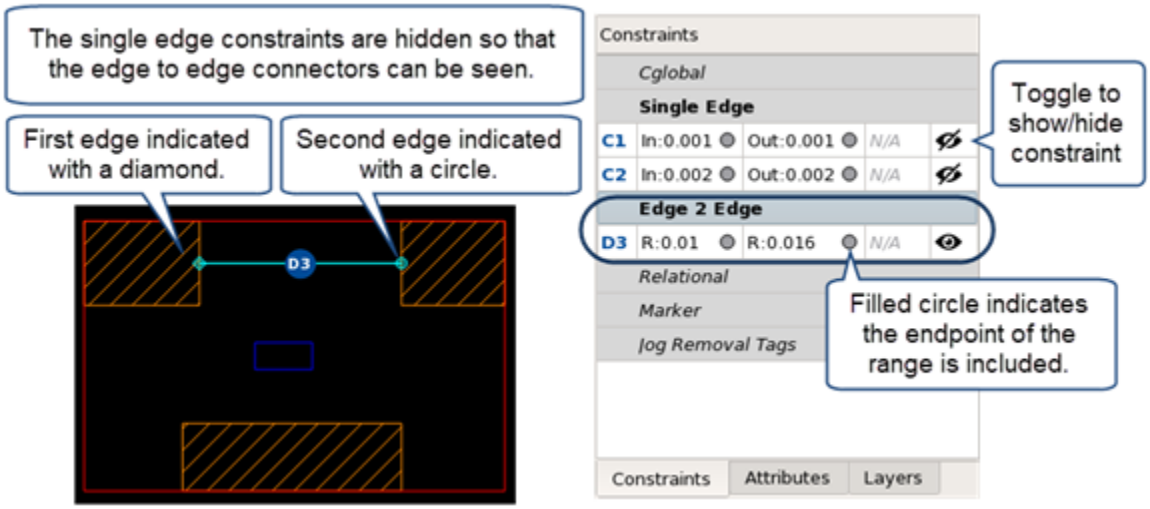
   - **Label** — An optional label for the constraint.
   - **Between** — The first value in the edge separation range.
   - **And** — The second value in the edge separation range.
   - **Right/Left** and **Up/Down** — Specifies the direction from the first edge.
   - **Fixed Distance** — Indicates that the separation between the two edges remains fixed as one or both of the edges move.
   - **Non-directional** — Indicates that the direction of the separation is not considered. The checkbox is unchecked by default, indicating a directional constraint.

     This selection must be made when the constraint is created, and cannot be changed later. Selecting Non-directional disables the Right/Left and Up/Down selections. See “Non-Directional Edge to Edge Constraints in TEM Patterns” on page 61 for applications of non-directional edge to edge constraints.
   - **Inc** — Indicates that the endpoint is included.

4. Click **OK** in the Add E2E Constraint dialog box.

The constraint is added to the pattern canvas and is listed in the Edge 2 Edge list on the **Constraints** tab.

Figure 3-6. TEM Edge to Edge Constraint in the GUI



Directional constraints (the default) are listed with a “D” and an index number. Non-directional constraints are listed with an “N” and an index number.

5. Choose **File > Save Library** to save changes to the pattern library.

Related Topics

Using Custom Constraint Labels

Edge to Edge Constraints in TEM Patterns

Adding a BCM1 (Accordion Extent) Edge to Edge Constraint

Edge to edge constraints in BCM1 patterns specify the allowed separation between two edges. The pattern extent changes as the edge separation changes. Constraints can only be added to single layer BCM patterns.

See “Edge to Edge Constraints in Accordion Extent BCM Patterns (BCM1)” on page 63 for general information.

Note

Single edge constraints are not used in BCM1 patterns.

Prerequisites

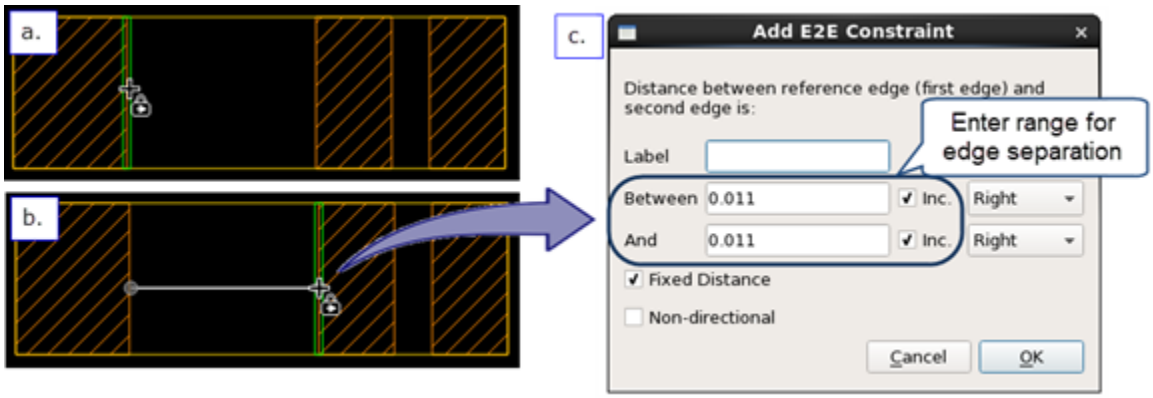
- A pattern library is open in the Calibre Pattern Matching GUI. See “Invoking the Calibre Pattern Matching GUI” on page 82.

- A single layer BCM1 pattern is selected in the pattern list.

Procedure

1. Click the **Constraints** tab in the pattern data pane (to the right of the pattern canvas) to view the constraint list.
2. Click the **Add Edge to Edge Constraint** ( ) button above the pattern canvas.
3. Do the following to add the constraint:
   a. Click the first edge of the edge to edge constraint. Valid edges are outlined in green.
   b. Click the second edge of the edge to edge constraint. Valid edges are outlined in green.

      The Add E2E Constraint dialog box opens after clicking the second edge.



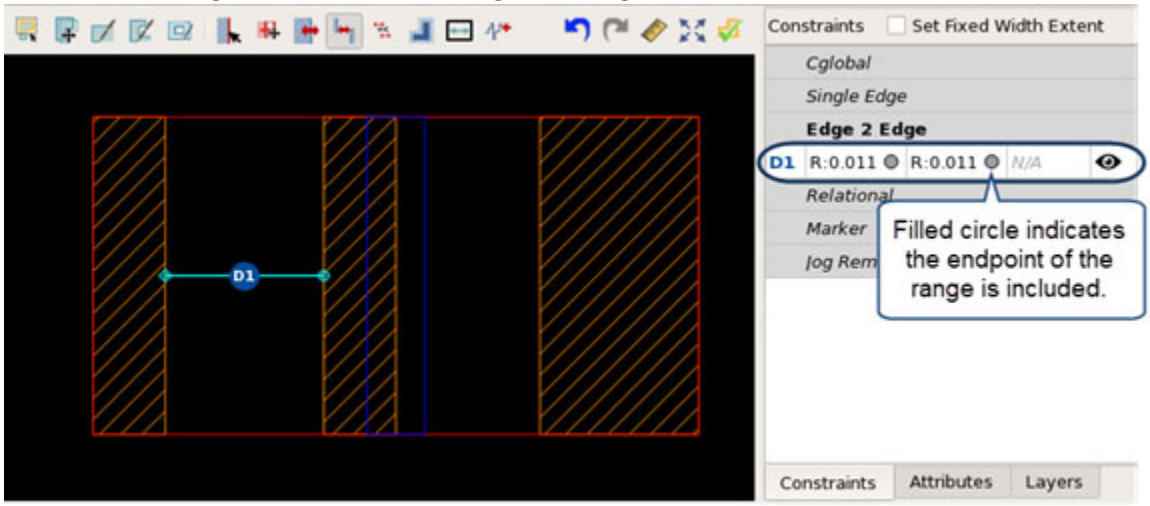
   a. Fill in the "Add E2E Constraint" dialog box.

      The constraint is initialized as a fixed constraint at the current edge separation.

      - **Label** — An optional label for the constraint.
      - **Between** — The first value in the edge separation range.
      - **And** — The second value in the edge separation range.
      - **Fixed Distance** — Indicates that the separation between the two edges remains fixed as one or both of the edges move.
      - **Inc** — Indicates that the endpoint is included.
4. Click **OK** in the Add E2E Constraint dialog box.

   The constraint is added to the pattern canvas and listed in the Edge 2 Edge list on the **Constraints** tab.

Figure 3-7. BCM1 Edge to Edge Constraint in the GUI



5. Choose **File > Save Library** to save changes to the pattern library.

Related Topics

Using Custom Constraint Labels

Adding a BCM2 (Fixed Extent) Edge to Edge Constraint

Adding a BCM2 (Fixed Extent) Edge to Edge Constraint

Edge to edge constraints in BCM2 patterns specify the allowed separation between two edges. Constraints can only be added to single layer BCM patterns.

See “Edge to Edge Constraints in Fixed Extent BCM Patterns (BCM2)” on page 62 for details.

Prerequisites

- A pattern library is open in the Calibre Pattern Matching GUI. See “Invoking the Calibre Pattern Matching GUI” on page 82.
- A single layer BCM2 pattern is selected in the pattern list.

  For most cases, the BCM2 pattern should have at least one single edge constraint to specify allowed edge movement. See “Adding a Single Edge Constraint” on page 138.

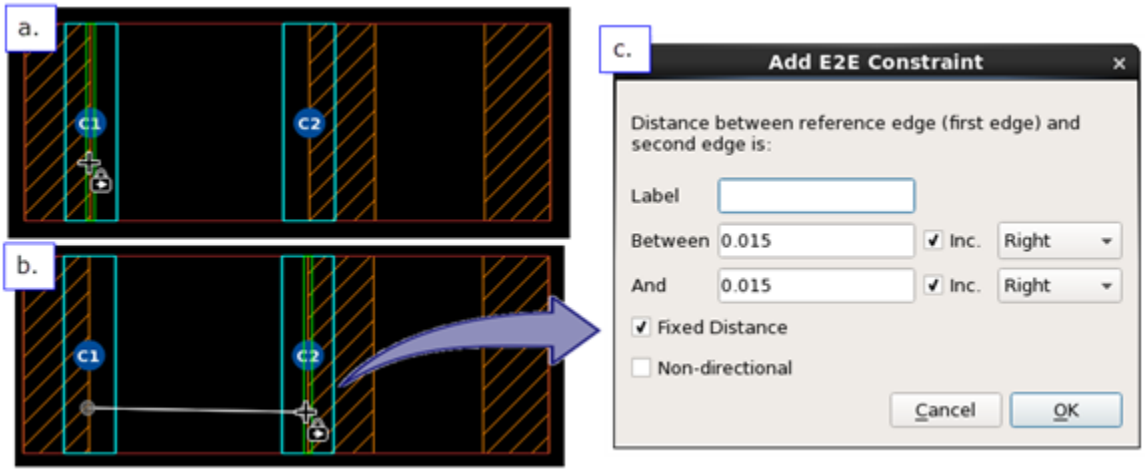
  Edge to edge constraints between two fixed edges can be used in forming relational constraints.

Procedure

1. Click the **Constraints** tab at the bottom of the panel to the right of the pattern canvas to view the constraint list.

2. Click the **Add Edge to Edge Constraint** ( ) button above the pattern canvas.
3. Do the following to add the constraint:
   a. Click the first edge of the edge to edge constraint. Valid edges are outlined in green.
   b. Click the second edge of the edge to edge constraint. Valid edges are outlined in green.

      The Add E2E Constraint dialog box opens after clicking the second edge.



   a. Fill in the “Add E2E Constraint” dialog box.

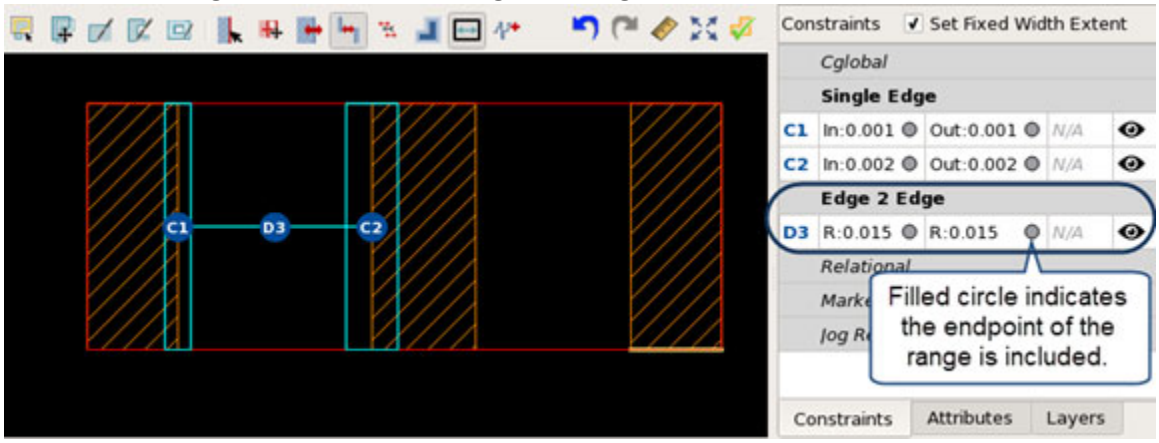
      The constraint is initialized as a fixed constraint at the current edge separation.

      - **Label** — An optional label for the constraint.
      - **Between** — The first value in the edge separation range.
      - **And** — The second value in the edge separation range.
      - **Fixed Distance** — Indicates that the separation between the two edges remains fixed as one or both of the edges move.
      - **Inc** — Indicates that the endpoint is included.

4. Click **OK** in the Add E2E Constraint dialog box.

   The constraint is added to the pattern canvas and listed in the Edge 2 Edge list on the **Constraints** tab.

Figure 3-8. BCM2 Edge to Edge Constraint in the GUI



5. Choose **File > Save Library** to save changes to the pattern library.

Related Topics

Using Custom Constraint Labels

Adding a BCM1 (Accordion Extent) Edge to Edge Constraint

Adding a Marker Constraint

Marker constraints specify the separation between a custom maker edge and a moving pattern edge. Marker constraints are only valid in TEM patterns.

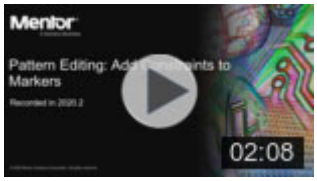
See “Marker Constraints” on page 64 for information and examples.

Prerequisites

- A pattern library is open in the Calibre Pattern Matching GUI. See “Invoking the Calibre Pattern Matching GUI” on page 82.
- A TEM pattern is selected in the pattern list. The pattern should have the following:
  - A custom marker. See “Adding Markers to a Pattern” on page 125.
  - At least one pattern edge with a single edge constraint. See “Adding a Single Edge Constraint” on page 138.

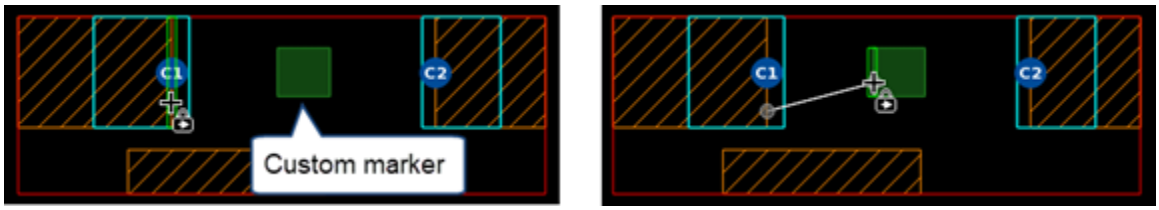
Video

For a demonstration, view the video “Pattern Editing: Add Constraints to Markers.”

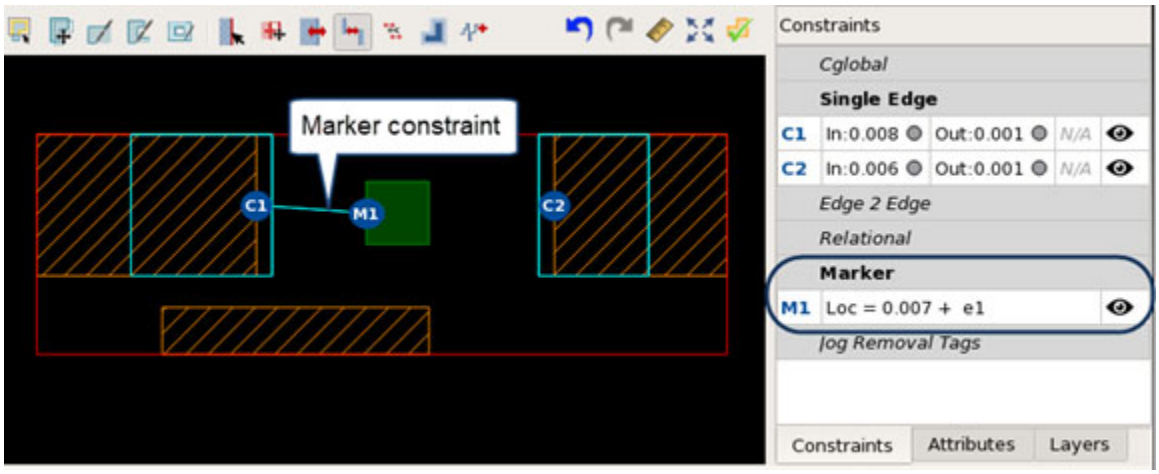


Procedure

1. Click the **Constraints** tab at the bottom of the panel to the right of the pattern canvas to view the constraint list.
2. Click the **Add Edge to Edge Constraint** ( ) above the pattern canvas.
3. Do the following to add the constraint:
   a. Click the pattern edge that determines the marker location. It should have a single edge constraint. Valid edges are outlined in green.
   b. Click the custom marker edge. Valid edges are outlined in green.



The constraint is created after clicking the marker edge.



The constraint parameters are automatically determined by the GUI so that the marker edge maintains a fixed separation from the moving pattern edge.

4. (Optional) To constrain the opposite marker edge so that the width of the marker remains constant, repeat the preceding step with the same pattern edge and the opposite marker edge.

   See Figure 2-35 on page 65 in the topic "Marker Constraints."

5. (Optional) Do the following to constrain the marker edge at a relative position between two moving pattern edges.

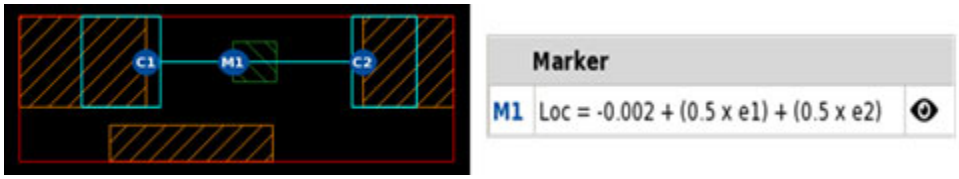
   These steps assume the marker edge from Step 3 is being constrained. Also see Figure 2-36 on page 66 in the topic "Marker Constraints."

a. Click the **Add Edge to Edge Constraint** ( ) button.

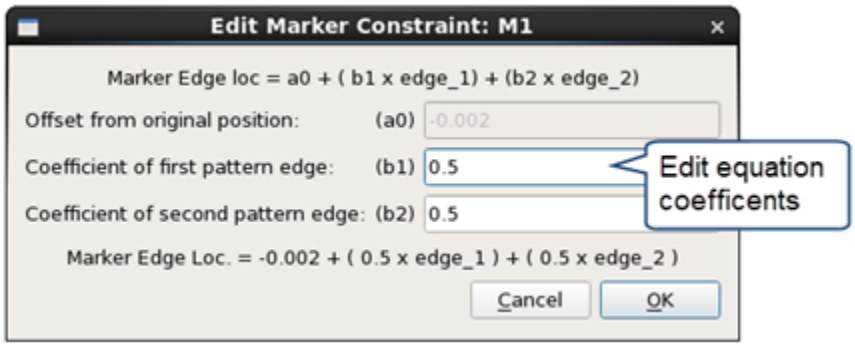
b. Click the custom marker edge from Step 3, which already has a marker constraint.

c. Click the second pattern edge that determines the marker position. The pattern edge should have a single edge constraint.

The existing marker constraint is updated to include the additional constraint to the second pattern edge. The constraint equation is determined automatically by the GUI according to the current edge positions.



d. (Optional) Double-click the constraint listing to edit the equation parameters. You can change the equation coefficients to give more or less weight to a pattern edge.



6. Choose **File > Save Library** to save changes to the pattern library.

Specifying a Dynamic Extent for a TEM Pattern

A dynamic extent is one that moves with a constrained pattern edge. You define a dynamic extent with the Calibre Pattern Matching GUI by adding an edge to edge constraint between a pattern edge and an edge of a custom extent.

See “Dynamic Extents” on page 39 for general information.

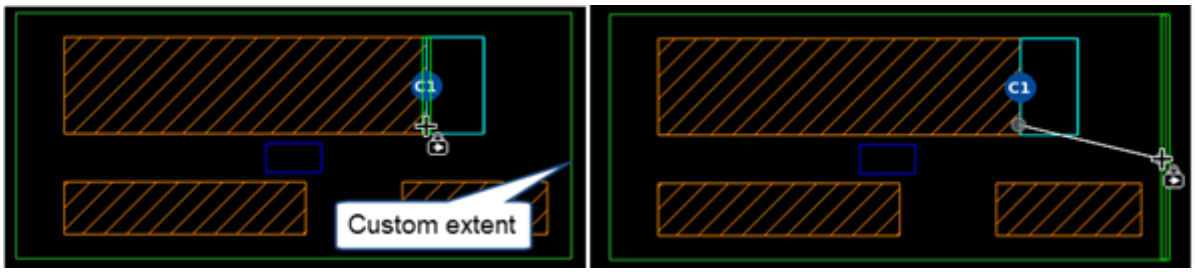
Prerequisites

- A pattern library is open in the Calibre Pattern Matching GUI. See “Invoking the Calibre Pattern Matching GUI” on page 82.

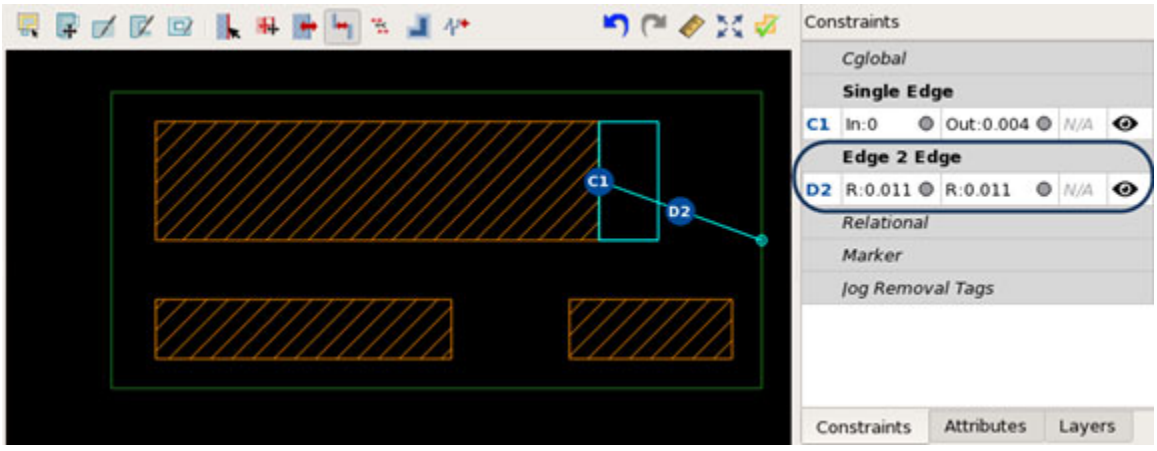
- A TEM pattern is selected in the pattern list. The TEM pattern should have the following:
  - A custom extent. See “Defining Custom and Per-Layer Extents for a TEM Pattern” on page 130.
  - At least one pattern edge with a single edge constraint. See “Adding a Single Edge Constraint” on page 138.

Procedure

1. Click the **Constraints** tab at the bottom of the panel to the right of the pattern canvas to view the constraint list.
2. Click the **Add Edge to Edge Constraint** ( ) button above the pattern canvas.
3. Do the following to add the constraint:
   a. Click a pattern edge with a single edge constraint. Valid edges are outlined in green.
   b. Click an edge on the custom extent that is parallel to the pattern edge. Valid edges are outlined in green.



The constraint is entered in the “Edge 2 Edge” constraint list as a fixed distance constraint.



Note

If a dynamic extent is coincident with a pattern edge, the GUI prints a warning in the transcript pane that the edge is within the extent movement range and can cause possible matches to be missed. See "Behavior of Constrained Edges at the Pattern Extent" on page 68.

a. (Optional) Double-click the constraint listing to add a label. This is the only parameter that can be edited.

4. Choose **File > Save Library** to save changes to the pattern library.

Adding a Relational Constraint

Relational constraints specify a relation between two edge to edge measurements. For example, two polygons may have variable widths, but require the same width for a successful pattern match. You can specify relational constraints in both TEM and BCM patterns.

See "Relational Constraints" on page 66 for examples and explanation.

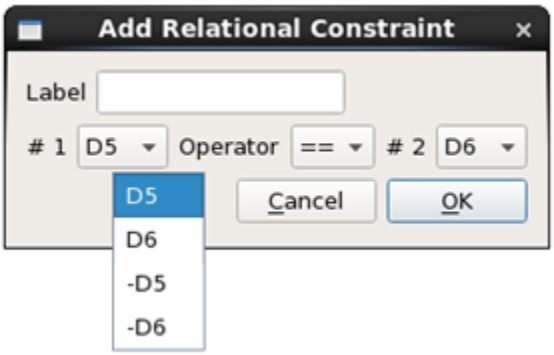
Prerequisites

- A pattern library is open in the Calibre Pattern Matching GUI. See "Invoking the Calibre Pattern Matching GUI" on page 82.
- A pattern with two edge to edge constraints is selected in the pattern list. This can be a TEM pattern or a single layer BCM pattern. See these topics:
  - "Adding a TEM Edge to Edge Constraint" on page 141
  - "Adding a BCM1 (Accordion Extent) Edge to Edge Constraint" on page 144
  - "Adding a BCM2 (Fixed Extent) Edge to Edge Constraint" on page 146

Procedure

1. Click the **Constraints** tab at the bottom of the panel to the right of the pattern canvas to view the constraint list.

2. Click the **Add Relational Constraint** ( ) button above the pattern canvas. This opens a dialog box.



3. Fill in the dialog box.

   Select the first edge to edge constraint, the relational operator, and the second edge to edge constraint. A negative sign in front of the edge to edge constraint indicates that the negative of the edge to edge measurement is used.

   Edge separation is measured from the first edge to the second edge in an edge to edge constraint.

4. Choose **File > Save Library** to save changes to the pattern library.

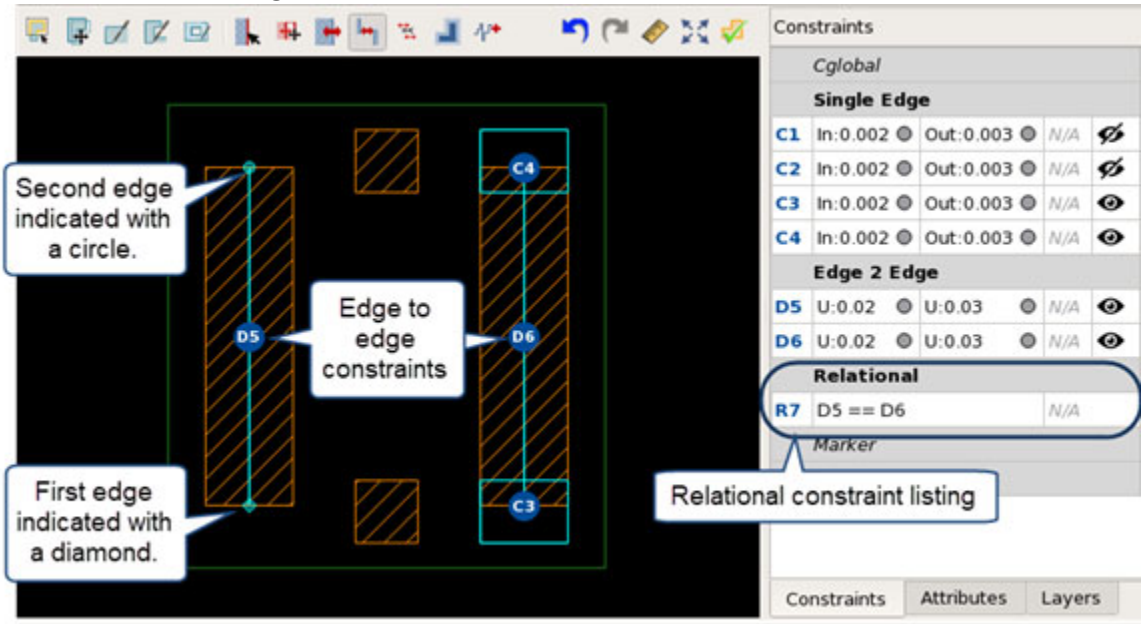
Results

A TEM pattern with a relational constraint is shown in the following figure. The relational constraint is listed in the “Relational” section of the constraint list, but is not indicated in the pattern canvas.

Tip

Hide the single edge constraints in order to see the end connectors of the edge to edge constraint.

Figure 3-9. Relational Constraint in the GUI



Related Topics

Using Custom Constraint Labels

Constraints

Relational Constraints

Adding Constraints to a Multilayer BCM1 (Accordion Extent) Pattern

Edge to edge constraints are supported in multilayer BCM1 patterns, with some special requirements. In order to provide unambiguous matching, if constraints are applied between edges on the same layer (intra-layer constraints), that layer also needs to have at least one constraint applied to another layer (an inter-layer constraint).

Prerequisites

- A multilayer BCM1 pattern is selected in the pattern list.

Procedure

1. Use the **Add Edge to Edge Constraint** ( ) button to add constraints as needed. See “Adding a BCM1 (Accordion Extent) Edge to Edge Constraint” on page 144.
2. View the transcript pane for messages. If the transcript pane is not visible, choose **Windows > Transcript**.

If the following message is displayed, there is a layer with intra-layer constraints that does not have a required constraint to one of the other layers.

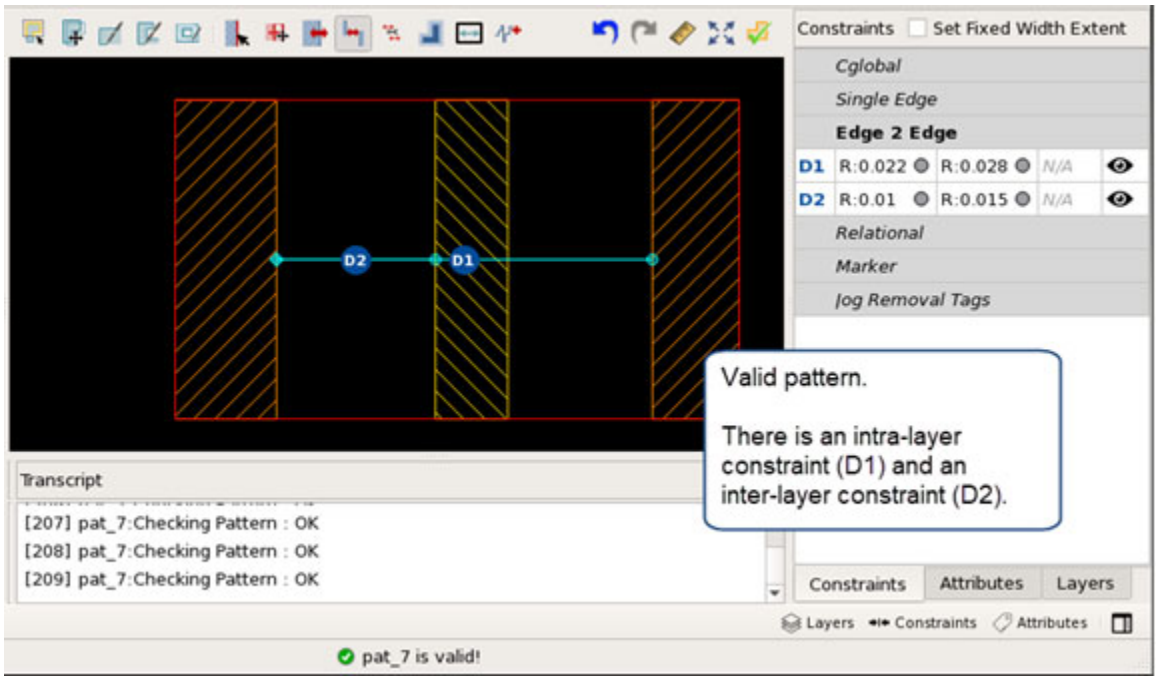
```
pat_<num>: Multi-layer BCM pattern needs all layers with intra-layer
constraints to be tied together by inter-layer constraints.
```

To fix this error, add an inter-layer constraint and check the transcript again.

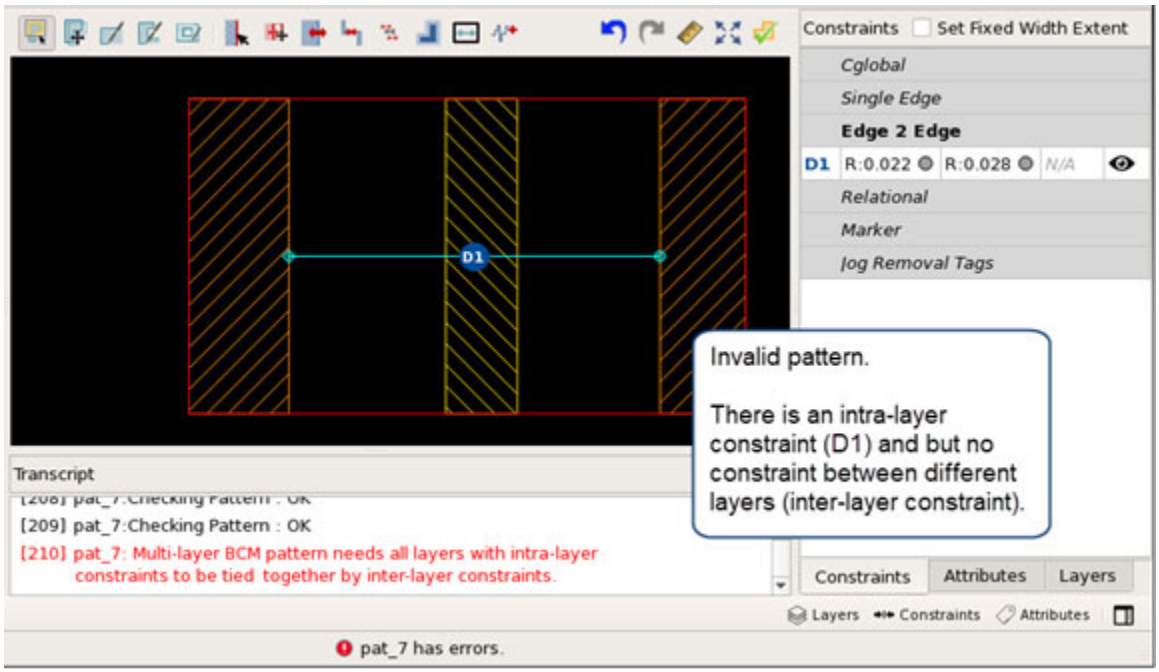
3. Choose **File > Save Library** to save changes to the pattern library.

Examples

This is an example of a valid multilayer BCM1 pattern with constraints. The D1 constraint is between two edges on the layer_1, and the D2 constraint is between an edge on layer_1 and an edge on layer_2.



This is an example of an invalid pattern. There is a constraint between layer_1 edges, but no constraint from a layer_1 edge to a layer_2 edge.



Related Topics

Edge to Edge Constraints in Accordion Extent BCM Patterns (BCM1)

Using Custom Constraint Labels

You can specify custom labels for constraints using the Calibre Pattern Matching GUI.

Prerequisites

- A pattern library is open in the Calibre Pattern Matching GUI. See “Invoking the Calibre Pattern Matching GUI” on page 82.
- A pattern with constraints is selected in the pattern list.

Procedure

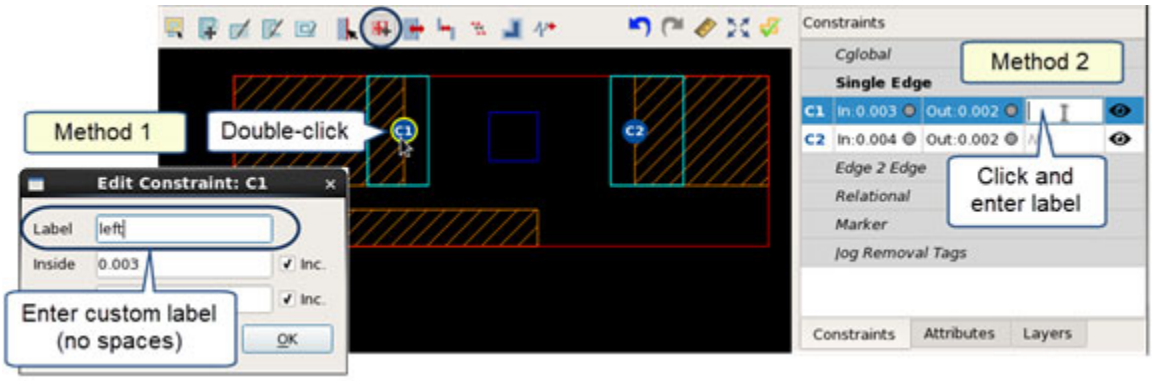
1. Select a pattern in the pattern list.
2. Click the **Constraints** tab at the bottom of the pane to the right of the pattern canvas to view the constraint list.

3. Add a constraint label:

   a. Do one of the following to add a constraint label:

      - **Method 1** — Click the **Edit Constraints** ( ) button, then double-click the circled constraint label in the pattern canvas. Enter the label name in the Edit Constraint dialog box.
      - **Method 2** — On the **Constraints** tab, click in the next to last column of the constraint listing and enter the label.

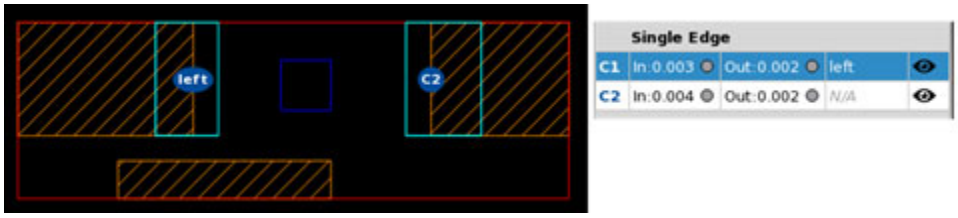
      Spaces are not allowed in the label.



4. Display the custom constraint label:

   a. Choose **Options > Preferences > Editor Preferences**.

   b. Check “Display constraint labels in canvas.”



Working with Properties and Attributes

You can add properties, keys, comments, and attributes to the patterns in a library using the Calibre Pattern Matching GUI.

Adding Custom and Output Properties and to a Library

You can add properties to a library using the Calibre Pattern Matching GUI. There are two types of properties: custom properties and output properties. All properties are automatically attached to the output markers for a matched result during a pattern matching run.

Property Type	Description
Custom property	A unique value can be assigned in each pattern. All patterns must have the same set of custom properties, but the value is assigned per pattern.
Output property	The output properties match_orient, match_rotation, and match_type are added at the library level, and the value is assigned by the tool when a pattern match is made.

For a general discussion of properties, see “Properties” on page 70.

Prerequisites

- A pattern library is open in the Calibre Pattern Matching GUI. See “Invoking the Calibre Pattern Matching GUI” on page 82.

Procedure

1. Choose **File > Library Attributes** to open the Library dialog box.
2. Choose the **Properties** category.
3. (Optional) To add output properties, choose the **Output Properties** tab and select options.
    - **Set match_orient Property** — Specifies the orientation of the matched result.

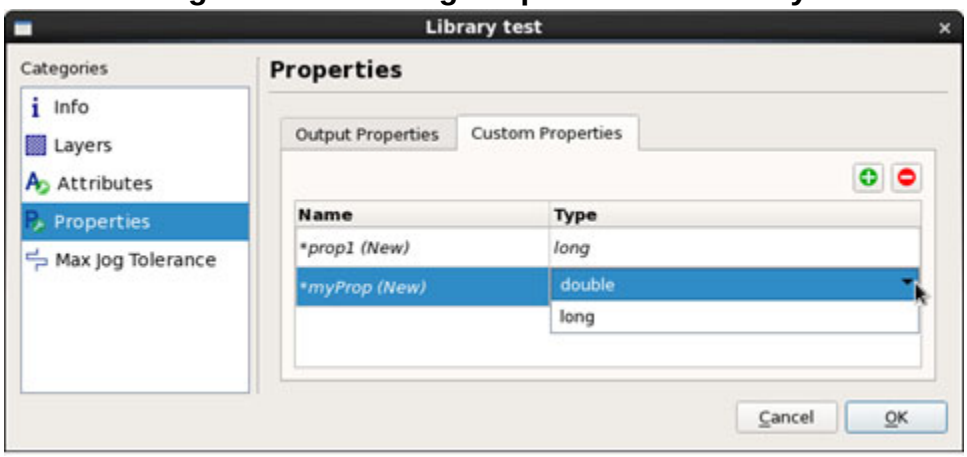
- **Set match_rotation Property** — Specifies the rotation of the matched result.
- **Set match_type Property** — Specifies whether constraints were used when making the match.

See "Reserved Property Names" on page 71. The properties match_orient and match_rotation are mutually exclusive.

4. (Optional) Click the **Custom Properties** tab to add custom properties.

   Click the ⊕ button to add a property. Click in the property name to change it. Select the property type with the dropdown list.

Figure 3-10. Adding Properties to a Library



   This step adds the custom property to all patterns in the library without an assigned value.

5. Click **OK**.

   A message is displayed to confirm that you want to save the library options. Click **Yes** to proceed. If a pattern is loaded, it is closed.

6. To assign a value to a custom property, see "Defining Custom Property Values in a Pattern" on page 159.

7. (Optional) To delete a property from a library, select the name in the Library dialog box and click the delete button (⊖). Deleting a property removes it from all patterns in the library.

Defining Custom Property Values in a Pattern

You can assign property values to a pattern using the Calibre Pattern Matching GUI. Custom properties have a value defined in the pattern. Properties are automatically attached to the output markers for a matched result during a pattern matching run.

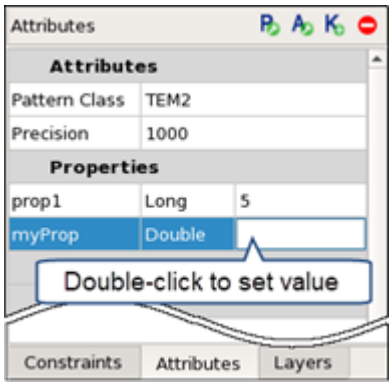
You can specify property values for a single pattern or multiple patterns. When specifying values for multiple patterns, you have the option to assign a value that is incremented automatically for each pattern.

Prerequisites

- A pattern library is open in the Calibre Pattern Matching GUI. See "Invoking the Calibre Pattern Matching GUI" on page 82.
- (Optional) Add properties at the library level. See "Adding Custom and Output Properties and to a Library" on page 158. (The output properties match_orient, match_rotation, and match_type can only be added at the library level, and do *not* have a value assigned in the pattern.)

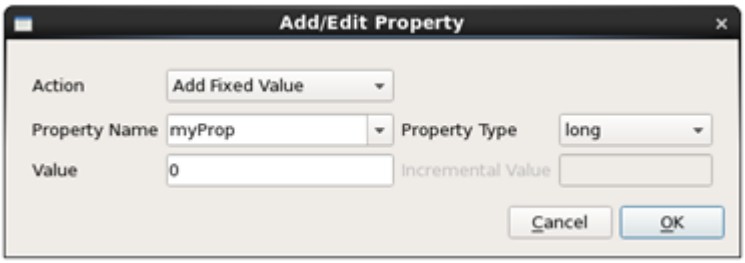
Procedure

1. Select one or more patterns in the pattern list.
2. Do one of the following:

Single Pattern 1. Click the **Attributes** tab at the bottom of the right panel of the GUI. 2. In the Properties section, double-click in the last column of the row containing the property name and enter a value. **Tip:** If the desired property name is not present, click the **(Manage Properties)** button to open the Library dialog box and add the property.	

Multiple Patterns

1. Right-click on the selected patterns and choose **Edit Selected Patterns > Add/Edit Property**.
2. Choose one of the following in the Action dropdown list:
   - **Add Fixed Value** — Add the same value to all selected patterns.
   - **Add Incremental Value** — Add a value that is incremented for each selected pattern. Choose the starting value and the increment.
   - **Clear Value** — Make the value unassigned for the selected patterns.
3. Choose an existing property name, or type in a new name.
4. Assign the value, and click **OK**.



If you add a new property, it is added to all patterns. The value is assigned for the selected patterns, but remains unassigned for unselected patterns.

3. Click the **Save Library** button ( ) to save the library.

Related Topics

Properties

Adding Custom Attributes to a Library

You can add custom attributes to a library using the Calibre Pattern Matching GUI. There are two types of custom attributes: custom library attributes and custom pattern attributes. Attributes are only saved in the pattern library—they are not used in a pattern matching run and are not attached to pattern matching results.

Attribute Type	Description
Custom library attribute	One value is assigned at the library level.
Custom pattern attribute	A unique value can be assigned in each pattern. All patterns must have the same set of attributes, but the value is assigned per pattern.

For a general description of attributes, see “Attributes” on page 73.

Prerequisites

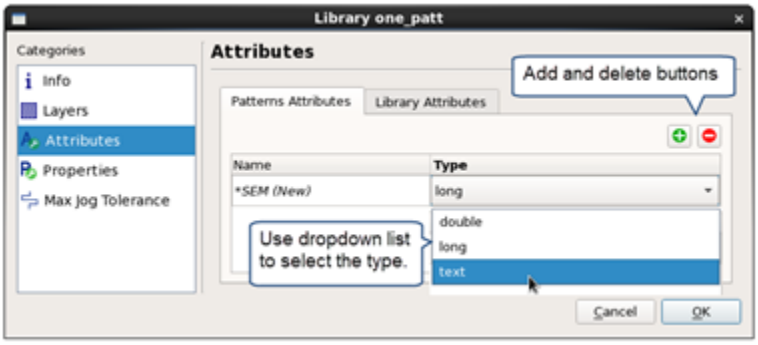
- A pattern library is open in the Calibre Pattern Matching GUI. See “Invoking the Calibre Pattern Matching GUI” on page 82.

Procedure

1. Choose **File > Library Attributes** to open the Library dialog box.
2. Choose the **Attributes** category.
3. (Optional) Click the **Pattern Attributes** tab to add custom pattern attributes.
   a. Click the ⊕ button to add an attribute.
   b. Click in the attribute name to change it.
   c. Select the attribute type with the dropdown list. You cannot change the type after you save the library.

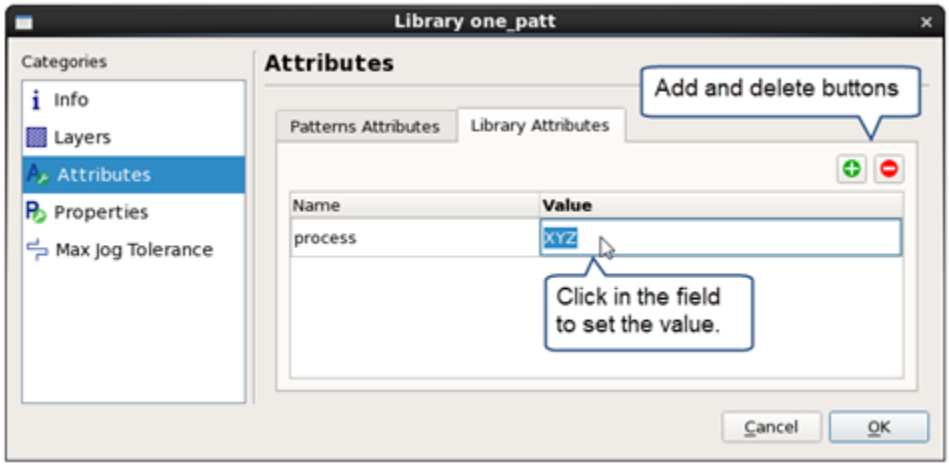
      The attribute is added to all patterns in the library, without an assigned value. Set the value in the pattern view using the **Attributes** tab.

Figure 3-11. Adding a Custom Pattern Attribute to a Library



4. (Optional) Click the **Library Attributes** tab to add custom library attributes.
   a. Click the ⊕ button to add an attribute.
   b. Click in the attribute name to change it.
   c. Click in the Value field to assign a value.

Figure 3-12. Adding a Custom Library Attribute



5. Click **OK**.

   A message is displayed to confirm that you want to save the library options. Click **Yes** to proceed. If a pattern is loaded, it is closed.

6. To assign a value to a custom attribute, see "Defining Custom Attribute Values in a Pattern" on page 163.

7. (Optional) To delete an attribute from a library, select the name in the Library dialog box and click the delete button ( ). Deleting an attribute removes it from all patterns in the library.

Defining Custom Attribute Values in a Pattern

You can add custom attributes to a pattern using the Calibre Pattern Matching GUI. Custom attributes can be string or numeric values. Unlike properties, custom attributes are only saved in the pattern library. Custom attributes are not used in a pattern matching run and they are not attached to pattern matching results. You define the value of a custom pattern attributes in the pattern view of the GUI.

You can specify attribute values for a single pattern or multiple patterns. When specifying numeric values for multiple patterns, you have the option to assign a value that is incremented automatically for each pattern.

Prerequisites

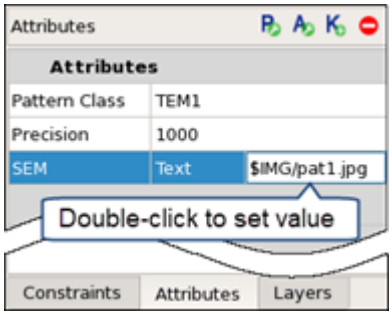
- A pattern library is open in the Calibre Pattern Matching GUI. See "Invoking the Calibre Pattern Matching GUI" on page 82.
- (Optional) Add attributes at the library level. See "Adding Custom Attributes to a Library" on page 161.

Procedure

1. Select one or more patterns in the pattern list.
2. Do one of the following:

Single Pattern

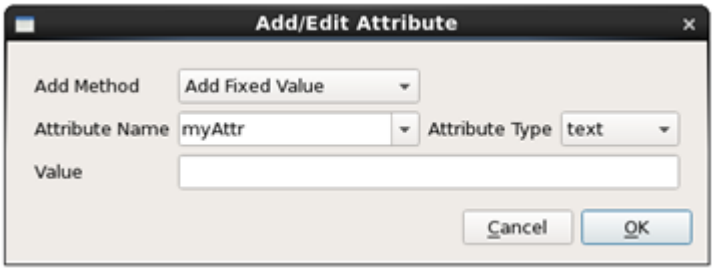
1. Click the **Attributes** tab at the bottom of the right panel of the GUI.
2. In the Attributes section, double-click in the last column of the row containing the attribute name and enter a value. The text entry box enforces the type assignment.



Tip: If the desired attribute name is not present, click the (**Manage Attributes**) button to open the Library dialog box and add the attribute.

Multiple Patterns

1. Right-click on the selected patterns and choose **Edit Selected Patterns > Add/Edit User Attribute**.
2. Choose one of the following in the Add Method dropdown list:
   - **Add Fixed Value** — Add the same value to all selected patterns.
   - **Add Incremental Value** — Add a value that is incremented for each selected pattern. Choose the starting value and the increment. Only valid for numeric attributes.
   - **Clear Value** — Make the value unassigned for the selected patterns.
3. Choose an existing attribute name, or type in a new name.
4. Assign the value, and click **OK**.



If you add a new attribute, it is added to all patterns. The value is assigned for the selected patterns, but remains unassigned for unselected patterns.

3. Click the **Save Library** button ( ) to save the library.

Related Topics

Adding Custom and Output Properties and to a Library

pmatch::add_int_attr [Calibre Pattern Matching Reference Manual]

pmatch::add_text_attribute [Calibre Pattern Matching Reference Manual]

Attaching Keys to a Pattern

You can attach keys to patterns using the Calibre Pattern Matching GUI.

If key names are specified in the CMACRO option strings, then only patterns tagged with specified keys are used for matching.

Prerequisites

- A pattern library is open in the Calibre Pattern Matching GUI. See "Invoking the Calibre Pattern Matching GUI" on page 82.

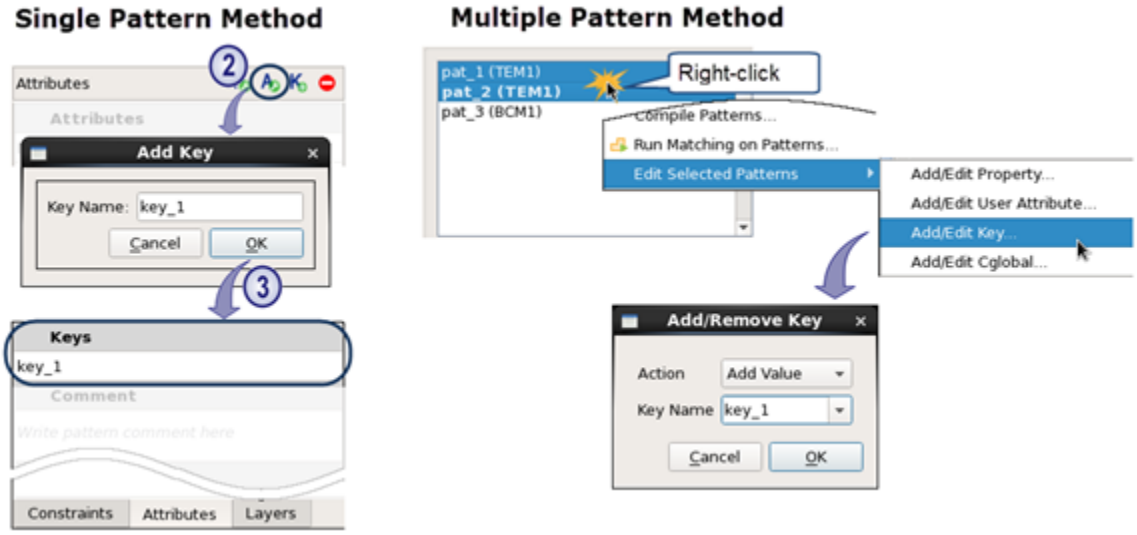
Procedure

1. Select one or more patterns in the pattern list.
2. Do one of the following:

For …	Do this …
Single pattern	1. Click the **Attributes** tab at the bottom of the left panel of the GUI. 2. Click the **Add Key** button () to open the Add Key dialog box.
Multiple patterns	Right-click on the selected patterns and choose **Edit Selected Patterns > Add/Edit Key** to open the Add/Remove Key dialog box.

3. Specify the key name in the dialog box and click **OK**.

Figure 3-13. Adding a Key to a Pattern



4. (Optional) To delete a key, do one of the following:
   - **Single pattern** — Select the key on the **Attributes** tab and press the Delete keyboard key.
   - **Multiple patterns** — Open the Add/Remove Key dialog box, select Remove Key as the Action, and choose the key name from the drop down list.
5. Choose **File > Save Library** to save changes to the pattern library.

Related Topics

Pattern Keys

CMACRO for Calibre Pattern Matching

Defining Orientation for a Pattern

You can set the pattern orientation attribute to specify which patterns orientations to match. The orientation is set on the **Attributes** tab of the Calibre Pattern Matching GUI. All orientations are selected by default.

Note

If no orientations are specified, a warning is generated, and the pattern is found in all orientations.

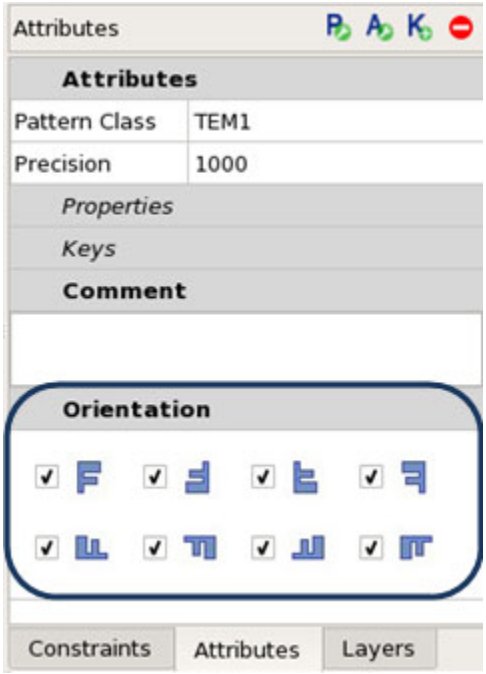
Prerequisites

- A pattern library is open in the Calibre Pattern Matching GUI. See “Invoking the Calibre Pattern Matching GUI” on page 82.

Procedure

1. Select a pattern in the pattern list.
2. Click the **Attributes** tab at the bottom of the right panel of the GUI.
3. In the Orientation area, select the pattern orientations that you want your pattern to match. All orientations are selected in the following figure.

Figure 3-14. Orientation Attribute in the PM GUI



4. Save the pattern library.

Related Topics

Pattern Orientation

pmatch::set_pattern_orient [Calibre Pattern Matching Reference Manual]

pmatch::add_pattern_orient [Calibre Pattern Matching Reference Manual]

Adding Comments to a Pattern

You can attach comments to the patterns in library using the Calibre Pattern Matching GUI.

Pattern comments are optional pattern attributes that allow you to add and save text along with your pattern. Pattern comments are useful for passing information about a pattern to other pattern users. A pattern comment can be made up of any keyboard characters.

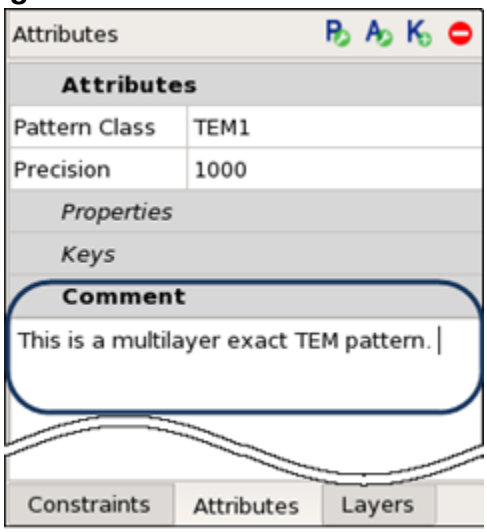
Prerequisites

- A pattern library is open in the Calibre Pattern Matching GUI. See “Invoking the Calibre Pattern Matching GUI” on page 82.

Procedure

1. Select a pattern in the pattern list.
2. Click the **Attributes** tab at the bottom of the right panel of the GUI.
3. Type the comment in the text entry box for the Comment area.

Figure 3-15. Add Pattern Comment



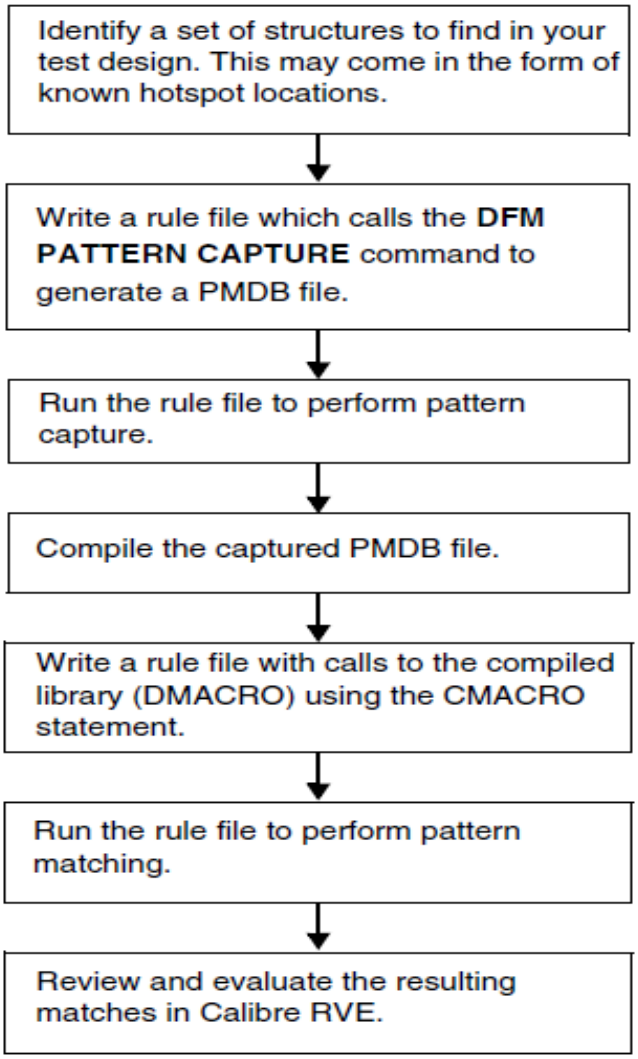
4. Click the **Save Library** button ( ) to save the library.

Chapter 4
Pattern Matching Integration with SVRF

You can capture and export pattern library files in batch mode using the DFM Pattern Capture SVRF command.

Patterns are captured in a layout design to generate a pattern matching database (PMDB). The following figure shows the basic flow for generating and executing a pattern generation run with SVRF.

Figure 4-1. Pattern Library Generation Flow



Pattern Matching Invocation

Calibre Pattern Matching runs as a set of SVRF operations enclosed in an SVRF DMACRO statement. The pattern matching SVRF DMACRO rule file is included in a standard Calibre rule file, which is then run with the standard Calibre invocation options.

The Calibre Pattern Matching invocation options are shown here.

```
calibre -drc [-hier]
        [ -turbo [number_of_processors][-turbo_all]
          [ { -remote host[,host]...  | -remotefile filename
            | -remotecommand filename count }
            [ -remotedata [-recoverremote | -recoveroff]] ]
          [-hyper [ remote ] ]
        ]
        [ -nowait | -wait n ]
        [ -lmconfig licensing_config_filename ]
        rule_file_name
```

Calibre nmDRC and Calibre nmDRC-H carry out pattern matching in either flat (calibre -drc) or hierarchical (calibre -drc -hier) mode to find your patterns.

The following invocation options are not supported with Calibre Pattern Matching:

-cb, -E, -fx, -tvfarg

Complete details on applicable invocation options are found in the section “Calibre nmDRC and Calibre nmDRC-H Command Line” in the *Calibre Verification User’s Manual*.

The following commands and options support MT and Calibre® MTflex™:

- DFM Pattern Capture

- DFM Pattern Classify
- CMACRO; see "CMACRO for Calibre Pattern Matching" on page 177.

Additionally, Pattern Matching CMACRO supports hyperscaling pseudo HDB execution.

Invocation Examples

The following examples show how to run both flat and hierarchical modes of pattern matching:

```
calibre -drc pmatch_rules

calibre -drc -hier /user/scratch1/pmatch/pmatch_rules
```

These show examples for MT and MTflex modes, respectively:

```
calibre -drc -hier -turbo pmatch_rules

calibre -drc -hier -turbo -remote host1,host2,host3 rules
```

These modes assume you have a local host machine with multiple CPUs. If Calibre MTflex mode is used, your remote hosts must be accessible from your network and must have multiple CPUs.

Pattern Matching also supports hyperscaling (-hyper), allowing Calibre to execute CMACROs in parallel on psuedo HDBs.

```
calibre -drc -hier -turbo -hyper pmatch_rules
```

Pattern Matching SVRF Commands

You can add SVRF commands to your pattern matching rules file to generate output specific to pattern matching.

SVRF Command	Description
DFM Pattern Capture	Captures patterns based on specified lithographic hotspots or based on a specified mask layer. You can optionally save the classified patterns to a pattern library. See "Generating and Merging PMDB Library Files" on page 172 for an example.
DFM Pattern Classify	Identifies and classifies patterns based on interactions between polygons on target layers and a seed or pattern mask layer. The seed or pattern mask layer polygons are written to the output layer with an attached pattern class property. You can optionally save the classified patterns to a pattern library.

SVRF Command	Description
DFM Pattern Match Report	Generates a comma-separated values (CSV) file which summarizes the matching statistics from a pattern matching run.

Generating and Merging PMDB Library Files

You can generate a pattern matching database (PMDB) for finding possible hotspots in your designs. This is done using an SVRF rule file that contains the DFM Pattern Capture statement.

This procedure illustrates capturing patterns based on a single layer (M1). It also illustrates how to merge PMDB files into one file. You can apply this procedure to patterns with more than one layer, as necessary.

Prerequisites

- A layout file with hotspots marked on a separate hotspot layer for pattern generation.
- Licenses to generate a PMDB. For more information on licensing for Calibre Pattern Matching, refer to the *Calibre Administrator's Guide*.

Procedure

1. Review the following rule file. You can use it as a template for creating an output PMDB from an existing layout with marked hotspots.

```
LAYOUT PATH "./layout.gds"
LAYOUT PRIMARY "*"
LAYOUT SYSTEM GDSII
DRC RESULTS DATABASE "drc_results.db"
DRC MAXIMUM RESULTS ALL
DFM DATABASE "capture.rdb" OVERWRITE

PRECISION 1000

LAYER M1 1
LAYER PAT_HOTSPOT 1001
LAYER PAT_BOUNDARY 1002

// capture hotspots on layer M1 based
// on the hotspot layer PAT_HOTSPOT
capture_hotspot = DFM PATTERN CAPTURE
                  layer_target M1
                  layer_hotspot PAT_HOTSPOT
                  max_search 0.5
                  max_length 1.0
                  pattern_halo 0.5
                  outfile M1_hotspot_pats.pmdb
                  pattern_name_prefix M1_hotspot_

hotspot_patterns { COPY capture_hotspot }
```

```
// capture the pattern boundaries based
// on the hotspot layer PAT_BOUNDARY
capture_boundary = DFM PATTERN CAPTURE
                   layer_target M1
                   layer_hotspot PAT_BOUNDARY
                   max_search 0
                   max_length 1.0
                   outfile M1_boundary_pats.pmdb
                   pattern_name_prefix M1_boundary_

boundary_patterns { COPY capture_boundary }
```

2. At the command line, invoke the pdl_lib_mgr utility to merge the input hotspot PMDBs into one PMDB. Be sure that you are in the directory location where the input files reside.

   ```
   pdl_lib_mgr merge input M1_hotspots_pats.pmdb input \
               M1_boundary_pats.pmdb output M1_all_pats.pmdb
   ```

   A merged final library called *M1_all_pats.pmdb* is generated in the current directory. This merged library is now ready for a DMACRO conversion.

   Alternatively, the DMACRO conversion can be automatically called immediately after the PMDB library merge with the following invocation:

   ```
   pdl_lib_mgr merge input M1_hotspots_pats.pmdb input \
               M1_boundary_pats.pmdb output M1_all_pats.pmdb \
               compile_output dmacro.svrf
   ```

Results

An output PMDB file, *M1_all_pats.pmdb*, ready for DMACRO conversion or a DMACRO for inclusion in the main rule file run.

Related Topics

Compiling a Pattern Library With the compile Utility

Using a DMACRO in a Rule File for a Pattern Matching Run

Compiling a Pattern Library With the compile Utility

You can create an SVRF DMACRO from a pattern matching database (PMDB) file with the pdl_lib_mgr compile utility.

Prerequisites

- A pattern library (PMDB) file.

Procedure

At the command line, use the following invocation in the directory where the PMDB file resides:

```
pdl_lib_mgr compile input my_patterns.pmdb output dmacro.svrf
```

See pdl_lib_mgr compile for complete syntax. Some useful arguments are:

- enable_runtime_options — Necessary if you want to specify runtime options in the CMACRO options string.
- gen_count_layer — Adds count markers.
- marker and gen_all_markers — Adds compile-time markers. If the input pattern library does not have the same markers for all patterns in the library, you must specify one or both of these arguments.

Results

An encrypted DMACRO file called *dmacro.svrf* is generated. The following example shows the DMACRO file with two input target layers and one output marker.

```
DMACRO pm:my_patterns.pmdb in_layer1 in_layer2
   options_string
   out_Marker {
   Encrypted file ...
}
```

Note

The DMACRO format changed in releases 2019.2 and 2021.2. DMACRO files created with Calibre 2021.2 cannot be used by earlier versions of Calibre. DMACRO files created from 2019.2 to 2021.1 cannot be used in Calibre versions 2019.1 and earlier.

DMACRO files created prior to 2021.2 can be used in later versions of Calibre.

Related Topics

compile

Count Markers

Using a DMACRO in a Rule File for a Pattern Matching Run

Compiling a Pattern Library with the Calibre Pattern Matching GUI

DMACRO File for Calibre Pattern Matching

Calibre Pattern Matching uses a DMACRO definition to include pattern libraries in the SVRF rule file. The DMACRO is compiled from a pattern library (PMDB).

You can compile a pattern library (PMDB) with two different methods:

To compile a library …	See …
From the Calibre Pattern Matching GUI	"Compiling a Pattern Library with the Calibre Pattern Matching GUI" on page 105
From the command line	The compile command for the pdl_lib_mgr utility

Note

The DMACRO format changed in releases 2019.2 and 2021.2. DMACRO files created with Calibre 2021.2 cannot be used by earlier versions of Calibre. DMACRO files created from 2019.2 to 2021.1 cannot be used in Calibre versions 2019.1 and earlier.

DMACRO files created prior to 2021.2 can be used in later versions of Calibre.

The compiled DMACRO has the following format:

```
DMACRO pm:<lib_name> in_<layer1> [in_<layer2> ...]
   options_string
   out_<marker1> [out_<marker2> ...] {
#DECRYPT
<Encrypted block>
#ENDCRYPT
}
```

The DMACRO has these elements:

- pm:<*lib_name*> — *lib_name* is the base filename of the pattern library. For example, the library *myLib.pmdb* is entered as pm:myLib.
- in_<*layer1*> — *layer1* is the first pattern layer in the library. Pattern layers are listed in order and prefixed with "in_". For example, pattern layer m1 is entered as in_m1.
- options_string — The literal text "options_string" is included in the DMACRO. The CMACRO command specifies the options to be used at runtime.
- out_<*marker1*> — marker1 is the first output marker in the library. Output markers are listed in order and prefixed with "out_". For example, an output marker named "Marker" is entered as out_Marker.

The compiled DMACRO file should not be edited. The compiled DMACRO pattern library is specified in the SVRF rule file with an Include statement. For example, where *myLibrary.svrf* is the compiled DMACRO:

```
INCLUDE "./myLibrary.svrf"
```

Note

Do not comment out the Include statement with /* … */ style comment characters. This results in the error "This is not a decryption error."

The CMACRO command is used to run pattern matching on a compiled DMACRO library. The number and order of the arguments in the CMACRO command must match the number and order of arguments in the DMACRO definition. See “CMACRO for Calibre Pattern Matching” on page 177.

For an example rule file, see “Using a DMACRO in a Rule File for a Pattern Matching Run” on page 185.

CMACRO for Calibre Pattern Matching

The CMACRO command runs pattern matching on the specified DMACRO pattern library. CMACRO creates derived layer(s) containing the marker layer shapes generated from the pattern matching process. Runtime options are specified with an option string.

Note

If ***options_string*** is non-empty, the pattern library must have been compiled with the enable_runtime_options argument. Failure to do this causes the message “ERROR: CMACRO (pm:<*lib*>) - The DMACRO is protected. This option is not allowed to be overridden as per the macro developer's specification.”

Usage

CMACRO *pattern_library_name in_param* [*in_param* ...] ***options_string out_param*** [*out_param* ...]

options_string

[{-enable_partial_match *options*} | {-puzzle_matcher *options*}]
[{-exclude_pattern_layers *layer_names_list*} | {-include_pattern_layers *layer_names_list*}]
[-expand_small_cells]
[-invalid_prop_value *invalid_value*]
[-keys *key_names*]
[-magnify *value*]
[-maxjog *maxjog_value*]
[-property *property_names*]
[-password *password*]
[-tile_size *value*
[-epd_string { on | off }]
[-cg_keep_real on]

Arguments

- ***pattern_library_name***

  A required parameter that specifies the name of the compiled pattern library as it appears in the DMACRO file. For example: pm:hotspot_lib

- ***in_param***

  A required parameter that specifies the name of an input layer, which can be an original or derived polygon layer. You can specify more than one ***in_param*** names. The input layers correspond to the pattern layers in the library specified with ***pattern_library_name***. The number and order of the ***in_param*** arguments must match the number and order of ***in_param*** arguments in the DMACRO definition.

- ***options_string***

  A required string specifying option names and values. The options control various aspects of the pattern matching process. The syntax is the following:

  "*-option_name value* [*-option_name value*] ..."

  Some options accept a Tcl list of values, which is specified as follows:

  "*-option_name* {*value1 value2 …*}"

  Line breaks are not allowed within the string. There is not a line continuation character.

  If no options are specified, you must specify a set of quotes surrounding a space (" ") for the ***options_string***. If ***options_string*** is non-empty, the pattern library must be compiled with the enable_runtime_options argument.

  See the section "options_string Arguments" for the available options.

- ***out_param***

  A required parameter specifying the name of an output layer generated by the pattern matching run. You can specify more than one ***out_param*** names. The output layers correspond to the pattern marker layers. The number and order of the ***out_param*** arguments must match the number and order of ***out_param*** arguments in the DMACRO definition. The ***out_param*** names must be unique within the rule file.

  The ***out_param*** names should not conflict with keywords in SVRF or with reserved names in the Tcl programming language. See "Avoiding Keyword - Name Conflicts in the Calibre Run" on page 18.

options_string Arguments

- -enable_partial_match '{'*layers_below_tolerance tol_value*
  [ref_layer *layer1* [*layer2 layer3*]]
  [req_layer '{'*layer1* [*tol_value1*]'}' ['{'*layer2* [*tol_value2*]'}']...]
  [exc_layer '{' *layer1* [*layer2*]...'}'] '}'

  Specifies to run per-layer partial matching. Partial matching identifies and highlights differences between a pattern and areas of the layout that produced a near match. Per-layer partial matching only runs on TEM1 (exact) match patterns with multiple layers. Pattern geometries are matched by layer. Only near matches are output, not exact matches. This option is useful in debugging expected match locations.

  The option -enable_partial_match cannot be specified with -puzzle_matcher.

  See "Pattern Debug with Partial Matching" on page 189 for information on running partial matching.

  layers_below_tolerance — This parameter value defines the maximum number of layers that do not meet the required tolerance value. The number of layers specified can be up to one less than the total number of layers. The program will run through layers until either a match is found or the maximum number of layers has been reached.

  tol_value — The area tolerance for a match as a specified as value between 0 and 1, where a value of 1 indicates a 100% match. Area tolerance is used to find matches

which may be slightly different in shape, or have a layer that has extra shapes that are still within the specified area tolerance. When -enable_partial_match evaluates a pattern, the layer difference percentage is calculated as 1- (XOR area/extent area).

ref_layer *layer1* [*layer2* [*layer3*]] — Reference layers are specified layers that have a high probability of being unaltered and without errors, and thus can be used as a reference layer. A partial match requires at least one layer to match 100% with the pattern. This serves as an alignment for the rest of the pattern. All other layers must match the orientation of the aligned reference layer. Up to three reference layers can be specified. This is helpful in successfully selecting an alignment layer. Specifying reference layers is not required, however, because the tool will search for appropriate reference layers when necessary.

A reference layer must include at least one real vertex. The list of reference layers must be enclosed in braces unless only one layer is specified. If only one or two layers are suitable reference layers, you can specify the same layer more than once.

If a partial match is not found but is expected, -enable_partial_match can be run again with different specified reference layers, or no layers specified.

req_layer ‘{’*layer1* [*tol_value1*]‘}’— Required layers are specified layers that must be included in the pattern and match by a set percentage. Typically, this match is equal to 100%; however, other percentages can be specified with *tol_value1* as a value between 0 and 1, where 1 is equal to 100%. A different percentage value can be specified for each required layer. There is no limit for how many required layers are specified. The list of required layers must be enclosed in braces unless only one layer is specified.

exc_layer ‘{’ *layer1* [*layer2*]...‘}’ — Excluded layers are specified layers that are ignored when determining the number of layers below tolerance. The list of excluded layers must be enclosed in braces unless only one layer is specified. The XOR results of ignored layers are still output. A layer specified as a ref_layer or req_layer cannot be specified with exc_layer. If -exclude_pattern_layers is specified, it overrides the effects of using exc_layer.

- -puzzle_matcher ‘{’ match_criteria *num_window tolerance* [others *other_tolerance*] [fixed_center | center_coverage *center_tolerance*] [corner_coverage *corner_tolerance*] [center_size *size*] [fixed_layer ‘{’ *layer_name* … ‘}’] [no_overlap] ‘}’

  Specifies to run grid-based partial matching, known as puzzle matching. Partial matching identifies and highlights differences between a pattern and areas of the layout that produced a near match. The input patterns must be TEM patterns with only Manhattan shapes and with an extent of at least 12 dbu by 12 dbu. Each pattern is divided into a grid of nine windows, and layout geometries are compared to pattern geometries per window. See “Grid-Based Partial Matching (Puzzle Matching)” on page 190 for additional details.

  The option -enable_partial_match cannot be specified with -puzzle_matcher.

  match_criteria *num_window tolerance* — Required parameters that specify the criteria for a partial match:

  num_window — The minimum number of windows that must match within the specified *tolerance* in order to qualify the layout region as a partial match.

If center_coverage and/or corner_coverage is specified, windows that are matched for those options do not contribute to the *num_window* match count. The maximum *num_window* value depends on the options center_coverage and corner_coverage:

- neither option: 1 <= *num_window* <= 9
- center_coverage: 1 <= *num_window* <= 8
- corner_coverage: 1 <= *num_window* <= 5
- corner_coverage and center_coverage: 1 <= *num_window* <= 4

If an invalid *num_window* value is used, a run-time error message is issued and the run is aborted.

tolerance — The area tolerance for a match within each window, specified as a value from 0 to 1, where a value of 1 indicates a 100% match. The *tolerance* is compared to the match value per window, calculated separately for each pattern layer as:

```
1 - ABS(Area(pattern_layer) - Area(design_layer))/Area(window)
```

In order for a window to qualify as a match, the window's match value for each pattern layer must meet the tolerance value.

others *other_tolerance* — The area tolerance for the windows that are not counted toward the "*num_window tolerance*" criteria. By default, *other_tolerance* is zero.

For example, "match_criteria 5 0.8 others 0.5" specifies that 5 windows must match with 0.8 area tolerance and the remaining 4 windows must match with 0.5 area tolerance.

fixed_center — Specifies that the center window must match exactly.

May not be specified with center_coverage.

center_coverage *center_tolerance* — Specifies the area tolerance for the center window as a value from 0 to 1. The center window must match within the *center_tolerance* for the layout region to qualify as a partial match. See *tolerance* for the definition of area tolerance.

corner_coverage *corner_tolerance* — Specifies the area tolerance for the corner windows as a value from 0 to 1. All four corner windows must match within the *corner_tolerance* for the layout region to qualify as a partial match. See *tolerance* for the definition of area tolerance.

center_size *size* — Specifies that the center window of the grid is of width and height *size* in user units. The size should be less than or equal to half the pattern extent in either direction for accurate operation. A warning is issued if this recommendation is not met.

An error occurs if the size value is greater than or equal to the pattern extent in either direction for any pattern in the library.

fixed_layer '{' *layer_name* … '}' — Specifies a list of one or more pattern layers that must match exactly in all windows.

no_overlap — Specifies to choose the best match from overlapping puzzles matches in a region. See "Using no_overlap to Eliminate Overlapping Matches" on page 195 with the "Grid-Based Partial Matching (Puzzle Matching)" topic.

The default is to output all qualified puzzle matches.

- {-exclude_pattern_layers *layer_names_list*} | {-include_pattern_layers *layer_names_list*}

  An optional argument set that specifies to include or exclude the listed pattern layers. You may specify one parameter set, not both. The pattern layer names are those defined in the pattern library and not those found in a verification (DRC) rules file.

  layer_names_list— Pattern layer names as defined in the pattern library. A library may have patterns composed of several layers. Use this option to specify a selection (or subset) of the library's pattern layers to include in or exclude from a matching run.

  Note

  Any constraint associated with an excluded layer is removed. Additionally, layer based markers are not included in the output when the associated layer is excluded.

- -expand_small_cells

  An optional argument that specifies to expand small cells during the pattern matching process.

- -invalid_prop_value *invalid_value*

  An optional argument that specifies the value to assign to a property when there is no assigned value in the compiled pattern, or the assigned value is invalid. The *invalid_value* must be a real number. The default is -1.

  For example, suppose a pattern library includes the property xyz but a value is not assigned for the property in pattern patA. When patA is matched, all properties in the library are attached to the output marker shapes. The property xyz is undefined for patA, so *invalid_value* is assigned as the value for the output marker shapes for matches to patA.

- -keys *key_names*

  An optional argument that specifies a list of one or more key names in the pattern library. Alphanumeric and underscore characters are allowed in the key names. When key names are specified, only patterns that include the listed keys are used in the pattern matching process. The default is to use all patterns in the pattern library. See "Pattern Keys" on page 74.

- -magnify *value*

  An optional argument used to specify a value for magnifying the input pattern geometries. The *value* must be any real number greater than 0. Values greater than 1 scale up the patterns; values less than 1 shrink the patterns. If a pattern is truncated due to magnification, a warning is generated.

- -maxjog *maxjog_value*

  An optional argument that specifies a jog tolerance for all patterns in the library. Jogs within the specified tolerance are ignored in both patterns and in the layout. The *maxjog_value* must be greater than or equal to 0 and is specified in user units. The default is 0. The maximum value for *maxjog_value* is two times the global constraint (2*cglobal). See "Jog Tolerance in Pattern Matching" on page 76.

  This option has no effect when -enable_partial_match or -puzzle_matcher is specified.

- -property *property_names*

  An optional argument that specifies the property names that are attached to the output shapes from a pattern matching run. The *property_names* can include alphanumeric and underscore characters. By default all properties in the pattern library are attached to the output.

  When -property is specified, only the properties included in the *property_names* list are attached to the output geometry. The properties should be defined in the pattern library, but no error is reported if they do not exist.

  See "Properties" on page 70 for general information. Also see the CMACRO option -invalid_prop_value.

- -password *password*

  An optional argument that specifies the password for the compiled pattern library. The *password* is a string containing one or more alpha-numeric and underscore characters.

  If the specification of runtime options was password-protected at compile time, then the same password must be included in the CMACRO options string if runtime options are specified. The -password argument is not necessary if runtime options are not specified.

- -tile_size *value*

  An optional argument that specifies the tile size used by the tool during pattern matching. The *value* is positive integer with a minimum value of 30. The default *value* is automatically calculated if -tile_size is not specified.

- -epd_string { on | off }

  An optional argument that turns on or off edge movement reporting. When on, the string property "EPD_string" is attached to the EPD marker layer that is specified when the library is compiled. The property value includes the pattern name and the list of constraint IDs with the corresponding displacement from the pattern edge to the layout edge. For example, a pattern with two constraints could have the following string property attached: "pat1 C1 0.25 C2 0".

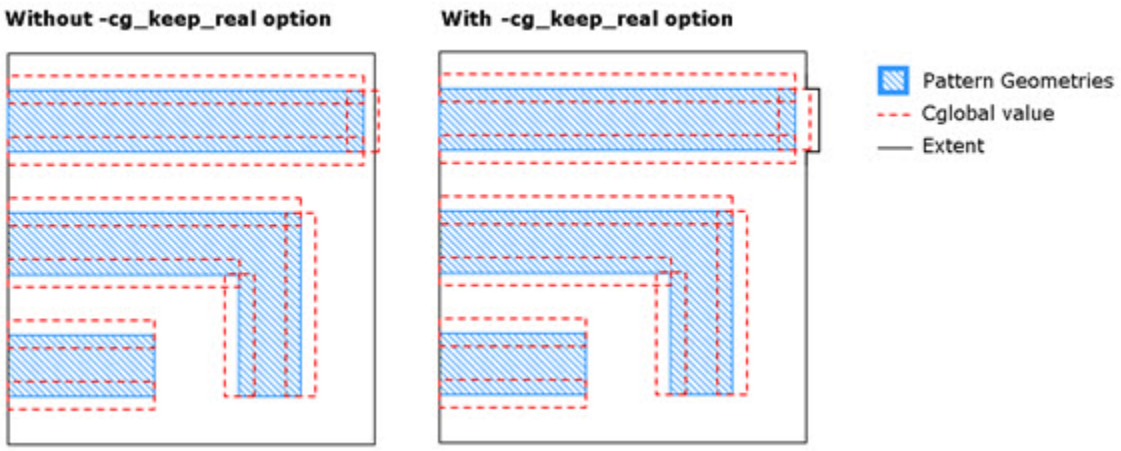
  For information on how to create the EPD marker layer, see the -epd_marker option in pdl_lib_mgr compile.

  This option has no effect when -enable_partial_match or -puzzle_matcher is specified.

- -cg_keep_real on

  An optional argument used to prevent pattern edges from becoming virtual edges. This can occur when the pattern edge has a cglobal value which causes the edge to be coincident with the pattern extent or completely outside the pattern extent. Specifying this option bumps out the extent at the edge location so the edge does not become virtual. All other dimensions remain the same, as shown in Figure 4-2. The left image shows the cglobal value extending past the extent. Using the -cg_keep_real on option, the extent is bumped out to accommodate the cglobal value. The edge remains real and the space between the extent and other geometries remains fixed, as shown on the right.

Figure 4-2. CMACRO Option String "-cg_keep_real on"



  This option has no effect when -enable_partial_match or -puzzle_matcher is specified.

Description

The CMACRO command must be used outside a rule check, as in this example:

```
CMACRO pm:myLibrary_1 m1 via1 "-maxjog 0.001" lib1_out_A
rule_1A {
   COPY lib1_out_A
   DFM RDB lib1_out_A "dfm.rdb"
}
```

The arguments m1 and via1 are layers that are to be matched to pattern layers. The argument lib1_out_A is the output marker for matched patterns. The rule check rule_1A copies the layer lib1_outA to the results database and also creates a DFM RDB of the layer.

See "Using a DMACRO in a Rule File for a Pattern Matching Run" on page 185 for a rule file example using the compiled DMACRO library and the CMACRO command.

Examples

The following examples are for the *options_string* within the CMACRO command.

Example 1

In this example, partial matching is run on a layout with three or more layers with one layer below tolerance, and an area tolerance of 0.99, or a 99% match with one of the layers.

```
-enable_partial_match {1 0.99}
```

Since reference layers are not specified, -enable_partial_match selects the reference layers.

Reference layers and required layers can be specified as shown here.

```
-enable_partial_match {1 0.99 ref_layer {poly cont met1}
   req_layer {poly 0.95}}
```

Example 2

In this example, partial matching is run on a layout with three or more layers. One layer below tolerance is specified with an area tolerance of 95%. Both layers L1 and L2 are used as reference layers to align the layers. Additionally, layer L1 is required to match at 100% and layer L2 is required to match at 99% only allowing for very minor differences.

```
-enable_partial_match {1 0.95 ref_layer {L1 L2}
   req_layer {{L1} {L2 0.99}}}
```

Example 3

For a library with pattern layers L1, L2, L3, and L4, you can specify to exclude layers L1 and L4 as follows:

```
-exclude_pattern_layers {L1 L4}
```

The following is functionally equivalent and yields the same result:

```
-include_pattern_layers {L2 L3}
```

Example 4

Only patterns with a key value of “pinch” are used in the pattern matching run.

```
-keys pinch
```

Example 5

In this example the input database is magnified by 20% its original size.

```
-magnify 1.2
```

Example 6

A property name can be attached to a geometry so that patterns is output only with the assigned property name. Pattern matches will only have the property name “pid” attached to the marker layer. Any other properties defined will be ignored.

```
-property pid
```

Example 7

Multiple CMACRO option strings can be specified at the same time. In this example both -keys and -property are used together.

```
CMACRO pm:patterns M1 "-keys pinch -property pid" matches
rule1 {
      COPY matches
}
```

Using a DMACRO in a Rule File for a Pattern Matching Run

A pattern matching rule file uses a CMACRO command to run pattern matching on a compiled DMACRO pattern library. The CMACRO creates derived layer(s) containing the marker shapes for each match.

When writing the CMACRO command, you should be aware of the target layers and pattern markers for the pattern libraries that are used. This example runs matching on two pattern libraries.

Library	Target Layers	Pattern Markers	Compile Option
libM1.pmdb	m1, via1	Marker, m1_custom	gen_count_layer enable_runtime_options
libM2.pmdb	m2, via2	Marker	gen_count_layer

Both libraries are compiled with the gen_count_layer argument, which adds a count marker to the compiled library. Library *libM1.pmdb* is also compiled with enable_runtime_options, so that a runtime option can be specified with a CMACRO options string. The compiled DMACRO pattern libraries are named *dmacro_libM1.svrf* and*dmacro_libM2.svrf*,

The DMACRO statements for each library are shown here:

```
DMACRO pm:libM1 in_m1 in_via1
  options_string
  out_Marker out_m1_custom out_count {
   <Encrypted block ...>
}

DMACRO pm:libM2 in_m2 in_via2
  options_string
  out_Marker out_count {
   <Encrypted block ...>
}
```

Prerequisites

- One or more compiled DMACRO pattern libraries.

See "Compiling a Pattern Library With the compile Utility" on page 173.

Procedure

1. Create a Calibre rule file using the following entries:

```
// Define the target layers.
LAYER m1    11
LAYER via1  12
LAYER m2    13
LAYER via2  14

// DRC results database
DRC RESULTS DATABASE "pm_results.db"

// Include the compiled DMACRO libraries
INCLUDE "./dmacro_libM1.svrf"
INCLUDE "./dmacro_libM2.svrf"

// Use CMACRO to call the DMACRO. The number and order of
// CMACRO arguments must match those of the DMACRO.
// Add runtime option -maxjog for libM1
CMACRO pm:libM1 m1 via1 "-maxjog 0.001" match_M1 custom_M1 count_M1
CMACRO pm:libM2 m2 via2 " " match_M2 count_M2

// Output matches to the results database.
libM1_match    { copy match_M1 }
libM1_custom   { copy custom_M1 }
libM1_count    { copy count_M1 }

libM2_match    { copy match_M2 }
libM2_count    { copy count_M2 }
```

2. Add the following code to also write the output to a DFM RDB, which includes property data:

```
libM1_match_dfm    { DFM RDB match_M1 M1_dfm.rdb }
libM1_custom_dfm   { DFM RDB custom_M1 M1_dfm.rdb }
libM1_count_dfm    { DFM RDB count_M1 M1_dfm.rdb }

libM2_count_dfm {DFM RDB count_M2 M2_dfm.rdb }
libM2_match_dfm {DFM RDB match_M2 M2_dfm.rdb }
```

   By default, DFM RDB writes out results as one rule check for each layout cell.

3. Invoke a Calibre nmDRC run at the command line with the rule file you just created.

```
calibre -drc rules |& tee log
```

4. Use Calibre RVE to view the results database (RDB).

   See "Using Calibre RVE to View Match Results" on page 187.

Results

Pattern matching results are reported in the DRC results database (*pm_results.db*) and the DFM RDBs (*M1_dfm.rdb* and *M2_dfm.rdb*).

Related Topics

compile

DMACRO File for Calibre Pattern Matching

CMACRO for Calibre Pattern Matching

Pattern Matching Invocation

Count Markers

Using Calibre RVE to View Match Results

Calibre RVE is a graphical debug program that interfaces with most IC layout tools. You can use Calibre RVE to view your pattern matching results.

Prerequisites

- A pattern matching run is complete. See one of these topics:
  - "Using a DMACRO in a Rule File for a Pattern Matching Run" on page 185
  - "Running Pattern Matching From the GUI" on page 100
  - "Running Pattern Matching From a Calibre Layout Viewer" on page 103

Procedure

1. Open your layout in a viewer that is supported by Calibre RVE.
2. Open Calibre RVE for DRC:
   - Calibre DESIGNrev and other Calibre viewers: **Verification > Start RVE**
   - Most other design tools: **Calibre > Start RVE**

   This displays the Calibre RVE dialog box.
3. Do the following to open the pattern matching results:
   a. Enter the results database (RDB) filename in the Database field.
   b. Select a Database Type of "DRC/ERC".
   c. Click **Open**.

   A tree view of the results is displayed on the left, with individual results listed on the right.

4. (Optional) If the pattern matching run produced DFM RDB files in addition to the main DRC results database, click the **Open Side RDBs** button ( ) or choose **File > Side RDBs**.

   The properties attached to each result are included in the DFM RDB. To view the properties, make sure the Result View on the right is set to Details view (**View > Result Options > Details**)

5. To highlight results, select a check in the tree view on the left or a result in the result view on the right, then type H on the keyboard or click the **Highlight** button ( ) in the toolbar.

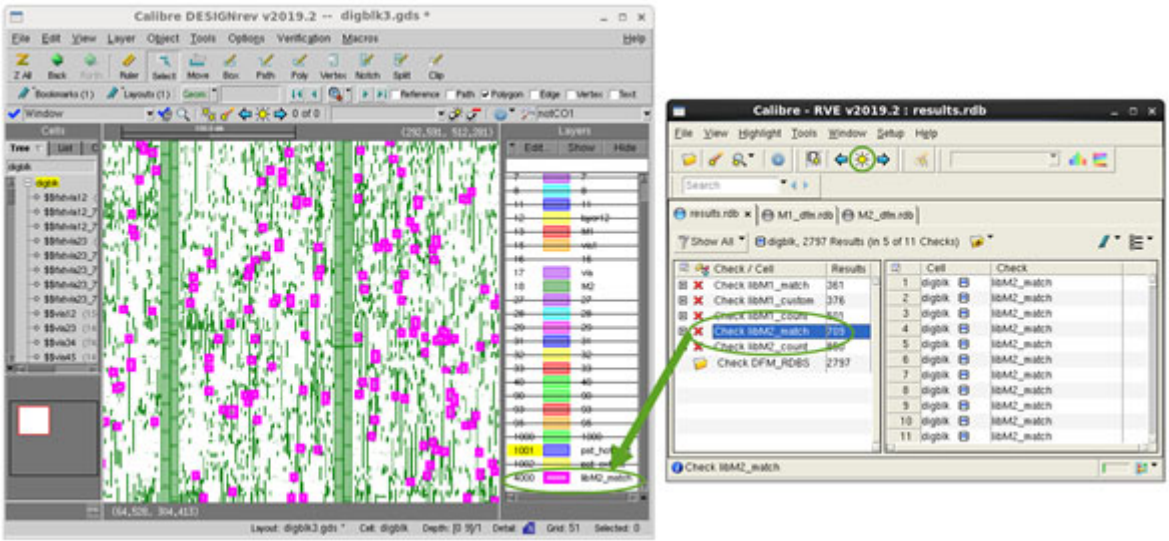
   Depending on your run, you may want to set one of these options in Calibre RVE.

   - **Highlight > Highlight in Context** — Highlights results in the cell in which they are reported, rather than in the top cell. Only available if the DRC CELL NAME YES CELL SPACE XFORM ALL statement is in the rule file.
   - **Display lower-level cell highlights in top cell** — In Calibre RVE, choose **Setup > Options** and choose the **Highlighting** category. Only available when highlighting to Calibre DESIGNrev. When used, the view depth in Calibre DESIGNrev must be set to at least one to view the highlights at the top level.

Results

The following figure highlights match locations from the rule check libM2_match.

Figure 4-3. Pattern Matching Results in Calibre RVE



Related Topics

Getting Started with Calibre RVE [Calibre RVE User's Manual]

Pattern Debug with Partial Matching

Partial matching identifies areas in a layout that match an input pattern within a given tolerance. If Calibre Pattern Matching does not report a match where expected, partial matching can help debug the problem. Partial matching applies only to exact TEM patterns.

Note

Partial matching is only intended for debugging on a small area of the layout. It is not a substitute for pattern matching with appropriately defined pattern constraints. In particular, partial matching may not find all locations of a partial match if a suitable partial match candidate is not identified.

A partial matching run produces the per-layer XOR difference between the pattern layer and the corresponding layout layer in the partially-matched region of the layout. The XOR difference can be used to debug why the pattern did not match.

Partial matching requires more processing than standard pattern matching. It is best used on a small area of the layout in order to debug a pattern that does not match where it is expected to match.

There are two methods for running partial matching, depending on whether you are debugging a single-layer pattern or a multilayer pattern. Both methods are enabled with an option in the CMACRO options string; see “CMACRO for Calibre Pattern Matching” on page 177.

- **Grid-Based Partial Matching (Puzzle Matching)**

  Puzzle matching finds partial matches to both single-layer and multilayer TEM patterns. The pattern must be an exact TEM pattern with only Manhattan shapes and an extent of at least 12 dbu by 12 dbu. The pattern is divided into a grid of nine windows and a matches are evaluated on a per-window basis. A partial match candidate is located by exact match to at least one window, then a partial match is returned if a specified number of windows match within a specified tolerance.

  Puzzle matching is specified with the -puzzle_matcher option in the CMACRO options string.

- **Per-Layer Partial Matching for Multilayer Patterns**

  Per-layer partial matching finds partial matches to multilayer patterns. The pattern must be an exact TEM pattern with multiple layers. Pattern geometries are matched by layer. A partial match candidate is located using one or more reference layers, then a partial match is returned if a specified number of layers match within a specified tolerance. Additional options are available to specify tolerance requirements per layer.

  Per-layer partial matching is specified with the -enable_partial_match option in the CMACRO options string.

Both partial matching methods require that the compiled pattern library have matched layer marker output for each pattern layer.

Grid-Based Partial Matching (Puzzle Matching)

Partial matching identifies regions in a layout that match an input pattern within a given tolerance. Grid-based partial matching, or puzzle matching, divides the pattern into a grid of nine windows and the matching criteria is calculated on a per-window basis. A match candidate is located by exact match to at least one window, then a puzzle match is returned if a specified minimum number of windows match within a specified tolerance.

Note

Puzzle matching is only intended for debugging on a small area of the layout. It is not a substitute for pattern matching with appropriately defined pattern constraints. In particular, puzzle matching may not find all locations of a puzzle match if a suitable match candidate is not identified.

Puzzle matching is enabled in the CMACRO options string with the -puzzle_matcher option, which has this syntax:

The patterns used for puzzle matching must be exact TEM patterns with only Manhattan shapes and with an extent of at least 12 dbu by 12 dbu.

-puzzle_matcher ‘{’ match_criteria *num_window tolerance* [others *other_tolerance*] [fixed_center | center_coverage *center_tolerance*] [corner_coverage *corner_tolerance*] [center_size *size*] [fixed_layer ‘{’ *layer_name* … ‘}’] [no_overlap] ‘}’

- match_criteria *num_window tolerance* — Required parameters that specify the criteria for a puzzle match:
  - *num_window* — The minimum number of windows that must match within the specified tolerance in order to qualify the layout region as a puzzle match.

    If center_coverage and/or corner_coverage is specified, windows that are matched for those options do not contribute to the *num_window* match count. The maximum *num_window* value depends on the options center_coverage and corner_coverage:

    - neither option: 1 <= *num_window* <= 9
    - center_coverage: 1 <= *num_window* <= 8
    - corner_coverage: 1 <= *num_window* <= 5
    - corner_coverage and center_coverage: 1 <= *num_window* <= 4

If an invalid *num_window* value is used, a run-time error message is issued and the run is aborted.

- *tolerance* — The area tolerance for a match within each window, specified as a value from 0 to 1, where a value of 1 indicates a 100% match. The *tolerance* is compared to the match_value per window, calculated as

  ```
  1 - |Area(pattern_layer) - Area(design_layer)|/Area(window)
  ```

  In order for a window to qualify as a match, the window's match value for each pattern layer must meet the tolerance value.

See the section "Using the center_coverage, corner_coverage, others, and fixed_center Options" on page 194 for details on using those options to define more precise control of the area tolerance.

- others *other_tolerance* — The area tolerance for the windows that are not counted toward the "*num_window tolerance*" criteria. By default, *other_tolerance* is zero.

  For example, {match_criteria 5 0.8 others 0.5} specifies that 5 windows must match with 0.8 area tolerance and the remaining 4 windows must match with 0.5 area tolerance.

- fixed_center — Specifies that the center window of the grid must match exactly.

  May not be specified with center_coverage.

- center_coverage *center_tolerance* — Specifies the area tolerance for the center window as a value from 0 to 1. The center window must match within the *center_tolerance* for the layout region to qualify as a puzzle match. See *tolerance* for the definition of area tolerance.

  For example, the center_coverage option can be used to specify a higher area tolerance for the center window:

  ```
  -puzzle_matcher {match_criteria 5 0.85 center_coverage 0.95}
  ```

  A puzzle match must have the center window match at 95%, and 5 other windows match at 85%.

- corner_coverage *corner_tolerance* — Specifies the area tolerance for the corner windows as a value from 0 to 1. All four corner windows must match within the *corner_tolerance* for the layout region to qualify as a puzzle match. See *tolerance* for the definition of area tolerance.

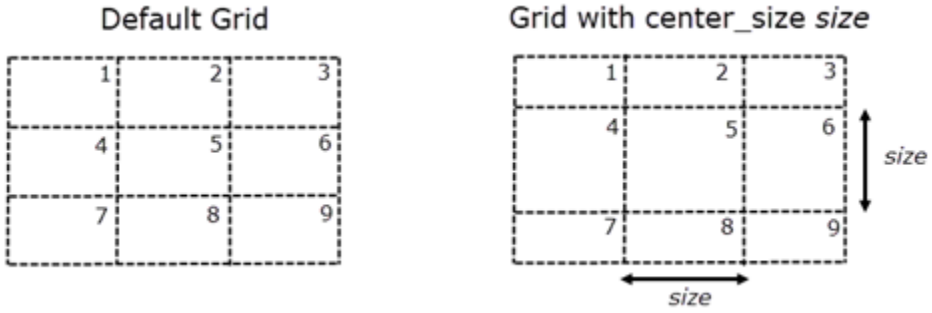
  For example, the corner_coverage option can be used to specify a lower area tolerance for the corner windows:

  ```
  -puzzle_matcher {match_criteria 3 0.95 corner_coverage 0.5}
  ```

  A puzzle match must have the 4 corner windows match at 50%, and 3 other windows match at 95%.

- center_size *size* — Specifies that the center window of the grid is of width and height *size* in user units. The default is to use a grid of nine approximately equal windows.

Figure 4-4. Grid for Puzzle Matching



The *size* should be less than or equal to half the pattern extent in either direction for accurate operation. A warning is issued if this recommendation is not met.

An error occurs if the *size* value is greater than or equal to the pattern extent in either direction for any pattern in the library.

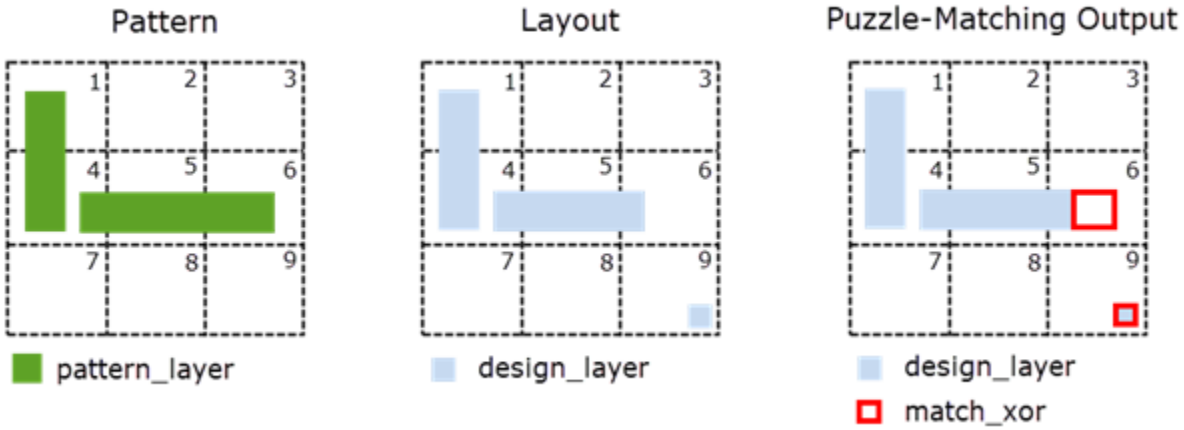
- fixed_layer '{' *layer_name* … '}' — Specifies a list of one or more pattern layers that must match exactly in all windows.
- no_overlap — Specifies to choose the best match from overlapping puzzle matches in a region. See "Using no_overlap to Eliminate Overlapping Matches" on page 195.

  The default is to output all qualified puzzle matches.

Basic Example

Consider the following pattern and layout region to be checked with grid-based puzzle matching:

Figure 4-5. Puzzle Matching Basic Example



The pattern extent is divided into nine windows, taking into account the optional center_size specification. The match value is calculated for each window as follows:

```
match_value = 1 - |Area(pattern_layer) - Area(design_layer)|/Area(window)
```

A layout region is considered a puzzle match if the following is true:

- At least one window containing a pattern edge or vertex is an exact match (satisfied by windows 1, 4, and 5 in the preceding figure).
- At least *num_window* windows have a match_value greater than or equal to *tolerance*. Windows that are empty in both the pattern and design have a match_value of 1.
- If fixed_center is specified, the center window must be an exact match.
- If fixed_layer is specified, the specified layers must match exactly in all windows.

When using puzzle matching, the compiled pattern must include matched layer marker output, which can be added when the pattern is compiled. See “Setting Up and Running Partial Matching” on page 196. Unlike in standard pattern matching, the matched layer marker output is the XOR of the pattern layer and the design layer in the region of the puzzle match.

The following CMACRO call and rule check are used with the pattern and layout shown in Figure 4-5. The compiled pattern library my_lib has only one marker output—the matched layer marker which is written to the derived layer xor_out.

```
CMACRO pm::my_lib design_layer "-puzzle_matcher {match_criteria 7 0.8}"
                   xor_out

match_xor {COPY xor_out}
```

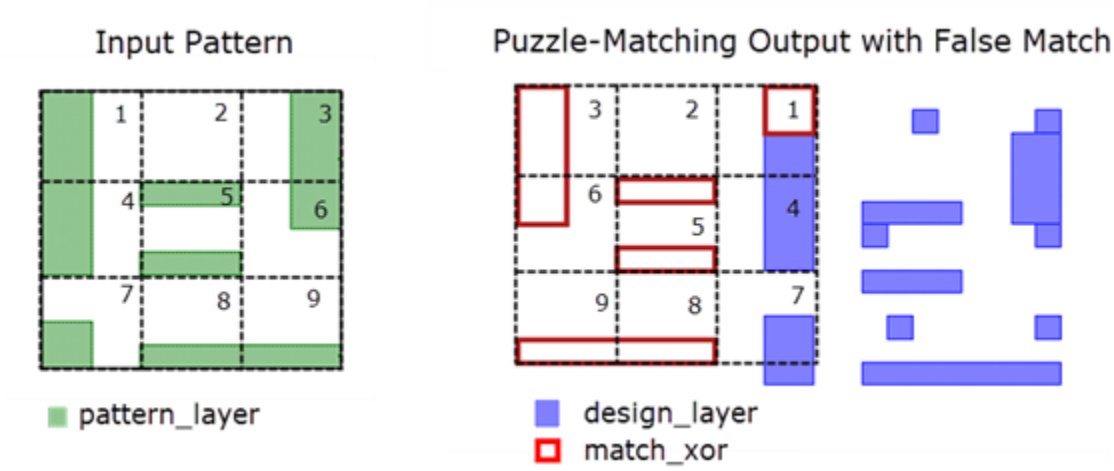
The match_criteria specifies that at least 7 windows must have a match_value of 0.8 or greater. For the pattern and layout in the example, all but two windows (6 and 9) are an exact match, so puzzle matching returns a match. For this example, the match value for windows 6 and 9 is not relevant. The results of the rule check match_xor contain the XOR difference between the pattern layer and the design layer, as shown in Figure 4-5.

Reducing False Matches

The options fixed_center and center_size can be used to reduce the possibility of false matches, especially when the *tolerance* and/or *num_window* values are small.

For example, consider the pattern and layout region shown in the following figure. The CMACRO options string was “-puzzle_matcher {match_criteria 7 0.5}”, where the tolerance value of 0.5 is low, allowing the pattern to match in a flipped placement to a mostly empty layout region.

Figure 4-6. Puzzle Matching with False Partial Match



False partial matches such as the one in the preceding figure can be reduced with high *tolerance* values, high *num_window* values, and the use of the fixed_center option. Using the CMACRO options string "-puzzle_matcher {match_criteria 7 0.8 fixed_center}", the layout region shown in the preceding figure is not a puzzle match.

Using the center_coverage, corner_coverage, others, and fixed_center Options

The options center_coverage, corner_coverage, others, and fixed_center provide precise control of the area tolerance for puzzle matching. Several examples are given.

Example 1: A puzzle match must have the center window match at 95%, and 5 remaining windows match at 85%:

```
-puzzle_matcher {match_criteria 5 0.85 center_coverage 0.95}
```

Example 2: A puzzle match must have the center window match at 95%, the four corner windows match at 25%, and two remaining windows at 75%:

```
-puzzle_matcher {match_criteria 2 0.75 center_coverage 0.95
corner_coverage 0.25}
```

Example 3: A puzzle match must match four windows at 80%, including an exact match to the center window. When using fixed_center, a matched center window counts toward the total *num_window* count. The other five windows must match at 50%.

```
-puzzle_matcher {match_criteria 4 0.8 fixed_center others 0.5}
```

Example 4: This specification causes a runtime error because corner_coverage is specified and *num_window* is greater than five:

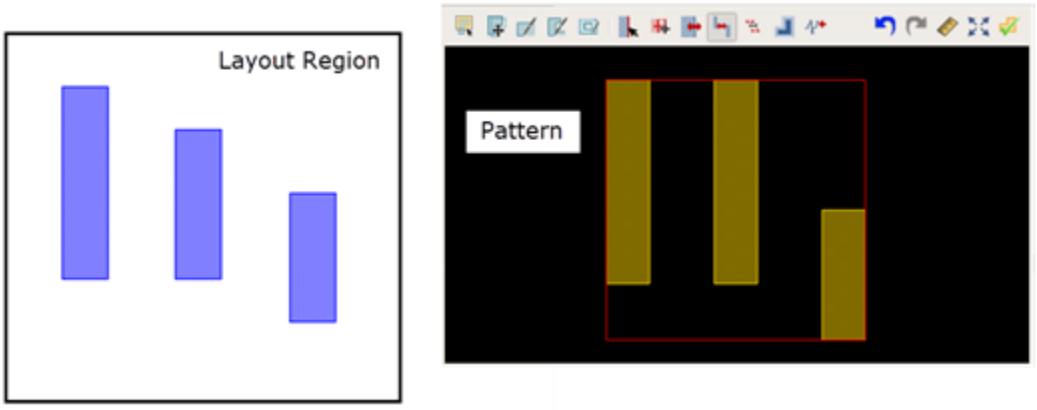
```
-puzzle_matcher {match_criteria 6 0.75 corner_coverage 0.25}
```

The error begins with “Error DECRYPT4 on encrypted line …”.

Using no_overlap to Eliminate Overlapping Matches

It is possible for multiple, overlapping matches to be made in the same region of the layout. Use the no_overlap option to return only the best match when overlapping matches are found.

The layout and a pattern are shown in the following figure.



The pattern is compiled with count markers and matched layout marker output:

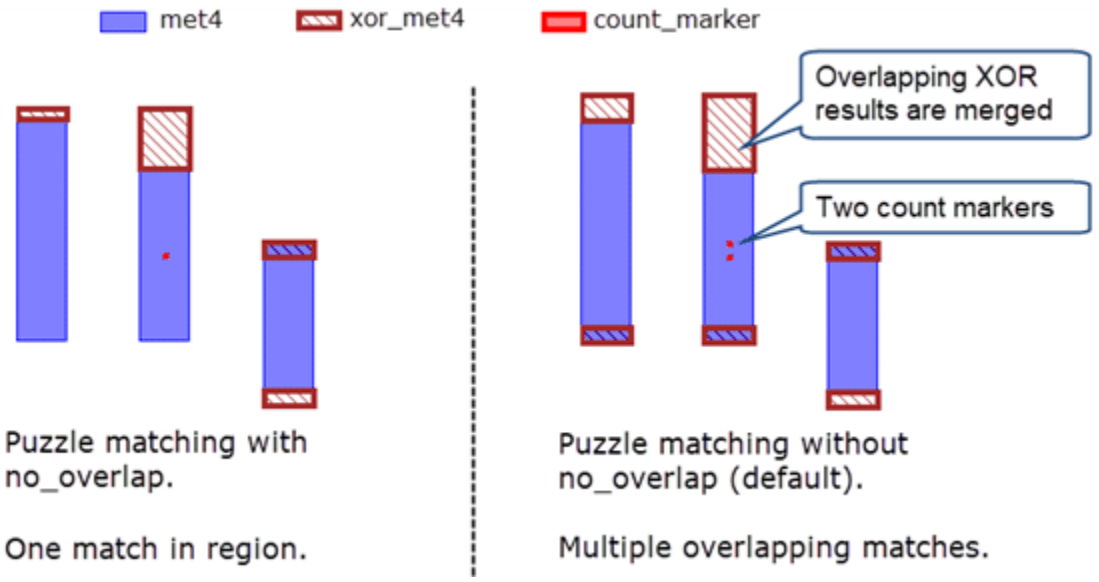
```
pdl_lib_mgr compile input one_pat.pmdb output one_pat.svrf \
   marker Marker1 matched layer 1 \
   enable_runtime_options gen_count_layer > compile.log
```

The SVRF rule file has the following CMACRO statement with no_overlap, and outputs the count marker and XOR match results:

```
CMACRO pm:one_pat met4 "-puzzle_matcher {match_criteria 7 0.85
no_overlap}" xor_met4 count_marker

xor_met4 {COPY xor_met4}
count_marker {COPY count_marker}
```

With no_overlap, only one match is returned in the layout region, as shown in the image on the left. If the CMACRO does not include no_overlap, two matches are returned, as shown in the image on the right. The count markers do not overlap, but the match results overlap and merge.



Setting Up and Running Partial Matching

Partial matching can be used to debug a pattern that does not match where it is expected to match. Partial matching requires a pattern library compiled with appropriate options. Partial matching is specified with CMACRO options in the rule file. The output of a partial matching run is the per-layer XOR difference between the pattern layers and the design layers in the layout regions that meet the partial matching criteria.

Note

Partial matching is only intended for debugging a small area of the layout. It is not a substitute for pattern matching with appropriately defined pattern constraints. In particular, partial matching may not find all locations of a partial match if a suitable partial match candidate is not identified.

Prerequisites

- A pattern library, and knowledge of the name of the pattern to debug.

  Partial matching only applies to exact TEM patterns. For grid-based puzzle matching, the patterns must be Manhattan shapes only.

Note

The examples in this procedure assume the library name is “library” and the pattern to debug is “pattern1”. The pattern has three layers: M1, V1, and M2.

Procedure

1. Compile the pattern library into a DMACRO.

   Compile options are used to do the following:

   - Add matched layer marker output for each pattern layer.
   - Enable runtime CMACRO options.
   - (Optional) Output a count marker.

   The following command is used for this example:

   ```
   pdl_lib_mgr compile input library.pmdb output lib_pat1.svrf
      marker matched_m1 matched layer M1
      marker matched_v1 matched layer V1
      marker matched_m2 matched layer M2
      enable_runtime_options
      gen_count_layer > compile.log
   ```

   If you want debug only one pattern in a library, you can use the option “patterns *pattern_name*” in the compile command.

   For this example, the compiled DMACRO includes these lines:

   ```
   DMACRO pm:library in_M1 in_V1 in_M2
   options_string
   out_matched_m1 out_matched_v1 out_matched_m2 out_match_count
   ```

   Tip

   You can also use the pattern matching GUI to compile a pattern library with the necessary options, including the matched layer compile-time markers. See “Compiling a Pattern Library with the Calibre Pattern Matching GUI” on page 105.

2. Determine which partial matching method to use and which options to specify in the CMACRO options string.

   See “Pattern Debug with Partial Matching” on page 189 for a summary.

   - **Grid-Based Partial Matching** — Supports both single-layer and multilayer patterns and uses the -puzzle_matcher option in the CMACRO call. See “Grid-Based Partial Matching (Puzzle Matching)” on page 190 for an overview and tips on choosing options.
   - **Per-Layer Partial Matching** — Supports multilayer patterns only and uses the -enable_partial_match option in the CMACRO call. See “CMACRO for Calibre Pattern Matching” on page 177 for the available options.

3. Set up the CMACRO call and the rule checks in the rule file.

Use the CMACRO option string arguments determined in the previous step, the appropriate input and output arguments for your pattern, and the compile options used in Step 1. Include rule checks for each output layer from the partial matching run.

This example is for a multilayer pattern, so the -enable_partial_match option is used. The first two numbers, 1 and 0.9, specify that at most 1 layer may be below the match tolerance of 0.9. The design layers v1 and m2 must match exactly, so they are listed with the req_layer argument.

```
CMACRO pm:library m1 v1 m2
"-enable_partial_match {1 0.9 req_layer {v1 m2}}"
xor_m1 xor_v1 xor_m2 count_marker

XOR_m1 {COPY xor_m1}
XOR_v1 {COPY xor_v1}
XOR_m2 {COPY xor_m2}
partial_matches {COPY count_marker}
```

If working with a full chip layout, run partial matching on only a small portion of the layout where a match was expected but not found.

4. Run Calibre.

   ```
   calibre -drc -hier rules > log
   ```

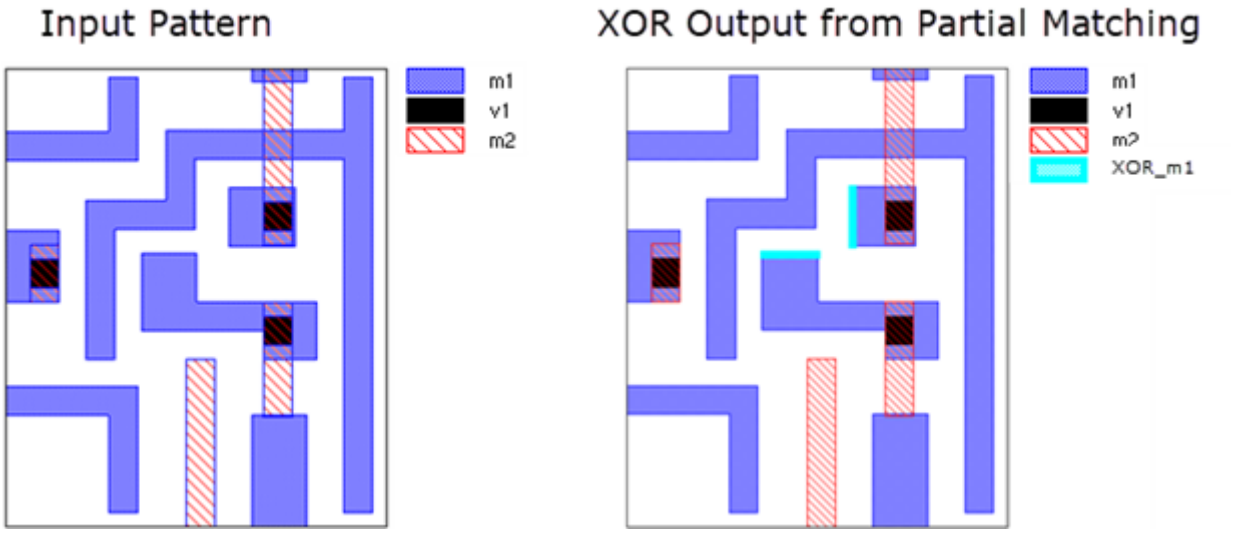
   The output layers from the matched layer markers contains the XOR difference of the pattern layer and the design layer in the regions that meet the partial matching criteria.

5. View the results using Calibre RVE.

   The XOR layer differences appear in the results and can be highlighted in the layout.

   In this example, the XOR differences for layer m1 are in the rule check XOR_m1. The rule checks for the layers m2 and v1 (XOR_v1 and XOR_m1) do not have results, as these layers were specified to match exactly (the req_layer argument in the CMACRO option string).

Figure 4-7. Partial Matching



Calibre Pattern Matching Reference Library Flow

The Calibre Pattern Matching Reference Library (PMRL) flow enables you to efficiently compare a reference pattern library to patterns captured from a layout using the DFM Pattern Classify operation.

The PMRL flow only considers pattern geometries and pattern extents. In contrast, the pdl_lib_mgr compare utility also considers layer extents, regions, and markers. The PMRL flow is more efficient than the compare utility, especially for large libraries.

The PMRL flow uses a pattern reference library saved in a proprietary PMRL format. The PMRL file is only used in the DFM Pattern Classify operation with the REFERENCE_LIB keyword. It cannot used as input to any other command. The PMRL file cannot be viewed in the Calibre Pattern Matching GUI, edited, or used in a pattern matching run.

Use the steps in the following table to compare libraries with the PMRL flow:

Table 4-1. Pattern Matching Reference Library (PMRL) Flow

Step	Description
1. Create the reference library (PMRL file) using the pdl_lib_mgr convert utility.	See “Creating a Pattern Matching Reference Library (PMRL)” on page 200.
2. Perform the compare using DFM Pattern Classify.	See “Comparing Patterns with the Pattern Matching Reference Library Flow” on page 202.

Creating a Pattern Matching Reference Library (PMRL)

The Pattern Matching Reference Library (PMRL) file is created from an existing pattern matching database (PMDB) library using the pdl_lib_mgr convert utility. The PMDB can be created using the DFM Pattern Classify or DFM Pattern Capture operations. The reference library is used in the PMRL flow to compare patterns in the reference library with patterns captured using the DFM Pattern Classify operation.

For example purposes, the procedure assumes patterns are captured from a layout using a seed or hotspot layer. The layout and seed layer correspond to the reference design.

Prerequisites

- A layout in a format supported by Calibre.
- A layer that can be used as the seed (hotspot) layer in a DFM Pattern Classify operation.

Procedure

1. Use the DFM Pattern Classify operation to capture patterns from a layout and save them to a pattern library (PMDB) file. For example:

```
LAYOUT PATH "ref_layout.gds"
LAYOUT SYSTEM GDS
LAYOUT PRIMARY "top"

DRC RESULTS DATABASE drc_capture_results
DRC CELL NAME YES CELL SPACE XFORM

LAYER M2 18
LAYER pat_hotspot 1001

hs_out = DFM PATTERN CLASSIFY
   LAYER_TARGET M2
   SEED pat_hotspot HALO 1.45
   OUTFILE  lib.pmdb

hs_out_check {
   @ RVE Show Layers: 18 1001
   COPY hs_out
   DFM RDB hs_out "cap_results.dfm" CELL SPACE CHECKNAME "%_l_"
}
```

   Make a note of the HALO value. The same value should be used in the later procedure to compare patterns using the PMRL flow.

   You can also use DFM Pattern Capture to capture patterns and save a pattern library. In this case make a note of the PATTERN_HALO value for later use.

2. Run Calibre with the rule file:

```
calibre -drc -hier rules_lib_pmdb >& log
```

   This creates the pattern library *lib.pmdb*.

3. Use the Calibre Pattern Matching GUI to view the pattern library (*lib.pmdb*) and confirm that all patterns are to be used in the reference library.

4. Convert the library as follows, where *lib.pmdb* is the input PMDB and *ref_lib.pmrl* is the output reference library in PMRL format:

```
pdl_lib_mgr convert input lib.pmdb output ref_lib.pmrl write_pmrl
```

Results

The reference library *ref_lib.pmrl* is created. Proceed to “Comparing Patterns with the Pattern Matching Reference Library Flow” on page 202.

Comparing Patterns with the Pattern Matching Reference Library Flow

The DFM Pattern Classify operation with the REFERENCE_LIB keyword compares patterns in a reference library to patterns captured from a layout.

For example purposes, the following procedure assumes patterns are captured from a layout using a seed or hotspot layer. The layout and seed layer correspond to the updated design that you want to compare to a reference design.

The rule file makes use of the is_new_pat property attached to the output shapes from the DFM Pattern Classify operation. Review the command reference for details on this property and the use of the REFERENCE_LIB keyword.

The optional OUTREF_ALL keyword is used to create a new reference library file that contains the patterns in the input reference library plus the new patterns found in the current layout.

Prerequisites

- A reference library in Pattern Matching Reference Library (PMRL) format. See “Creating a Pattern Matching Reference Library (PMRL)” on page 200.
- A layout in a format supported by Calibre.
- A layer that can be used as the seed (hotspot) layer in a DFM Pattern Classify operation.

Procedure

1. Use the DFM Pattern Classify operation with the REFERENCE_LIB keyword to capture new patterns and compare them to an existing reference library.

   The HALO value should match what was used to capture the PMDB that was used to create the reference library (PMRL). The following rule file corresponds to that used in “Creating a Pattern Matching Reference Library (PMRL)” on page 200.

   The DFM Pattern Classify operation attaches the class and is_new_pat properties to the output shapes from the operation. These two properties are used to create a derived layer new_hs that has hotspot locations corresponding to new patterns.

```
LAYOUT PATH "new_layout.gds"
LAYOUT SYSTEM GDS
LAYOUT PRIMARY "top"

DRC RESULTS DATABASE drc_compare_results
DRC CELL NAME YES CELL SPACE XFORM

LAYER M2 18
LAYER pat_hotspot 1001
```

```
hs_out = DFM PATTERN CLASSIFY
   LAYER_TARGET M2
   SEED pat_hotspot HALO 1.45
   REFERENCE_LIB  ref_lib.pmrl
   OUTREF_ALL new_reflib.pmrl
   // OUTFILE_NEW new_lib.pmdb
   // OUTFILE_COMMON common_lib.pmdb

// new layer with only the output shapes that correspond to
// new (is_new_pat == 1) and valid (class != -1) patterns
new_hs = DFM Property hs_out
   [new_pat = PROPERTY(hs_out, is_new_pat)] == 1
   [ - = PROPERTY(hs_out, class)] != -1

new_hs_pats {
@ RVE Show Layers: 18 1001
   DFM Copy new_hs
   DFM RDB new_hs compare.rdb CELL SPACE CHECKNAME "%_l_"
      COMMENT "@ RVE Show Layers: 18 1001"
}

hs_out {
@ RVE Show Layers: 18 1001
   DFM Copy hs_out
   DFM RDB hs_out compare.rdb CELL SPACE CHECKNAME "%_l_"
      COMMENT "@ RVE Show Layers: 18 1001"
}
```

The check new_hs_pats includes only the shapes that correspond to new patterns. The check hs_out contains all shapes on the seed layer, including those that did not result in a valid pattern. The RVE Show Layers check text comment is used to control the visibility of the design layers when highlighting the results. See “RVE Show Layers” in the *Calibre RVE User’s Manual*.

The DFM Pattern Classify keywords OUTFILE_NEW and OUTFILE_COMMON specify to create pattern libraries with the new and common patterns between the current capture and the reference library. Uncomment these lines in the preceding example if you want to compare results by viewing a pattern library.

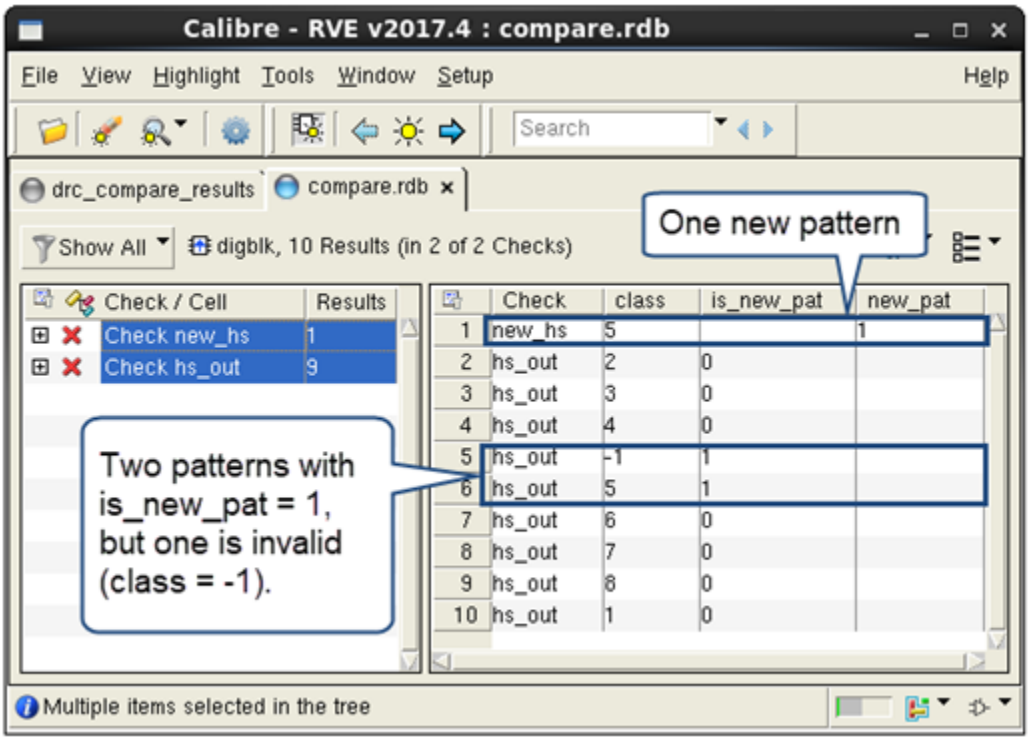
2. Run Calibre with the rule file:

```
calibre -drc -hier rules_pmrl_compare >& log
```

3. Open the layout in Calibre DESIGNrev and the results in Calibre RVE for DRC:

```
calibredrv new_layout.gds -rve -drc drc_compare_results compare.rdb
```

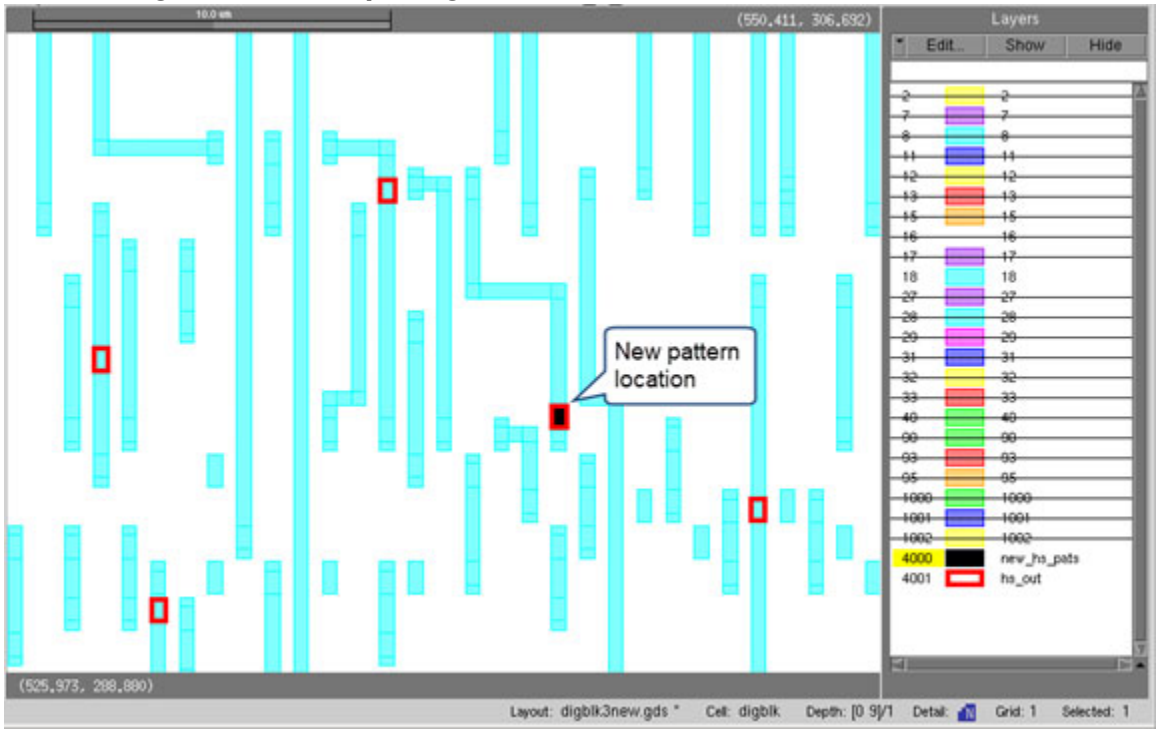
For the example shown, there is one new pattern in the check new_hs. There are nine pattern locations in the check hs_out, but one of the locations has the class property equal to -1, which indicates that a valid pattern was not found at the location.



4. Highlight the results from both the new_hs_pats and hs_out checks.

   The highlights from new_hs_pats show the locations of new patterns. The highlights from hs_out show the output shapes from the DFM Pattern Classify operation, which are the same as the shapes on the seed layer. The image in the following figure is zoomed into the region around the new pattern location.

Figure 4-8. Comparing Pattern Libraries with the PMRL Flow



Results

New pattern locations are determined by highlighting the output shapes that have the is_new_pat property equal to 1 and have the class property not equal to -1. Depending on your design flow, these locations may need more analysis and inspection.

Chapter 5
Pattern Library Manager Utility

The pattern library manager utility can be used to manipulate and manage PMDB files.

The pdl_lib_mgr utility merges multiple libraries into a single pattern library file and converts a PMDB file into an SVRF DMACRO file for pattern matching. This utility also converts PDL files from an earlier version of the Calibre Pattern Matching tool to a PMDB library file. The utility is invoked at the command line.

Caution

The pattern library manager utility for Calibre releases prior to 2011.3 cannot read pattern libraries generated by 2011.3 or later releases.

pdl_lib_mgr Utility Options

The pdl_lib_mgr has several different command line options.

Utility	Summary
cluster	Identifies similar patterns within a pattern library. Each pattern in the output pattern library includes a cluster property that identifies the cluster group the pattern belongs to.
compare	Compares pattern libraries and determines if any patterns are duplicated. Only exact patterns are compared. A log file with results is generated by default. Optionally, output pattern libraries can be generated that contain common or unique patterns.
compile	Generates an SVRF DMACRO file from the input pattern library. The DMACRO file is the compiled pattern library that is used in a pattern matching run.
convert	Generates new pattern matching libraries (PMDBs) from libraries created with older versions of the Calibre Pattern Matching tool or saves the library in the Pattern Matching Reference Library (PMRL) format.
export_properties	Generates a comma-separated values (CSV) file that contains the property values defined for each pattern in the input pattern library.
merge	Combines multiple pattern matching libraries (PMDBs) into a single PMDB library and removes duplicate patterns. If only one library is input, the command removes the duplicate patterns in that library.
prompt	Opens the Tcl command-line interface to the Pattern Manager Library Utility, or executes a specified Tcl script.
unfold	Reads the patterns in a pattern matching library (PMDB) and outputs a series of patterns without constraints. The output patterns are the result of applying the pattern constraints in various permutations. Options control the number of output patterns and the edge movement between permutations.
write_oasis	Generates an OASIS file including all polygon information from pattern layers. The utility can also generate PNG-formatted image files corresponding to each pattern in the library.
PMDB_info	Reports information about the input pattern library (PMDB).

cluster

Identifies similar patterns within a pattern library. Each pattern in the output pattern library includes a cluster property that identifies the cluster group the pattern belongs to.

Usage

Syntax 1:

pdl_lib_mgr [-turbo [*number_of_processors*]] [-remote *host*[, *host* …] | -remotefile *filename*]]
cluster input *input_lib* output *cluster_lib*
[compress *compress_library* [no_unique]] [pattern_start *num*] [pattern_limit *count*]
[prop_name *name*] [cluster_unique] [allowed_shift *shift_val*]
[center_halo *halo_size* [crop] [exact | exact_cg *cglobal* | line_ends *line_ends_options*]
[outer_halo [exact | exact_cg *cglobal* | line ends *line_ends_options*]]]
[center_edge] [dc_layer *pattern_layer_num* …]
[exact_cg *cglobal*] [ignore_extent] [restricted]
[line_ends *line_ends_options*]

where the *line_ends_options* is

[layer *pattern_layer_num*]
[length1 *operator num* [*operator num*] [length2 *operator num* [*operator num*]]]
with_length *operator num* [*operator num*]
inside *inside_num* outside *outside_num*
[other_edges inside *inside_num* outside *outside_num*]
[allow_partial_edges]

Arguments

- -turbo [*number_of_processors*]

  An optional argument set that specifies to use multithreaded parallel processing. The argument set must be specified as shown in the syntax line, before the **cluster** argument.

  The optional value *number_of_processors* is a positive integer that specifies the number of CPUs to use for multithreaded processing.If *number_of_processors* is not specified, Calibre runs on the maximum number of CPUs available for which you have licenses. For best performance, it is recommended that you avoid specifying this value.

- -remote *host* [,*host* …] | -remotefile *filename*

  An optional argument choice that specifies to use Calibre MTflex multithreaded parallel processing. Specify either -remote or -remotefile. Must be specified -turbo.

  - -remote *host* [,*host* …] — Specifies the names of the remote hosts. You must specify at least one host with this option. To specify multiple hosts, use a comma to separate the host names.
  - -remotefile *filename* — Specifies the path to a configuration file containing remote host information. For details about the configuration file, see “About the Configuration File” in the *Calibre Administrator’s Guide*.

- **input *input_lib***

  A required argument that specifies the input pattern matching database (PMDB). The file extension (*.pmdb*) is optional.

- **output *cluster_lib***

  A required argument that specifies the name of the output PMDB file; this library includes the clustering information.

- compress *compress_library* [no_unique]

  An optional argument set that specifies to output a compressed library to the *compress_library* PMDB file. The compress option is useful to reduce the number of patterns in a library. The library *compress_library* can be used as you would any other pattern library.

  no_unique — Specifies to not include unique patterns (those that are not part of any cluster) in *compress_library*.

  A compressed library contains one “cluster pattern” for each cluster group, where the cluster pattern includes constraints such that each pattern in the cluster is represented. Each cluster pattern includes a property named Cluster_size that contains the number of patterns represented by the cluster pattern.

  Unique patterns (those that are not part of any cluster) are included by default but do not have a value for the Cluster_size property. Specify the cluster_unique option to assign a value of 1 to the Cluster_size property for unique patterns.

- pattern_start *num*

  An optional argument typically used only in testing that specifies the first pattern to process. The tool skips the first *num*-1 patterns and starts clustering with the *num* pattern. All patterns prior to *num* are ignored. All patterns after the *num* pattern are processed unless pattern_limit is specified.

  If pattern_limit *count* is specified, only *count* patterns are processed. For example, consider a library with 1 million patterns. Skip the first 100,000 patterns and cluster the next 100,000 patterns, by specifying the following:

  ```
  pattern_start 100001 pattern_limit 100000
  ```

- pattern_limit *count*

  An optional argument that specifies the maximum number of patterns to process. All remaining patterns are ignored by the cluster utility. This argument is typically used only in testing.

- prop_name *name*

  An optional argument that specifies the name of the property used to identify a clustered pattern in the library. If not specified, the default name is “cluster”.

- cluster_unique

  An optional argument that specifies to add a cluster property (see prop_name) with a value of zero to all unique patterns. By default, unique patterns are not assigned a value for the cluster property.

  When the cluster_unique and compress options are both specified, a value of 1 is assigned to the Cluster_size property for unique patterns.

- allowed_shift *shift_val*

  An optional argument that allows the alignment of patterns to shift by the specified amount when determining similarity. By default, the shift value is zero.

  This example allows patterns to shift relative to other patterns for similarity comparison up to 0.07 um:

  ```
  pdl_lib_mgr cluster ... allowed_shift 0.07
  ```

- center_halo *halo_size* [crop] [exact | exact_cg *cglobal* | line_ends *line_ends_options*] [outer_halo [exact | exact_cg *cglobal* | line_ends *line_ends_options*]]

  An optional argument set that specify a center halo region and the matching behavior for the center and outer halo regions.

  halo_size — The perpendicular distance from the pattern center point to the edge of a square center halo region. Everything outside the center halo region is termed the outer halo.

  crop — An optional parameter that specifies *only* the center halo region is used for matching. The outer halo region is ignored (cropped) when matching patterns to determine clustering. Matching conditions apply to the center halo region. The crop option may not be specified with outer_halo.

  exact | exact_cg *cglobal* | line_ends *line_ends_options* — Optional parameters that specify matching conditions for the different halo regions. The exact option specifies exact matching, which is the default for both the center and outer halo regions. The line_ends and exact_cg options are defined elsewhere in this reference topic. When neither crop nor outer_halo is specified, matching conditions apply to the *outer* halo region; otherwise the conditions apply to the halo region given by the keyword preceding the condition.

  outer_halo — Specify outer_halo in order to specify different matching conditions for the center and outer halo regions. May not be specified with crop.

  The allowed syntax, logic for applying the matching conditions, and examples are given in “center_halo and outer_halo Options” on page 215 in the “Description” section.

- center_edge

  An optional argument that optimizes clustering for line-end hotspots. The center_edge argument only considers shapes on the first pattern layer. When center_edge is specified, patterns with edges at or near the center of the pattern are placed in a different cluster than

patterns without an edge near the center. The exact behavior depends on the presence of the line_ends or exact_cg argument:

- With exact_cg — Patterns with edges within half the *cglobal* value of the pattern center are placed in a different cluster from patterns with no edge near the center.
- With line_ends — Patterns with edges that intersect exact center of the pattern are placed in a different cluster from patterns with no edge at the center.

See "Example 3" on page 217.

- dc_layer *pattern_layer_num …*

  An optional argument set that specifies one or more layers as Don't Care layers, which are ignored during the clustering process. Pattern layers that are specified as Don't Care layers in the library are always ignored by the clustering process.

  For example, to only cluster according to the shapes on the first pattern layer in a library with three layers, specify "dc_layer 2 3".

- exact_cg *cglobal*

  An optional argument that causes exact patterns to be compared for similarity within the specified cglobal variability value to exact-match patterns. Only exact patterns are compared with added tolerance while variable patterns are compared as is. The maximum recommended value is less than a quarter of the minimum space or width.

  Note

  The exact_cg option may not be combined with the line_ends option. If you use the center_halo option, see "center_halo and outer_halo Options" on page 215.

  This example allows pattern layers and edges to vary by 0.004 um between patterns while comparing similarity:

  ```
  pdl_lib_mgr cluster ... exact_cg 0.004
  ```

- ignore_extent

  An optional argument that enables patterns with different extents to cluster together, grouping patterns with similar geometric pattern layers regardless of extent differences.

- restricted

  An optional argument that prevents the maximum edge difference across clustered patterns from exceeding the exact_cg or line_ends constraint values.

- line_ends [layer *pattern_layer_num*] inside *inside_num* outside *outside_num*
  with_length *operator num* [*operator num*]
  [length1 *operator num* [*operator num*] [length2 *operator num* [*operator num*]]]
  [other_edges inside *inside_num* outside *outside_num*] [allow_partial_edges]

  An optional argument set that enables you to adjust some parameters used in finding similar patterns. In this case, a set of rules is used to select pattern line ends that are variable in some way.

The line_ends option may not be combined with the exact_cg option. If you use the center_halo option, also see “center_halo and outer_halo Options” on page 215.

layer *pattern_layer_num* — An optional argument set that specifies which pattern layer number the rules apply to. When not specified, the rules apply to all pattern layers.

inside *inside_num* — A required line_ends argument set that defines how far a line end can move to the inside.

outside *outside_num* — A required line_ends argument set that defines how far a line end can move to the outside.

operator num — Arguments used with length1, length2 and with_length to specify a constraint. The *operator* argument supports the operators listed in the following table. You must quote or escape constraint operators to avoid operating system errors. *num* is a numeric value.

Table 5-1. Cluster Constraint Operators

Definition	Operator
Equal	==
Less than	<
Less than or equal	<=
Greater than	>
Greater than or equal	>=
Not equal	!=

with_length *operator num* [*operator num*] — A required line_ends argument set that defines the required length of the line-end edge.

length1 *operator num* [*operator num*] — An optional argument set that defines the required length of one side of the line end. If length2 is not specified, the constraint applies to only one side of the line end.

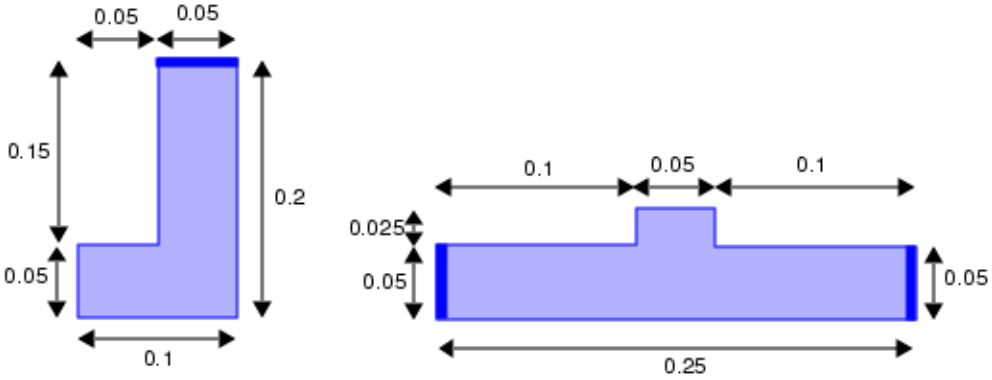
length2 *operator num* [*operator num*] — An optional argument set that defines the required length of the second side of the line end.

As an example of the length1 and length2 options, consider the following line_ends values:

```
line_ends layer 1 with_length "==" 0.05
                      length1 ">=" 0.1 length2 ">=" 0.1
```

The patterns in Figure 5-1 contain a total of three line ends that meet the criteria.

Figure 5-1. cluster line_ends



other_edges inside *inside_num* outside *outside_num* — An optional line_ends argument set that defines movement of all non-line end edges in the pattern.

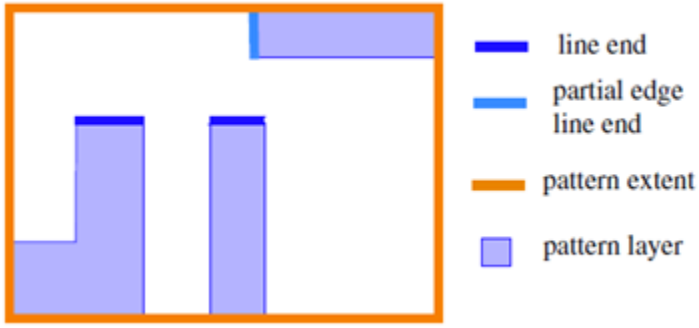
Note

A significant runtime penalty occurs if large values are used for the other_edges settings. For performance reasons, the *inside_num* and *outside_num* values are recommended to be no more than 0.004 *or* less than one-fourth of the minimum space or width, whichever is less.

allow_partial_edges — An optional line_ends argument that causes partial edges to be considered as line ends if they meet the line_ends criteria. A partial edge has one or two vertices that lie on the pattern extent and has at least one adjacent edge that is a virtual edge. Refer to “Edges in Patterns” on page 28.

The following figure has three line ends if allow_partial_edges is specified, but only two line ends without allow_partial_edges.

Figure 5-2. cluster allow_partial_edges



Description

The cluster utility identifies similar patterns in a pattern library. The utility copies patterns from the input library to the output library. Similar patterns receive a property with the number of the group the pattern belongs to. Similarity in patterns is defined as the edge-position difference between two or more patterns. Tolerance for similar patterns is determined by line-end variability, shifted extents, and global constraint (cglobal) variability. BCM and XBAR patterns are ignored by this utility, and they are passed through without being clustered.

By default unique patterns (those not part of a cluster) are not assigned a value for the cluster property. This can be changed with the cluster_unique argument.

By default the cluster property is named “cluster”. You can change the property name with the prop_name argument.

The write_oasis utility can use the clustering property information to output similar patterns in a group hierarchy.

center_halo and outer_halo Options

The center_halo and outer_halo options allow you to specify a center and outer halo region and differing matching conditions for the two regions. The *halo_size* value specifies the perpendicular distance from the pattern center point to the edge of a square center halo region. Everything outside the center halo region is termed the outer halo.

The allowed syntax permutations with behavior are defined as follows:

center_halo *halo_size* [exact | exact_cg *cglobal* | line_ends *line_ends_options*]

> Only center_halo is specified, with optional matching conditions. Exact matching is used in the center halo region. If an optional matching condition is specified, it apples to the *outer* halo region.

center_halo *halo_size* crop [exact | exact_cg *cglobal* | line_ends *line_ends_options*]

> The options center_halo and crop are specified, with optional matching conditions. Matching takes place *only* in the center halo region; the outer halo region is cropped out for purposes of matching. If specified, the exact, exact_cg, or line_ends options apply to the *center* halo region. Exact matching (the default) is used in the center_halo region if no matching condition is specified.

center_halo *halo_size* [exact | exact_cg *cglobal* | line_ends *line_ends_options*]
outer_halo [exact | exact_cg *cglobal* | line_ends *line_ends_options*]

> Both center_halo and outer_halo are specified, with optional matching conditions. The first condition applies to the center_halo region. The second condition applies to the outer halo region. Exact matching is the default for both regions if no matching condition is specified. In order to specify a matching condition other than exact for the center_halo region when not using crop, you must specify both center_halo and outer_halo.

All edges crossing the center_halo region boundary are compared according to the matching criteria of the center_halo region.

This following figures give some examples. In all cases the two patterns are clustered together.

Figure 5-3. Cluster center_halo with exact_cg

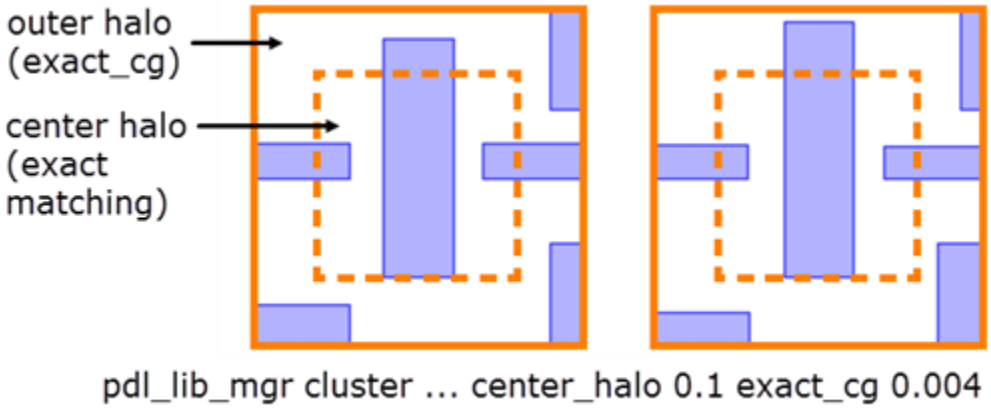


Figure 5-4. Cluster center_halo with crop

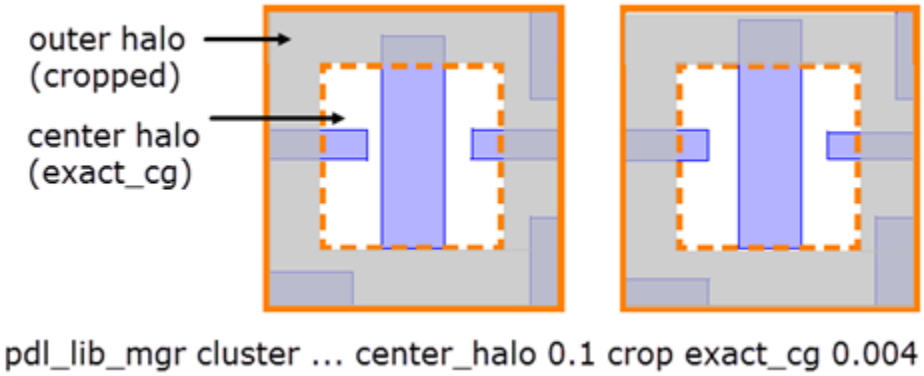
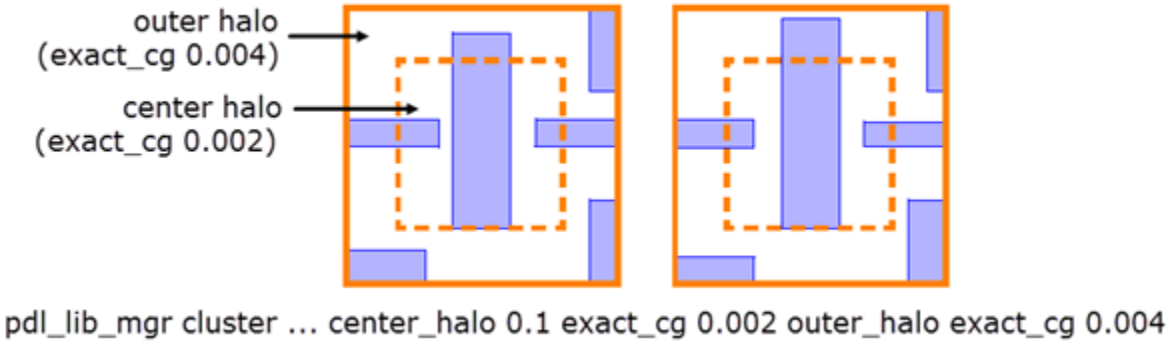


Figure 5-5. Cluster center_halo and outer_halo



Examples

Example 1

This example selects all edges from all pattern layers with an abutting edge length of more than 0.05 um and a second abutting edge length of 0.07 um or more, with a line end length equal to

0.05 um and applies a constraint outside by 0.1 um and all other real edges are constrained inside by 0.001 um and outside by 0.001 um.

```
pdl_lib_mgr cluster input mix.pmdb output mix_post.pmdb
   line_ends length1 ">" 0.05 length2 ">=" 0.7 with_length "==" 0.05
   inside 0.1 outside 0.1 other_edges inside 0.001 outside 0.001
```

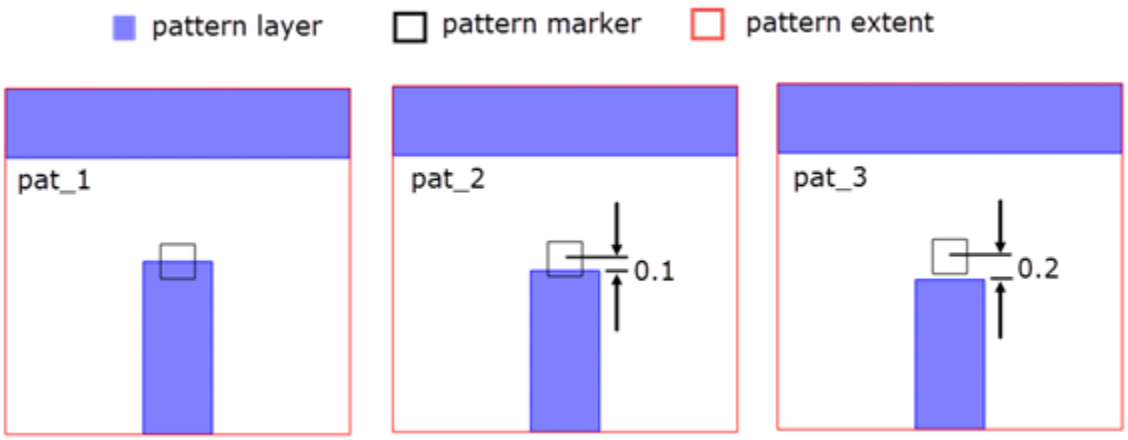
Example 2

This example selects all pattern layer 1 edges between 2 real vertices with a length less than 0.05 um and applies a constraint inside by 0.01 um and outside by 0.05 um. You must quote or escape the constraint operators to avoid Linux errors.

```
pdl_lib_mgr cluster input mix.pmdb output mix_post.pmdb line_ends layer 1
   with_length "<" 0.05 inside 0.01 outside 0.05
```

Example 3

This example shows the effect of using the center_edge argument. The pattern library *lib1.pmdb* has the three patterns shown in the following figure. The pattern marker is a square of width 0.3 user units at the center of the pattern.



center_edge and exact_cg:

```
pdl_lib_mgr cluster input lib1.pmdb output lib1_ce_cg.pmdb \
   exact_cg 0.2 center_edge
```

Only pat_1 and pat2 have edges within 0.1 (half of the exact_cg value) of the pattern center, so they are not clustered with pat_3. pat_1 and pat_2 are placed in the same cluster, while pat_3 is not in a cluster.

center_edge and line_ends:

```
pdl_lib_mgr cluster input lib1.pmdb output lib1_ce_le.pmdb \
   line_ends with_length "<" 0.8 inside .15 outside .15 center_edge
```

Only pat_1 has an edge that intersects the pattern center, so it is not clustered with pat_2 and pat_3. pat_2 and pat_3 are placed in the same cluster, while pat_1 is not in a cluster.

Without the center_edge argument, both the preceding examples cause all three patterns to be placed in the same cluster.

compare

Compares pattern libraries and determines if any patterns are duplicated. Only exact patterns are compared. A log file with results is generated by default. Optionally, output pattern libraries can be generated that contain common or unique patterns.

Usage

pdl_lib_mgr [-turbo [*number_of_processors*]]
compare input1 ***pmdb_1*** **input2** ***pmdb_2*** **log_file** ***log_filename***
[output_common *filename*] [output_1only *filename*] [output_2only *filename*]
[ignore_layer_extents] [ignore_markers] [ignore_regions] [compare_orientation]

Arguments

- [-turbo [*number_of_processors*]]

  An optional argument set that specifies to use multithreaded parallel processing. The argument set must be specified as shown in the syntax line, before the **compare** argument.

  The optional value *number_of_processors* is a positive integer that specifies the number of CPUs to use for multithreaded processing. If *number_of_processors* is not specified, Calibre runs on the maximum number of CPUs available for which you have licenses. For best performance, it is recommended that you avoid specifying this value.

- **input1** ***filename***

  A required argument set that specifies the first input PMDB file.

- **input2** ***filename***

  A required argument set that specifies the second input PMDB file.

- **log_file** ***log_filename***

  A required argument set that specifies the name of the log file. The log file shows comparison results and any warnings or error messages.

- output_common *filename*

  An optional argument set that specifies to create a PMDB file containing only patterns that are duplicated, or common, between ***pmdb_1*** and ***pmdb_2***. Common patterns must be exact duplicates, otherwise each is considered unique. When written to the specified output file, duplicate patterns receive a super-set of matching pattern attributes including properties, keys, and cglobal values. The following rules apply for duplicate patterns written to the output file:

Pattern Feature	Rule
Pattern Polygons	All pattern polygons must be geometrically identical. If a pattern specifies allowed pattern orientations, the definition of a duplicate or unique pattern depends on the compare_orientations argument; see “Pattern Orientation Handling” on page 221.

Pattern Feature	Rule
Pattern Name	A common pattern retains the name specified in the ***pmdb_1*** pattern.
Layer Name	A common pattern retains the layer names specified in the ***pmdb_1*** library.
Properties	A common pattern inherits a cumulative list of properties from the duplicate patterns in ***pmdb_1*** and ***pmdb_2***. The smallest property value is saved. Duplicate patterns sharing property names with different property types are not considered common and are saved as unique patterns.
Keys	A common pattern inherits a cumulative list of keys from the duplicate patterns in ***pmdb_1*** and ***pmdb_2***.
Cglobal	A common pattern retains the larger value of cglobal from duplicate patterns.

- output_1only *filename*

  An optional argument set that specifies to create a PMDB file containing only patterns unique to ***pmdb_1***.

- output_2only *filename*

  An optional argument set that specifies to create a PMDB file containing only patterns unique to ***pmdb_2***.

- ignore_layer_extents

  An optional argument that specifies to ignore defined layer extents while comparing patterns. The extents defined for the first pattern always go to the output database. This only applies to layer extents, not the pattern extent.

- ignore_markers

  An optional argument that specifies to ignore defined markers while comparing patterns. The markers defined for the first pattern are always used in the output database.

- ignore_regions

  An optional argument that specifies to ignore defined regions while comparing patterns. The regions defined for the first pattern are always used in the output database.

- compare_orientation

  An optional argument that specifies how to consider the allowed orientations of the input patterns. This argument affects how duplicate and unique patterns are categorized, and affects what patterns are saved by the output_common, output_1only, and output_2only arguments. The behavior is described in “Pattern Orientation Handling” on page 221.

Description

The compare utility compares two pattern libraries and determines if any patterns are duplicated between the two libraries. Patterns with constraints are not compared. Patterns geometries must match exactly for two patterns to be identified as duplicate. In addition to the pattern layer polygons, the pattern and layer extents, regions, and markers are also compared, subject to optional arguments. The input libraries must have the same number of layers. Comparison results are written to ***log_filename***.

For comparing large libraries, see "Calibre Pattern Matching Reference Library Flow" on page 200 for a more efficient method.

Pattern Orientation Handling

The allowed pattern orientations for a match are considered when comparing patterns. The allowed pattern orientations are specified on the **Attributes** tab in the GUI. The compare behavior differs depends on whether compare_orientation is specified.

- **Default**
  - Two patterns are considered *duplicates* if they result in the *same* matches.

    If pattern geometries are the same but the drawn orientation is different, two patterns are duplicates if the pattern orientation values result in the same pattern matches. The log file notes if the duplicate determination included consideration of the pattern orientation (the orient_detail value). The pattern in ***pmdb_1*** is saved to the output_common library.
  - Two patterns are considered *unique* if they result in *different* matches.
- **With compare_orientation**

  Patterns with identical geometries, but different drawn orientations and/or pattern orientations are decomposed into individual patterns for each orientation and then analyzed. This happens in two stages:

  a. First, within each input library, patterns with identical geometries but different drawn orientations and/or pattern orientations are merged into one pattern, with appropriate settings for pattern orientation.

  b. Next, the two pre-processed libraries from the previous step are compared. Patterns with identical geometries, but different drawn orientations and/or pattern orientations are decomposed into individual patterns for each orientation and then analyzed. The duplicate and unique portions are noted in the log file for each pattern orientation. For each pattern orientation, the duplicate and unique patterns are saved to the appropriate library specified by the output_common, output_1only, and output_2only arguments.

  More duplicate patterns are identified when compare_orientation is specified.

See "Example 3" on page 223 and "Example 4" on page 224.

Examples

Example 1

This example invokes pdl_lib_mgr compare and outputs the common patterns between two libraries. The input *knownLibrary.pmdb* is shown in Figure 5-6. The input *newLibrary.pmdb* is shown in Figure 5-7.

The following command writes the common patterns between the two input libraries to *duplicate_patterns.pmdb*:

```
pdl_lib_mgr compare input1 knownLibrary.pmdb input2 newLibrary.pmdb \
   log_file common.log output_common duplicate_patterns.pmdb
```

Figure 5-6. *knownLibrary.pmdb*

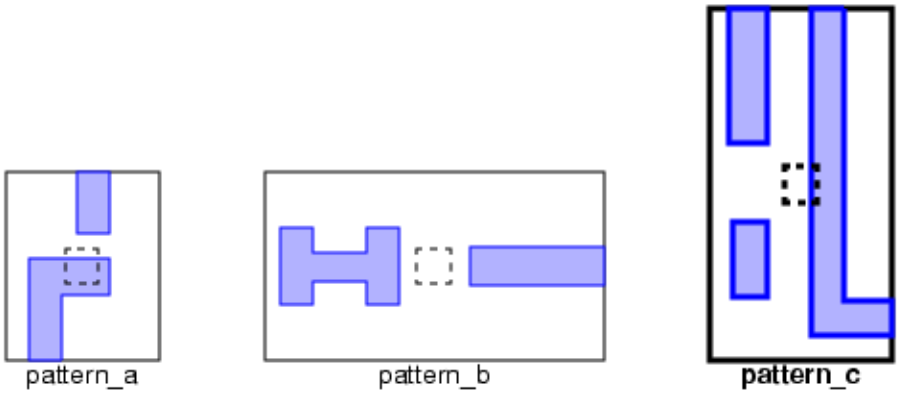
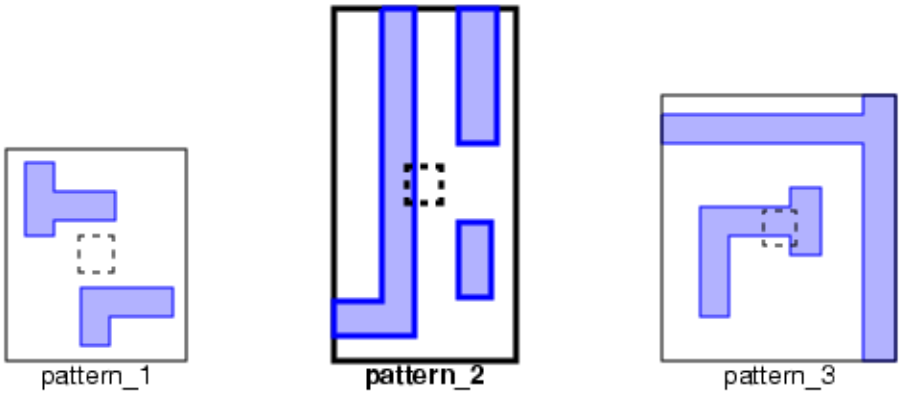


Figure 5-7. *newLibrary.pmdb*



The output pattern library *duplicate_patterns.pmdb* contains pattern_c, because it is duplicated in *newLibrary.pmdb* as pattern_2. Comparison results are written to the text file *common.log*.

Example 2

This example invokes pdl_lib_mgr compare and outputs the unique patterns in the first input library (*knownLibrary.pmdb*).

Refer to Figure 5-6 and Figure 5-7 for the contents of the input pattern libraries.

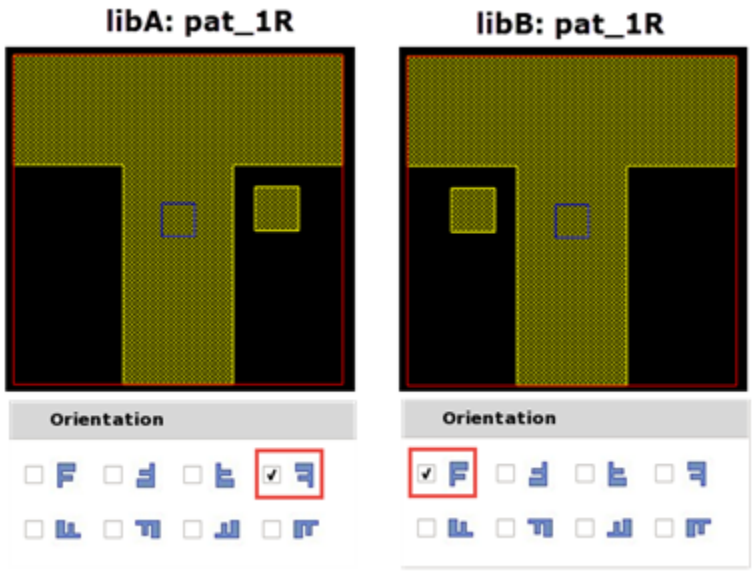
```
pdl_lib_mgr compare input1 knownLibrary.pmdb input2 newLibrary.pmdb \
   log_file unique1.log output_1only knownLibrary_unique.pmdb
```

This results in an output PMDB file that contains pattern_a and pattern_b, because these patterns exist in *knownLibrary.pmdb* but are not duplicated in *newLibrary.pmdb* .

A similar invocation with output_2only produces an output that contains only pattern_1 and pattern_3 from *newLibrary.pmdb*.

Example 3

Consider these two patterns, which are flipped across the y axis and have different pattern orientations:



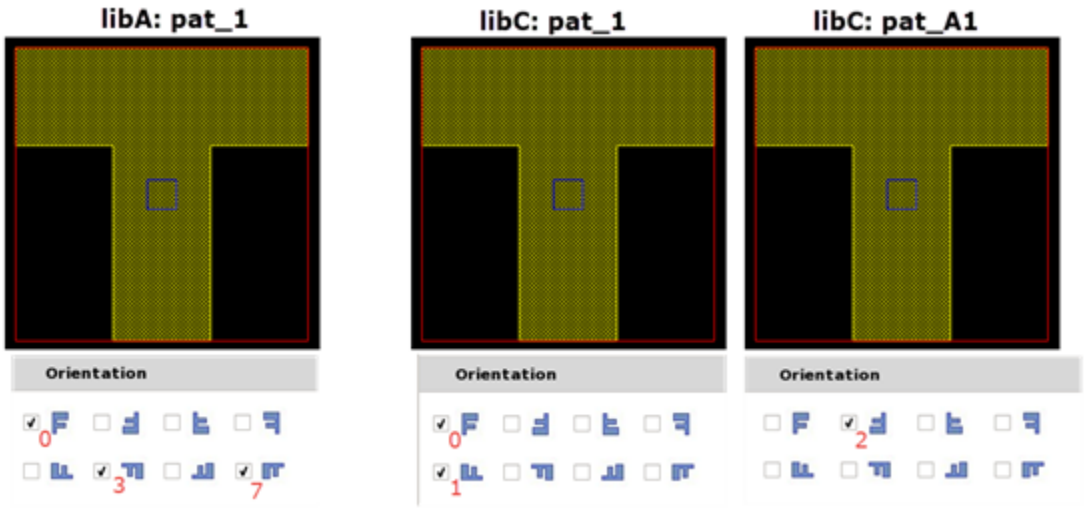
The patterns are duplicates when orientation is considered, because they produce the same pattern matches. This is true both for the default case and with compare_orientations specified. This message is printed to the log file:

```
Second_database::Pattern 'pat_1R' is duplicate of First_database::Pattern
'pat_1R'.
Second_database::Pattern 'pat_1R' and First_database::Pattern 'pat_1R'
have the same geometry and their orient detail will result in the same
matches.
```

The pattern in first library is saved to the library created with the output_common argument.

Example 4

Consider these patterns in libraries libA and libC, with different allowed pattern orientations. For convenience, the numeric pattern orient value is included in red font next to the orient checkbox.



Default Case

By default, the patterns are considered unique because they produce different matches. Nothing is saved to the library created by the output_common argument. This message is printed to the log file for the default case:

```
Second_database::Pattern 'pat_1' and First_database::Pattern 'pat_1' have
the same geometry but they have different orient details so they will
produce different matches.
Second_database::Pattern 'pat_A1' and First_database::Pattern 'pat_1' have
the same geometry but they have different orient details so they will
produce different matches.
First_database::Pattern 'pat_1' doesn't have a matching pattern in
Second_database.
Second_database::Pattern 'pat_1' doesn't have a matching pattern in
First_database.
Second_database::Pattern 'pat_A1' doesn't have a matching pattern in
First_database.
```

With compare_orientations

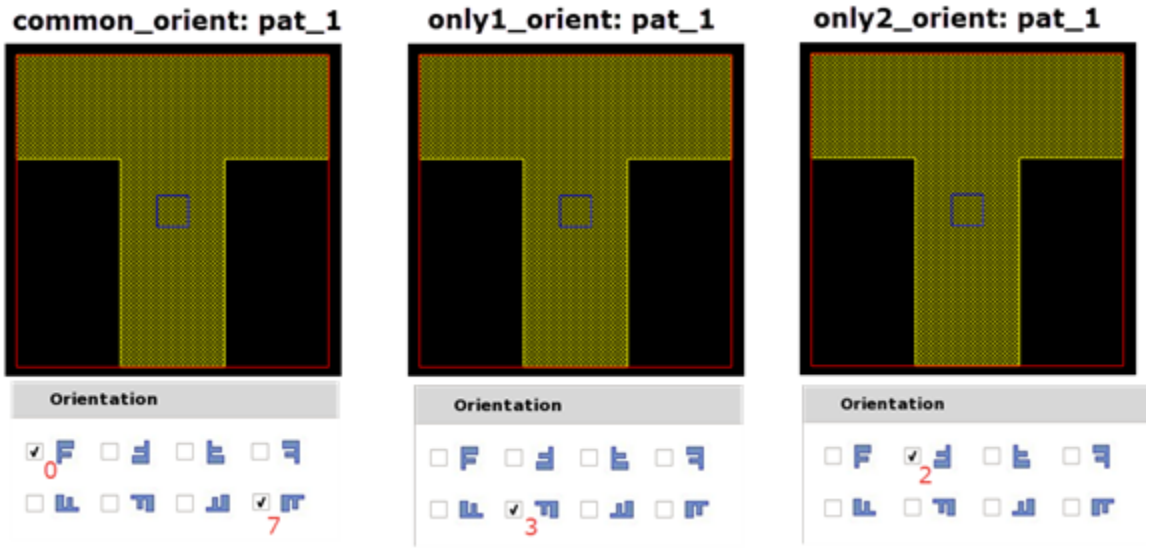
When compare_orientations is specified, first, the two patterns in libC are merged because they have the same geometry with different orientations. Next, the patterns in the input libraries are decomposed into separate patterns for each orientation and compared. The as-drawn orientation is a duplicate, while the other orientations are unique.

```
pdl_lib_mgr compare input1 libA.pmdb input2 libC.pmdb
   log_file log_orient output_common common_orient.pmdb
   output_1only only1_orient.pmdb output_2only only2_orient.pmdb
   compare_orientation
```

This is the output in the log file, where the orient detail values denote the orientations (see “Pattern Orientation” on page 75):

```
Second_database::Pattern 'pat_A1' is duplicate of Second_database::Pattern
  'pat_1'.
Second_database::Patterns 'pat_1' and 'pat_A1' have the same geometry so
  their orient detail will be merged.
Second_database::Pattern 'pat_1' is duplicate of First_database::Pattern
  'pat_1'.
First_database::Pattern 'pat_1' with orient detail 0 is the same as
  Second_database::Pattern 'pat_1' with orient detail 0.
First_database::Pattern 'pat_1' with orient detail 7 is the same as
  Second_database::Pattern 'pat_1' with orient detail 1.
First_database::Pattern 'pat_1' with orient detail: 3 is unique.
Second_database::Pattern 'pat_1' with orient detail: 2 is unique.
```

The libraries generated by the output_common, output_1only, and output_2only arguments have the patterns shown in the following figure. The two unique orientations of pat_1 are included in the only1_orient and only2_orient libraries. For patterns that have duplicates, one of the patterns is included in the common_orient library.



compile

Generates an SVRF DMACRO file from the input pattern library. The DMACRO file is the compiled pattern library that is used in a pattern matching run.

Note

The DMACRO format changed in releases 2019.2 and 2021.2. DMACRO files created with Calibre 2021.2 cannot be used by earlier versions of Calibre. DMACRO files created from 2019.2 to 2021.1 cannot be used in Calibre versions 2019.1 and earlier.

DMACRO files created prior to 2021.2 can be used in later versions of Calibre.

Usage

pdl_lib_mgr [-turbo [*number_of_processors*]]
compile input *filename* output *filename*
[marker *marker_name* {bbox | bbox10 | empty | matched | drawn} [layer *layer_name*]] ...
[maxjog *value*] [tilesize *value*] [password *password*] [enable_runtime_options]
[gen_count_layer] [gen_pat_names] [gen_all_markers] [exclude_error_patterns]
[epd_marker *epd_marker_name*] [patterns "*pat1* [*pat2* ...]"]

Arguments

- -turbo [*number_of_processors*]

  An optional argument set that specifies to use multithreaded parallel processing. The argument set must be specified as shown in the syntax line, before the **compile** argument.

  The optional value *number_of_processors* is a positive integer that specifies the number of CPUs to use for multithreaded processing. If *number_of_processors* is not specified, Calibre runs on the maximum number of CPUs available for which you have licenses. For best performance, it is recommended that you avoid specifying this value.

- **input *filename***

  A required argument that specifies the input pattern library.

- **output *filename***

  A required argument that specifies the name of the generated DMACRO file.

- marker *marker_name* {bbox | bbox10 | empty | matched layer *layer_name* | drawn layer *layer_name*}

  An optional argument set that defines a marker name (*marker_name*) and marker type to output to the compiled DMACRO library. This argument set can be used to enable compilation of a library in which all patterns do not have the same set of markers.

  The marker argument set can be used both to specify which markers in the input PMDB are output to the compiled library and to add new markers. If a pattern in the PMDB has the marker *marker_name*, the marker is output to the compiled library with the marker type

used in the PMDB. If a pattern does not have the marker *marker_name*, the marker is added with the specified *marker_type* (also termed the “backup type”).

The exact behavior of this argument set depends on the set of markers present in the PMDB and whether gen_all_markers is specified. See “Compile-Time Markers” on page 230 in the “Description” section and “Example 3: marker and gen_all_markers” on page 232.

The allowed marker types are:

- bbox

  The output pattern marker is the minimum bounding box for the pattern layer, including the extent.

- bbox10

  The output pattern marker is approximately ten percent of the default extent.

- empty

  An empty marker generates no output on the marker layer for a pattern match.

- matched layer *layer_name*

  The output pattern marker is the matched geometries on *layer_name*.

- drawn layer *layer_name*

  The output pattern marker is the pattern geometries on *layer_name*.

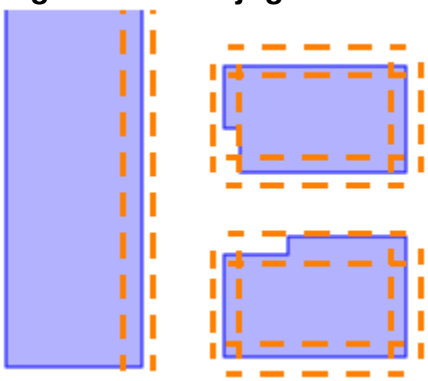
Note

For exact matches, the drawn_layer marker is the same as the matched_layer marker.

- maxjog *value*

  An optional argument that enables the tool to ignore jogs equal to or less than the specified tolerance value. The maximum value for maxjog is two times the global constraint (2*cglobal). The *value* argument is specified in microns. See “Jog Tolerance in Pattern Matching” on page 76.

Figure 5-8. maxjog Removal



Note

The maxjog argument should only be used when generated patterns all contain the cglobal parameter with the same value.

- tilesize *value*

  An optional argument set specifies the tile size used by the tool during the pattern matching. The minimum *value* for tilesize is 30. The *value* argument is a positive integer specified in microns, and the default tile size is automatically calculated if the -tilesize option is not used. If the tilesize *value* entered is smaller than either 30 or the minimum calculated tilesize required, then a warning is generated and the user-defined tile size is ignored in order to prevent pattern matches from being missed.

- password *password*

  An optional argument set that specifies a password, where *password* is a string containing one or more alphanumeric and underscore characters. The password must be provided when specifying runtime options with the CMACRO options string. See the -password argument in “CMACRO for Calibre Pattern Matching” on page 177.

  If the password argument is specified, enable_runtime_options is automatically used during library compilation.

- enable_runtime_options

  An optional argument that enables runtime options. Runtime options are specified in the CMACRO options string.

  If the CMACRO options string is non-empty and enable_runtime_options was not specified when the pattern library was compiled, the following error occurs: ERROR: CMACRO (pm:<*lib*>) - The DMACRO is protected. This option is not allowed to be overridden as per the macro developer's specification.

- gen_count_layer

  An optional argument that causes count markers to be added to the compiled patterns. The marker layer name is count_marker.

  Count markers enable you to determine the number and location of matches from a pattern matching run. See “Count Markers” on page 45 for information.

- gen_pat_names

  An optional argument that adds pattern names in the DMACRO comment section. When the DMACRO is generated, the names of all patterns in the compiled library are listed in the comment section before the definition of the DMACRO. For example:

  ```
  // Patterns to search for: pat_1, pat_2
  ```

- gen_all_markers

  An optional argument that causes the complete set of all markers in the input library to be output for all patterns. This option can be used to enable library compilation when all patterns in a library do not have the same marker set.

The complete set of markers defined across all patterns is output. If a marker is missing in a particular pattern, the marker is added with the type empty (the default backup type).

The gen_all_markers option can be used with the marker argument set to specify new markers and to specify a different backup type for existing markers. See "Compile-Time Markers" on page 230 in the "Description" section for details and "Example 3: marker and gen_all_markers" on page 232.

- exclude_error_patterns

  An optional argument that causes the tool to good patterns to be compiled into the DMACRO, while patterns with errors are noted as errors and are not included.

  Note

  If the exclude_error_patterns option is not set, a library with one or more error patterns fails to compile.

  If patterns in the library have different marker sets and gen_all_markers is not specified, the library does not compile even when exclude_error_patterns is specified.

- epd_marker *epd_marker_name*

  An optional argument that specifies the marker layer to use for the string property "EPD_string", which reports edge movement. This option is required if edge movement reporting is specified with -epd_string in the CMACRO options string. The *epd_marker_name* can be any of the following:

  - A marker that exists in the input pattern library. The marker must exist on all patterns.
  - A compile-time marker that is generated with the marker option.
  - The compile-time marker count_marker that is generated with the gen_count_layer option.

  For more information regarding the EPD_string property, see the -epd_string option in "CMACRO for Calibre Pattern Matching" on page 177.

- patterns "*pat1* [*pat2* ...]"

  An optional argument that specifies a subset of patterns to compile. Only the specified patterns are included in the SVRF DMACRO output file; other patterns are ignored. Only the specified patterns are considered when running other options. If a specified pattern does not exist in the library, a warning is issued. The list of patterns must be enclosed in quotes.

Description

The pdl_lib_mgr compile utility reads a pattern library (PMDB) file and produces an SVRF DMACRO output file. See "DMACRO and CMACRO for Pattern Matching" on page 79.

An error is generated if the DMACRO generated from a compile command would have only markers of type empty, as such a library never generates any output. To avoid this possibility

you can specify the marker argument set with at least one *marker_type* other than empty. Also see “Example 4: DMACRO Library With Only the Count Markers and Empty Markers” on page 233 for a relevant example.

For default operation, with no optional arguments specified, all patterns in the PMDB must have the same marker set in order for the library to compile. Compile-time markers can be used to manage this restriction, as explained in the next section.

Compile-Time Markers

If the PMDB does not have the same marker set for all patterns you must specify either gen_all_markers or include the marker argument set. The markers that are output to the compiled DMACRO library depend on the set of markers present in the input PMDB and the marker, gen_count_layer, and gen_all_markers arguments. The general behavior is described here. See “Example 3: marker and gen_all_markers” on page 232 for specific examples.

The following behavior takes place in all cases:

- marker argument set — If the marker *marker_name* exists in a pattern it is output with the marker type used in the PMDB. If the marker *marker_name* does not exist in a pattern it is added with the specified *marker_type* (the backup type).
- gen_all_markers — The default backup type is empty; that is, if a marker is missing in a particular pattern, the marker is added with the type empty. However, if the marker argument set is specified, the specified *marker_type* is used as the backup type for *marker_name*.

Same Markers Exist in All Patterns of the PMDB

- Default (no options)

  All markers are output.

- gen_count_layer

  All markers plus the count_marker layer are output.

- gen_all_markers

  All markers are output.

- {marker *marker_name marker_type* [layer *layer_name*]}… [gen_count_layer]

  Only the specified markers are output.

  If gen_count_layer is specified, the count_marker layer is also output.

- {marker *marker_name marker_type* [layer layer_name]}… gen_all_markers [gen_count_layer]

  All markers are output due to the presence of the gen_all_markers option.

If gen_count_layer is specified, the count_marker layer is also output.

Differing Markers Exist in the Patterns of the PMDB

- Default (no options)

  Error. The library does not compile.

- gen_count_layer

  Error. The library does not compile.

- gen_all_markers

  The complete set of all markers in the input PMDB is output to the compiled library.

- {marker *marker_name marker_type* [layer *layer_name*]}… [gen_count_layer]

  Only the specified markers are output.

  If gen_count_layer is specified the count_marker layer is also output.

- {marker *marker_name marker_type* [layer layer_name]}… gen_all_markers [gen_count_layer]

  The complete set of all markers in the input PMDB is output to the compiled library. If the marker argument specifies a new marker, it is added.

  If gen_count_layer is specified the count_marker layer is also output.

Examples

Example 1: Basic Operation

This example converts a previously-generated PMDB file into a DMACRO file for use in a pattern matching run.

```
$ pdl_lib_mgr compile input batch_patterns.pmdb output dmacro.svrf
```

Example 2: gen_pat_names

The gen_pat_names option specifies to include the compiled pattern names in the DMACRO file. The pattern names are included in the comment section before the DMACRO definition. For example:

```
$ pdl_lib_mgr compile input lib_a.pmdb output lib_a.dmacro gen_pat_names
```

The resulting DMACRO has these lines for a library with two compiled patterns named pat_1 and pat_2.

```
// Calibre PDL compiler v2017.3 Fri Sep 1 02:57:45 PDT 2017

// Patterns to search for: pat_1, pat_2

// PMDB library last modified Mon Mar 27 10:36:19 2017
// DMACRO generated Fri Sep  1 10:03:58 2017

DMACRO pm:lib_a in_layer1 in_layer2
 options_string
 out_Marker {
#DECRYPT<encrypted block>
#ENDCRYPT
```

Example 3: marker and gen_all_markers

This example demonstrates the how to use gen_all_markers and the marker argument set to compile a PMDB in which the patterns have differing markers sets. See “Compile-Time Markers” on page 230 for a general explanation of these options.

Assume the input PMDB has these two patterns with the indicated markers:

- Pattern_1: Marker_A (bbox) and Marker_B (matched)
- Pattern_2: Marker_A (bbox)

This command line is used:

```
pdl_lib_mgr compile input lib.pmdb output lib.dmacro <options>
```

The following table gives the results for different <options> specifications.

<options>	**Result**
None	Error, because the two patterns do not have the same markers.
marker Marker_A empty	Pattern_1: Marker_A (bbox) Pattern_2: Marker_A (bbox) Only Marker_A is output. The backup type of empty is not used because Marker_A exists for both patterns in the PMDB.
marker Marker_B empty	Pattern_1: Marker_B (matched) Pattern_2: Marker_B (empty) Only Marker_B is output. The backup type of empty is used in Pattern_2 because Marker_B does not exist for Pattern_2 in the PMDB. Marker layers of type empty do not generate output from a pattern matching run, so Pattern_2 never generates output.

<options>	Result
marker Marker_B bbox marker Marker_C bbox10	Pattern_1: Marker_B (matched) and Marker_C (bbox10) Pattern_2: Marker_B (bbox) and Marker_C (bbox10) Marker_B and Marker_C are output. The backup type of bbox for Marker_B is used in Pattern_2 because Marker_B does not exist for Pattern_2 in the PMDB. Marker_C does not exist in either pattern, so it is added with the specified type.
gen_all_markers	Pattern_1: Marker_A (bbox) and Marker_B (matched) Pattern_2: Marker_A (bbox) and Marker_B (empty) Each pattern in the DMACRO has the complete set of markers. Pattern_2 in the PMDB does not have Marker_B, so it is added with the default backup type of empty.
gen_all_markers marker Marker_B bbox	Pattern_1: Marker_A (bbox) and Marker_B (matched) Pattern_2: Marker_A (bbox) and Marker_B (bbox) Each pattern in the DMACRO has the complete set of markers. Pattern_2 in the PMDB does not have Marker_B, so it is added with the specified backup type of bbox.
gen_all_markers marker Marker_A (bbox10) marker Marker_C (bbox10)	Pattern_1: Marker_A (bbox), Marker_B (matched), Marker_C (bbox10) Pattern_2: Marker_A (bbox), Marker_B (empty), Marker_C (bbox10) Each pattern in the DMACRO has the complete set of markers plus the added Marker_C. Pattern_2 in the PMDB does not have Marker_B, so it is added with the default backup type of empty. Marker_C is added to both patterns with the specified type.

Example 4: DMACRO Library With Only the Count Markers and Empty Markers

You can use the compile-time marker specification to generate a DMACRO library with only the count_marker layer and empty markers. Such a library generates only the count marker output in a pattern matching run.

```
pdl_lib_mgr compile input lib.pmdb output lib.dmacro
   marker junk empty gen_count_layer
```

where the marker junk does not exist in the PMDB.

It is instructive to understand why other combinations of options do not accomplish the same result. These cases are discussed next.

If the PMDB does not have any markers named junk with a type other than empty, the following command produces an error because it is not allowed to compile a library that has only empty markers.

```
pdl_lib_mgr compile input lib.pmdb output lib.dmacro marker junk empty
```

The following command only succeeds if the patterns in the PMDB all have the same marker sets. When it does succeed all markers in the PMDB are output, in addition to the count_marker layer.

```
pdl_lib_mgr compile input lib.pmdb output lib.dmacro gen_count_layer
```

If a the patterns in a PMDB do not all have the same marker set, the gen_all_markers can be used to enable the compilation to succeed. However, the gen_all_markers causes all markers in the PMDB to be output for all patterns. The following command causes all pattern markers to be output, plus the new marker junk with type empty, plus the count_marker layer.

```
pdl_lib_mgr compile input lib.pmdb output lib.dmacro
   marker junk empty gen_all_markers gen_count_layer
```

convert

Generates new pattern matching libraries (PMDBs) from libraries created with older versions of the Calibre Pattern Matching tool or saves the library in the Pattern Matching Reference Library (PMRL) format.

Usage

Syntax 1

pdl_lib_mgr [-turbo [*number_of_processors*]]
convert input *filename* output *filename*
[precision *value*] [magnify *value*]
[property *propname* property_type {float | integer}] …
[{add_layer *layer_name* {dc | empty} [*layer_index*] }
| {delete_layer *layer_name* [*layer_name* …] }]

Syntax 2

pdl_lib_mgr [-turbo [*number_of_processors*]]
convert input *filename* output *filename* write_pmrl

Arguments

- [-turbo [*number_of_processors*]]

  An optional argument set that specifies to use multithreaded parallel processing. The argument set must be specified as shown in the syntax line, before the **convert** argument.

  The optional value *number_of_processors* is a positive integer that specifies the number of CPUs to use for multithreaded processing. If *number_of_processors* is not specified, Calibre runs on the maximum number of CPUs available for which you have licenses. For best performance, it is recommended that you avoid specifying this value.

- **input *filename***

  A required argument that specifies the input pattern library.

- **output *filename***

  A required argument that specifies the output filename.

- precision *value*

  An optional argument set that specifies the precision of the output pattern library. See the Precision statement in the *SVRF Manual*. If not specified, the precision of the input library is used.

  If the precision setting causes pattern geometries to be truncated, a warning is issued and the pattern is not included in the output library.

- magnify *value*

  An optional argument and value by which to scale the converted pattern library. For example, specifying 1.5 increases the size of the pattern, extent, markers, and so forth, by 50 percent. Specify *value* as a floating point number.

If the magnify setting causes pattern geometries to be truncated, a warning is issued and the pattern is not included in the output library.

- property *propname* property_type {float | integer}

  An optional argument set that specifies the property type for the property *propname* in the output library. This argument can be used to convert a property to the specified output type. When converting a floating-point value to an integer, the value is rounded and a warning message is printed. The argument set can be specified multiple times.

- add_layer *layer_name* {dc | empty} [*layer_index*]

  An optional argument set that adds a layer to the input library.

  layer_name — Specifies the name of the new layer.

  {dc | empty} — Specifies the layer as a Don't Care pattern layer (dc) or an empty layer.

  layer_index — Optional argument that specifies the layer index of the new layer, where index values start at 1. By default the new layer is added at the end of the layer list.

  For example, if the input library has two layers, the new layer has index 3 by default. If you specify *layer_index* as 2, the new layer has index 2 and the original second layer now has index 3.

- delete_layer *layer_name* [*layer_name* …]

  An optional argument set that deletes one or more layers from the input library. The layers to delete are specified by name.

  Note

  You cannot specify both add_layer and delete_layer at the same time.

- **write_pmrl**

  A required argument in Syntax 2 that specifies the output file is saved in the PMRL format. The default output format is the PMDB format.

Description

In Syntax 1, the utility converts pattern libraries created with older versions of the Calibre Pattern Matching tool to the current PMDB format.

The add_layer argument is useful if you want to add Don't Care pattern layers to a library in preparation for merging one or more libraries that do not have the same number of layers. See "Don't Care Pattern Layers" on page 55 and the merge utility.

Truncation and rounding effects can occur when magnifying a pattern, just as when magnifying a layout; see "Layout Magnification" in the *Calibre Solutions for Physical Verification* manual for a discussion of such effects. These effects can cause pattern matches to be missed unless the precision is also increased when scaling by fractional numbers. For example, when magnifying by 0.9, or 9/10, the precision should be increased by a factor of 10.

In Syntax 2 with the **write_pmrl** argument, the output library is saved in the Pattern Matching Reference Library (PMRL) format. See “Calibre Pattern Matching Reference Library Flow” on page 200 for details.

Note

The PMDB file format created from this utility is required for pattern matching in releases later than 2010.3.

Examples

Example 1

In this example, two integer properties are converted to float properties.

```
pdl_lib_mgr convert input pat1.pmdb output pat.pmdb property process
property_type float property iteration property_type float
```

Example 2

A library has patterns captured at a precision of 1000. You want to shrink the patterns by 90%, which is equivalent to a magnification of 0.9. To avoid grid snapping in the pattern, the pattern precision is increased to 10000. This concept is also true when using the Magnify layer operation.

```
pdl_lib_mgr convert input pat1.pmdb output pat.pmdb magnify 0.9
precision 10000
```

Example 3

This example converts the input PMDB to the PMRL format.

```
pdl_lib_mgr convert input std_lib.pmdb output ref_lib.pmrl write_pmrl
```

export_properties

Generates a comma-separated values (CSV) file that contains the property values defined for each pattern in the input pattern library.

Usage

pdl_lib_mgr export_properties input *pmdb* output *filename*
[{prop_name *name*} …] [no_empty] [no_header]

Arguments

- **input *pmdb***

  A required argument that specifies the input pattern library.

- **output *filename***

  A required argument that specifies the output filename.

- prop_name *name*

  An optional argument set that defines the properties to output. The argument set may be specified multiple times. By default all properties are output.

  If prop_name is specified, the properties are listed in the order given.

- no_empty

  An optional argument that causes only patterns that include properties to be listed. By default all patterns are listed in the output.

- no_header

  An optional argument that eliminates the header line from the output. By default, the output includes a header line that defines the output columns.

Description

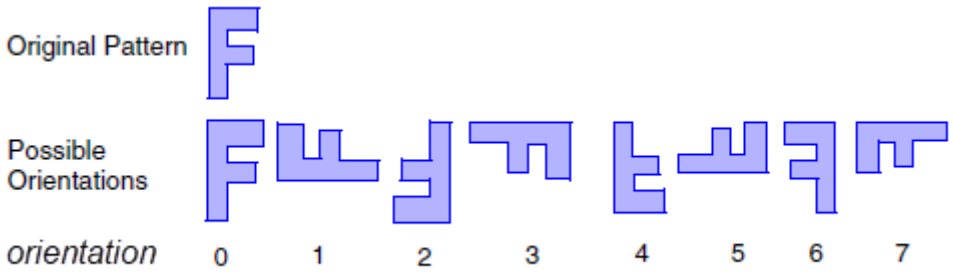
The output file has this format:

```
"pattern name",prop_name1[,prop_name2]...,orient_detail
pattern_name,value1[,value2]...,orient_value
...
```

The first line is a header line defining the columns. Specify no_header to exclude the header. If a property does not have an assigned value in a pattern, an empty string is output for the value.

The orient_detail property gives the allowed orientations of a match to the pattern. The reported value (*orient_value*) is a string containing the setting (0 or 1) for the eight possible pattern orientations. A pattern that can match in all orientations has an *orient_value* of 11111111.

Figure 5-9. orient_detail



Examples

The pattern library *liba.pmdb* has properties prop1 and prop2, and three patterns. prop1 is not defined in pat_3. prop2 is not defined in pat_2.

Example 1

```
pdl_lib_mgr export_properties input liba.pmdb output liba_props.csv
```

The output file *liba_props.csv* is this:

```
"pattern name",prop1,prop2,orient_detail
pat_1,1,1,11111111
pat_2,52,,11111111
pat_3,,23,11111111
```

Example 2

```
pdl_lib_mgr export_properties input liba.pmdb output liba_prop1.csv \
    prop_name prop1
```

The output file *liba_prop1.csv* only outputs the property prop1:

```
"pattern name",prop1
pat_1,1
pat_2,52
pat_3,
```

Example 3

```
pdl_lib_mgr export_properties input liba.pmdb output liba_prop1only.csv \
   prop_name prop1 no_empty
```

The output file *liba_prop1only.csv* only outputs the property prop1, and only lists patterns that have prop1 defined:

```
"pattern name",prop1
pat_1,1
pat_2,52
```

merge

Combines multiple pattern matching libraries (PMDBs) into a single PMDB library and removes duplicate patterns. If only one library is input, the command removes the duplicate patterns in that library.

Usage

pdl_lib_mgr [-turbo [*number_of_processors*]]
merge input *filename* [input *filename*]… **output *filename***
[precision *value*] [compile_output *filename* [maxjog *value*]]
[log_file *logfilename*] [ignore_layer_extents] [ignore_regions]
[gen_all_markers] [ignore_markers | merge_markers]
[conflict_prefix *prefix_string*]
[index *index_prop_name* [incr_max | first_available]]

Arguments

- -turbo [*number_of_processors*]

  An optional argument set that specifies to use multithreaded parallel processing. The argument set must be specified as shown in the syntax line, before the **merge** argument.

  The optional value *number_of_processors* is a positive integer that specifies the number of CPUs to use for multithreaded processing.If *number_of_processors* is not specified, Calibre runs on the maximum number of CPUs available for which you have licenses. For best performance, it is recommended that you avoid specifying this value.

- **input *filename*** [input *filename*] …

  A required argument set that specifies the input PMDB file. Multiple input libraries can be specified.

- **output *filename***

  A required argument that specifies the filename of the merged output PMDB.

- precision *value*

  An optional argument whose integer value is the number of points per database unit as defined within the pattern library. The default value if not specified is 1000. When merging libraries, it is recommended to use the lowest common multiple of precision values of all of the input libraries to ensure that information is not lost.

  If the precision setting causes pattern geometries to be truncated, a warning is issued and the pattern is not included in the output library.

- compile_output *filename*

  An optional argument that automatically runs the compilation step on the post-merge output PMDB file. This automatically creates the DMACRO file used by the pattern matching tool.

  For more information on the DMACRO file, see “DMACRO and CMACRO for Pattern Matching” on page 79.

- maxjog *value*

  An optional argument that is only used with compile_output. This option sets the jog tolerance feature, which instructs the tool to ignore jogs equal to or less than the specified *value*. The maximum value for maxjog is two times the global constraint (2*cglobal). The value is specified in microns.

  For information on global constraints, see "Global Constraints" on page 58.

  Note

  If a library being merged has a maxjog value, it is ignored (dropped) and a warning is issued. The maxjog value for the merged output library is determined by the maxjog argument.

  The maxjog argument should only be used when all patterns contain the same cglobal value. However, if patterns in the output library have differing cglobal values, the maxjog must be less than twice the minimum cglobal value for all patterns in the output library.

- log_file *logfilename*

  An optional argument that specifies to write out a log file with name *logfilename*. The file logs whether secondary patterns of geometrically identical patterns are removed or not and includes notes on the analysis.

- ignore_layer_extents

  An optional argument that specifies to ignore per layer extent differences while comparing patterns. The global pattern extent is always compared and is unaffected by this argument. Patterns that only differ in per layer extents are considered duplicate and only the first such pattern is saved to the output library.

- ignore_regions

  An optional argument that specifies to ignore region differences while comparing patterns. Patterns that differ only in their regions are considered duplicate and only the first such pattern is saved to the output library.

- gen_all_markers

  An optional argument that specifies to remove duplicate patterns that differ only in their markers. The output pattern includes a superset of all unique markers.

  Unique markers are defined as having both different names and different types or geometries. Identical markers have both the same name and same type or geometry. Markers that are not unique or identical have the same name but different types or geometries. The secondary pattern is only removed if all markers are unique or identical for the duplicate patterns. If this condition is not met, the secondary pattern is not removed—the differing patterns are saved as separate patterns to the output library.

  The gen_all_markers argument may be specified with ignore_markers or merge_markers. See "Marker Considerations" on page 248 and the "Example 2" on page 249.

- ignore_markers | merge_markers

  An optional argument choice that specifies behavior for marker handling.

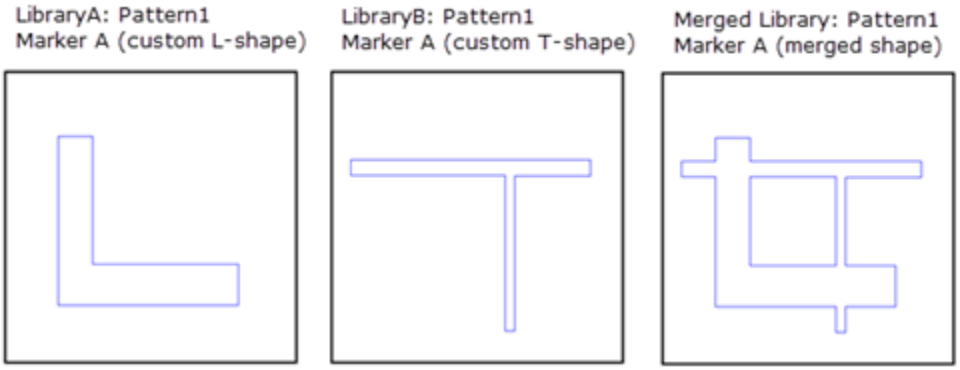
  ignore_markers — Specifies to ignore markers while comparing patterns. Patterns that differ only in their markers are considered duplicate and only the first such pattern is saved to the output library.

  merge_markers — Specifies to remove duplicate patterns that differ only in the custom marker geometries and to merge the geometries for the custom markers.

  To qualify, the patterns must have identical markers for the non-custom markers and the custom markers must have identical names. The custom marker geometries are merged with an OR function, combining the geometries. Only custom type markers are merged; computed type markers (bbox, drawn, matched, etc.) do not merge.

  Figure 5-10 shows how two different custom markers are merged. Pattern1 in LibraryA and LibraryB differ only in the custom marker A.

Figure 5-10. Result from merge_markers



  See “Marker Considerations” on page 248 and the “Example 2” on page 249.

- conflict_prefix *prefix_string*

  An optional argument set that specifies to prefix the pattern name with *“prefix_string_”* in order to resolve naming conflicts when unique patterns in different libraries have the same name. The default behavior is to prefix the pattern name with the library name. See “Resolving Pattern Name Conflicts” on page 245 for complete details.

- index *index_prop_name* [incr_max | first_available]

  An optional argument set that specifies how to handle the property *index_prop_name* in the merged output library.

  If the property *index_prop_name* does not exist in any input library, it is added to the merged library with sequential values starting at 1 for the first pattern. The property type is double.

If *index_prop_name* exists in any input library, the following rules control the property in the merged library:

- If the property type is not the same in all input libraries, the property is saved as type double in the merged library. (This differs from the behavior without the index argument, in which property types must match.)
- If duplicate patterns have conflicting values for *index_prop_name*, the value in the first found duplicate pattern is used. (This differs from the behavior without the index argument, in which the minimum property value is used.)
- If there is no assigned property value for a unique pattern or a set of duplicate patterns, the value in the merged library is assigned as follows, depending on the choice of incr_max or first_available:
  - incr_max — Assign sequential values starting with the largest value of *index_prop_name* and incrementing by 1. If the largest value is a floating point number, it is truncated before incrementing. This is the default when "index *index_prop_name*" is specified.
  - first_available — Assign the first integer index value that is not assigned to another pattern.

See "Property Handling During Merging" on page 247 for a discussion of the default behavior and examples of both the default behavior and the behavior with the index argument.

Description

This utility merges one or more input PMDB libraries into a single PMDB library by removing duplicate patterns. By default, patterns are considered duplicate if the pattern shapes, layer extents, regions, Don't Care pattern layers, and markers are identical. If all pattern orientations are allowed, patterns that differ only in orientation are considered duplicate and are removed. Some options cause certain pattern differences to be ignored, and this changes the definition of a duplicate pattern.

If patterns have Don't Care pattern layers, the patterns must have Don't Care layers at the same layer index in order for the patterns to be considered duplicates. The geometries on the Don't Care layers are not compared. If pattern shapes, layer extents, regions, Don't Care layers, and markers are identical between patterns, the first pattern is saved to the merged library.

Only exact patterns are evaluated for duplicate removal, meaning any pattern with constraints (with the exception of cglobal) is passed directly to the output library.

Pattern precedence is determined by the order of patterns in the library and the order of input libraries on the command line. When duplicate patterns are found, the first pattern instance is considered the primary pattern and takes precedence over all duplicate patterns, termed secondary patterns.

Conflicts occur when otherwise identical patterns have different values of cglobal, attributes, properties, orientation, or comments. The following table describes how the conflict is resolved.

Pattern Feature	Conflict Resolution
cglobal value	Use the larger cglobal value.
Custom attribute	Text attribute: Keep the value from the pattern that is encountered first and report the conflict in the transcript. Numeric attribute: Keep the lowest value of the numeric attribute when there is a conflict. If two attributes have the same name but different types, the type of the attribute that is encountered first is used and a message is written to the transcript.
Comments	The comments from all duplicate patterns are concatenated and saved to the pattern in the merged library.
Keys	The keys from all duplicate patterns are saved to the pattern in the merged library.
Properties	See “Property Handling During Merging” on page 247.
Orientation	See “Pattern Orientation Handling During Merging” on page 245.

Caution

When merging libraries, the tool only uses the layer definitions from the first input library. It is highly important that layer names, order, and number of layers are identical for all the input libraries. An error is reported if the number of layers differ between input libraries and the merge does not take place.

When layer definitions differ, patterns defined in subsequent libraries use the layer definitions from the first input library of the merge operation, resulting in patterns that do not apply to the intended layer and causing incorrect pattern matches. For additional information, see “About Merging Pattern Libraries” on page 23.

The merge utility can also be run from the Calibre Pattern Matching GUI; see “Merging Pattern Libraries Using the GUI” on page 99.

See these sections for detailed behavior:

- Resolving Pattern Name Conflicts
- Pattern Orientation Handling During Merging
- Property Handling During Merging
- Marker Considerations

Resolving Pattern Name Conflicts

By default, if unique (differing) patterns with the same name are found in different input libraries, the pattern name from the second library is prefixed with "*<lib2>*_" when the pattern is added to the merged output library, where *<lib2>* is replaced with the name of the second library. For example, if libA and libB are merged, with libA listed first on the command line, and both libraries have a unique pattern named mypat, then the merged output library contains the patterns mypat (from libA) and libB_mypat (from libB). If more than two libraries are merged, the corresponding library name is used to resolve pattern naming conflicts. If a library name has period (.) or other special character that is not allowed in pattern names, the special character is replaced with an underscore (_).

You can specify a user-defined prefix string with the "conflict_prefix *prefix_string*" argument and value. Pattern renaming is done as described in the previous paragraph using "*prefix_string_*" instead of "*<lib2>*_". For example with a *prefix_string* of "rename", if libA and libB are merged and both libraries have a unique pattern named mypat, then the merged output library contains the patterns mypat (from libA) and rename_mypat (from libB). If a third or subsequent library also has a unique pattern with the same name, "_*<ordinal>*" is appended to the pattern name, where *<ordinal>* starts at 1 and is incremented for each library. For example, if four libraries are merged and all have a unique pattern named mypat, the patterns in the merged library are named mypat, rename_mypat, rename_mypat_1, and rename_mypat_2, corresponding to the four input libraries.

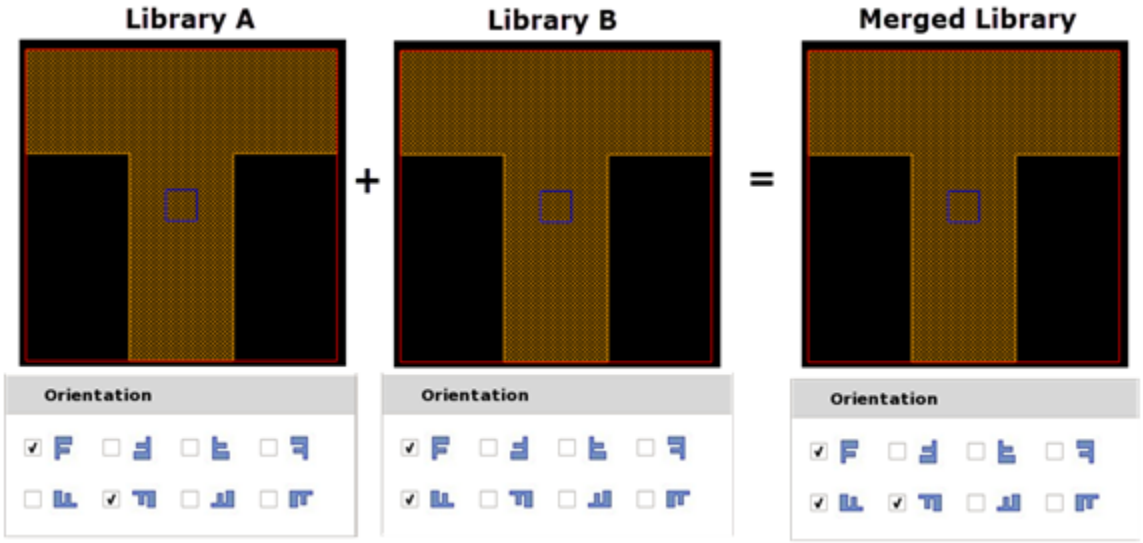
Pattern Orientation Handling During Merging

When all orientations of a pattern are allowed, patterns that differ only in drawn orientation are considered duplicates. Pattern orientation is specified on the **Attributes** tab in the GUI, and with the orient_detail reserved property name (see "Reserved Property Names" on page 71).

If specific pattern orientations are allowed, the merged pattern has a pattern orientation setting such that pattern matching results in the same matches as the original patterns. Specifically, the merge process can be summarized with the following two cases:

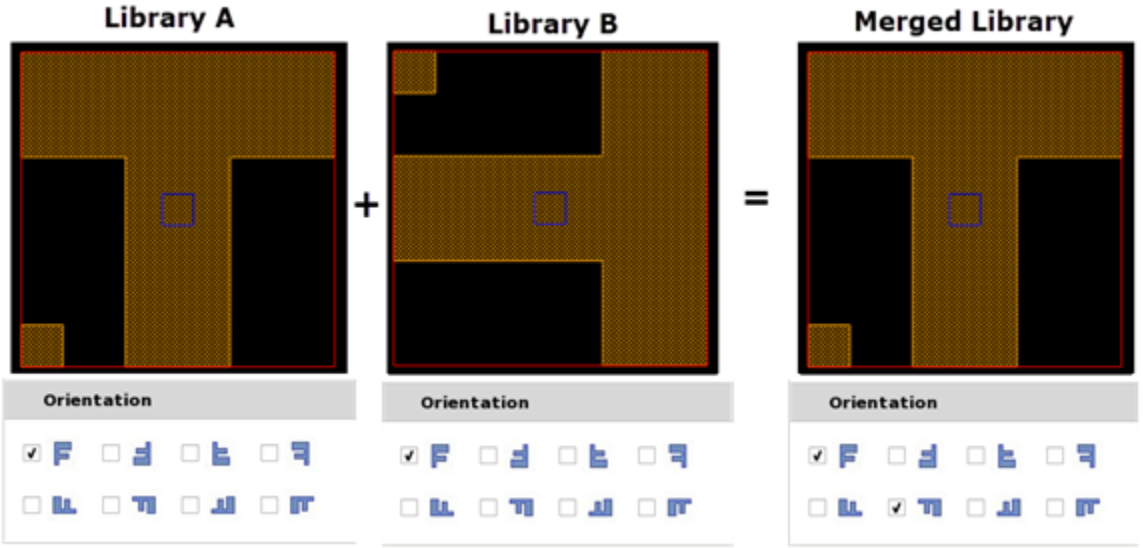
- **Case A**: Two patterns have the same geometries drawn in the same orientation, but differing values of pattern orientation.

Result: The merged pattern has the combined (OR'd) pattern orientations. For example:



- **Case A**: Two patterns have the same geometries, but they are drawn in differing orientations.

  Result: The primary pattern geometry is saved to the merged library. The secondary pattern is rotated and/or flipped to match the primary pattern and the pattern orientation setting is adjusted so that the same matches result from the rotated/flipped pattern. The pattern orientation of the primary pattern and the rotated/flipped secondary pattern are combined and saved to the merged library.



Property Handling During Merging

The merged library contains the complete set of all properties defined in the input libraries. By default, property values are assigned as follows:

- If duplicate patterns have conflicting values of a property, the smaller property value is used.
- If the source of a pattern in the merged library did not have an assigned value for a particular property, no property value is assigned for that pattern in the merged library.

By default, an error occurs if two input libraries have a property with the same name but different property types. You can use the convert utility to change the property type.

The default behavior for a specified property name can be changed with the index argument set. This argument set is useful to assign unique index values to all patterns in the merged library as a way of tracking the source of a pattern match. See "Identifying the Source Pattern" on page 18.

For example, suppose two libraries *lib_a.pmdb* and *lib_b.pmdb* have the patterns in the following table, with the listed values for property pID:

lib_a	lib_b	Comment
pat_1: pID=1	pat_1: pID: no value	Duplicate patterns
pat_2: pID: no value	pat_2: pID: no value	Duplicate patterns
pat_3: pID=33	pat_3: pID=23	Duplicate patterns
uniq_1: pID=13	pat_4: pID: no value	Unique patterns

The merged library has the following patterns, where library lib_a the first input library in the merge operation. The pID property value depends on the arguments used (columns 2, 3, and 4).

Patterns	Default merge	index pID incr_max	index pID first_available
pat_1	pID=1	pID=1	pID=1
pat_2	pID: no value	pID=34	pID=2
pat_3	pID=23	pID=33	pID=33
uniq_1	pID=13	pID=13	pID=13
pat_4	pID: no value	pID=35	pID=3

Comments:

- Patterns pat_1 and uniq_1 have assigned pID values with no conflicts, so the value in the merged library is the same in all cases.

- Patterns pat_2 and pat_4 have no assigned pID value in any input library, so there is no assigned value in the merged library with the default behavior. When “index pID incr_max” is specified, values are assigned, incremented from the largest value of pID (33). When “index pID first_available” is specified, values are assigned using the first available integer values (2 and 3).
- Pattern pat_3 has a pID value of 33 in lib_a, and a value of 23 in lib_b. With the default behavior, the merged pattern has a value of 23, the lowest value. When index is specified, the merged pattern has a value of 33, the value from the pattern in the first library (lib_a).

Marker Considerations

Both ignore_markers and gen_all_markers can be used together in order to exclude the secondary pattern while removing only the conflicting (non-unique or non-identical) secondary markers from the generated superset of markers for the primary pattern. This is different from ignore_markers, where all markers from a secondary pattern are removed. For an example of how gen_all_markers and ignore_markers can affect the output of the pdl_lib_mgr merge command, see “Example 2” on page 249.

The following table describes the behavior of different options when input patterns differ only in their markers. The second column in which “markers are unique or identical” assumes at least one unique marker between the pattern marker sets so that the marker sets are different. Completely identical patterns (including identical marker sets) result in the removal of the secondary pattern by default. Properties in the secondary pattern are retained in the merged pattern. The option merge_markers is not present in this table as it depends on having identical names for custom type markers.

Table 5-2. Behavior of pdl_lib_mgr merge With Marker-Related Options

Options	Markers unique or identical (at least one unique marker)	Markers not unique or identical
default	Saves separate patterns	Saves separate patterns
ignore_markers	Removes secondary pattern	Removes secondary pattern
gen_all_markers	Generates a superset of unique markers, and the secondary pattern is removed	Saves separate patterns
gen_all_markers ignore_markers	Generates a superset of unique markers, and the secondary pattern is removed	Generates a superset of unique markers, non-unique markers from secondary patterns are not saved

When gen_all markers and merge_markers are specified together, the behavior is as with gen_all_markers, but customer markers with the same name are merged. See “Example 2” on page 249.

Examples

Example 1

The following example merges *pmfile1.pmdb* and *pmfile2.pmdb* files and outputs the merged PMDB file. This command line invocation also automatically runs the compilation step that creates the DMACRO file that is used in your main SVRF rule file for the pattern matching run.

```
$ pdl_lib_mgr merge input pmfile1.pmdb input pmfile2.pmdb output new.pmdb
compile_output new_dmacro.svrf
```

Example 2

This example presents several cases where two otherwise identical patterns have different markers, and how using ignore_markers and gen_all_markers impacts the output. In these cases, Pattern 1 is the primary pattern, and Pattern 2 is the secondary pattern.

Case 1

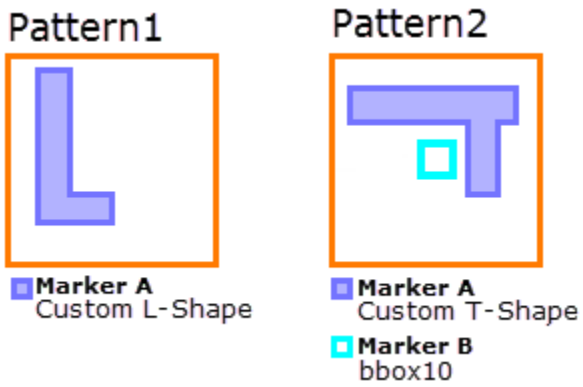
Pattern Inputs
Pattern 1
Marker A (bbox)
Marker C (custom L-shape)
Pattern 2
Marker B (bbox10)
Marker C (custom L-shape)

In this case Pattern 1 and Pattern 2 have unique or identical markers: Marker A and Marker B are unique and Marker C for both patterns is identical. This means that gen_all_markers can remove the secondary pattern and generate a superset of the unique markers for the primary pattern.

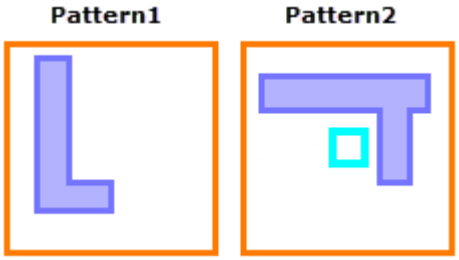
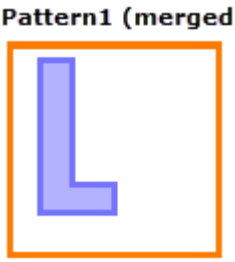
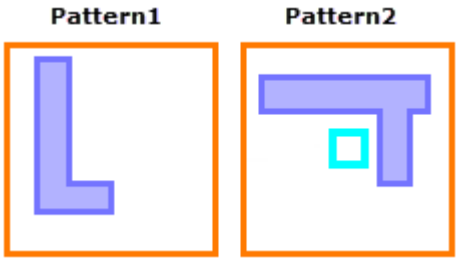
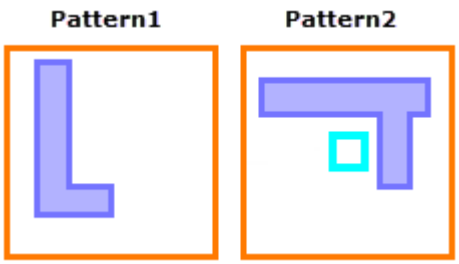
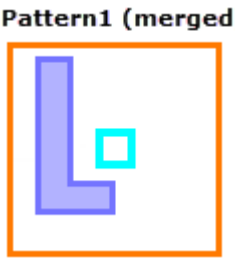
Arguments	Output Merged Library	Comment
ignore_markers	Pattern 1 Marker A (bbox) Marker C (custom L-shape)	The patterns are considered duplicates because the secondary pattern's markers are ignored. The secondary (duplicate) pattern is removed with its markers.
gen_all_markers	Pattern 1 Marker A (bbox) Marker B (bbox10) Marker C (Custom L-shape)	The markers are unique or identical, so there are no conflicting markers. A superset of unique markers is generated. The secondary pattern is removed.

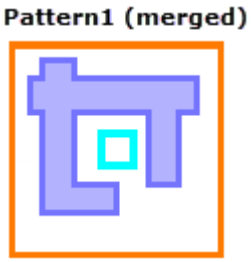
Arguments	Output Merged Library	Comment
merge_markers	Same as pattern input: Pattern 1 Marker A (bbox) Marker C (custom L-shape) Pattern 2 Marker B (bbox10) Marker C (custom L-shape)	While Marker C for both patterns consist of a custom shape marker, Pattern 1 includes Marker A and Pattern 2 includes Marker B. Since the names of the markers are not identical, both patterns are saved in the output.
gen_all_markers ignore_markers	Pattern 1 Marker A (bbox) Marker B (bbox10) Marker C (Custom L-shape)	The markers are unique or identical, so there are no conflicting secondary markers to ignore. A superset of unique markers is generated.The secondary pattern is removed.
gen_all_markers merge_markers	Pattern 1 Marker A (bbox) Marker B (bbox10) Marker C (Custom L-shape)	A superset of markers is generated, and precedence is given for computed markers. Custom type markers with the same name are merged.

Case 2



In this case Pattern1 and Pattern2 do not have unique or identical markers: Marker A is not a unique name as it is used in both patterns, and is also not identical due to the difference in custom shape. This means gen_all_markers does not remove the secondary pattern. However, when gen_all_markers and ignore_markers are specified together, the secondary pattern is removed and the primary pattern is output with the generated set of unique markers—in this case Marker A from the secondary pattern is ignored.

Arguments	Output	Comment
default		The patterns are not considered duplicate because they have different markers.
ignore_markers		Successful pattern merge. The patterns are considered duplicates because the markers for Pattern2 (the secondary pattern) are ignored. The secondary (duplicate) pattern is removed with its markers.
gen_all_markers		Marker A is used in both patterns and is not identical, so a superset of unique markers can not be generated. Both patterns are saved in the output.
merge_markers		While Marker A for both patterns consists of a custom shape marker, Pattern 2 includes marker B. Since Pattern1 does not contain marker B, both patterns are saved in the output.
gen_all_markers ignore_markers		Successful pattern merge. Marker A for the secondary pattern is conflicting, so it is not included in the superset of unique markers generated for the pattern. The secondary pattern is removed.

Arguments	Output	Comment
gen_all_markers merge_markers		Successful pattern merge. A superset of markers is generated, and precedence is given for computed markers. Custom type markers with the same name are merged.

prompt

Opens the Tcl command-line interface to the Pattern Manager Library Utility, or executes a specified Tcl script.

Note

The pattern matching API interface is discussed in the *Calibre Pattern Matching Reference Manual*.

Usage

pdl_lib_mgr prompt [*tcl_script*]

Arguments

- *tcl_script*

  An optional argument that specifies a Tcl scripting file.

Description

Provides access to the pattern matching Tcl-based API batch interface, which enables you to edit and traverse pattern libraries.

Examples

This example invokes the Tcl-based API console, processes some patterns, and exits.

```
$ pdl_lib_mgr prompt
//  Calibre PDL Lib Manager v2014.1 ...
%

# Load a pattern library and save a list of library pattern names
% set x_lib [pmatch::load_pattern_lib -path example_lib.pmdb]
Pattern Library DATABASE
%
% set patterns [pmatch::get_pattern_names -lib $x_lib]
p1_exact p4_bcm p5_bcm
%

# Output the number of patterns in the pattern library
% set num_patterns [llength $patterns]
3
% puts "There are $num_patterns patterns in this library"
There are 3 patterns in this library
%
% exit
$
```

unfold

Reads the patterns in a pattern matching library (PMDB) and outputs a series of patterns without constraints. The output patterns are the result of applying the pattern constraints in various permutations. Options control the number of output patterns and the edge movement between permutations.

Usage

pdl_lib_mgr [-turbo [*number_of_processors*]]
unfold input *filename* output *filename*
[pattern_name *pat_name*]
[pattern_limit *limit_num*]
[step_size *step_dist* | linear_step_size *step_dist*]
[index_prop *prop_name*]

Arguments

- [-turbo [*number_of_processors*]]

  An optional argument set that specifies to use multithreaded parallel processing. The argument set must be specified as shown in the syntax line, before the **unfold** argument.

  The optional value *number_of_processors* is a positive integer that specifies the number of CPUs to use for multithreaded processing. If *number_of_processors* is not specified, Calibre runs on the maximum number of CPUs available for which you have licenses. For best performance, it is recommended that you avoid specifying this value.

- **input *filename***

  A required argument that specifies the input pattern library.

- **output *filename***

  A required argument that specifies the name of the output (unfolded) pattern library.

  See “Pattern Naming in the Output Library” on page 256 for the pattern naming convention in the output library.

- pattern_name *pat_name*

  An optional argument that specifies the name of a pattern. If this option is specified, only the pattern *pat_name* is unfolded and saved to the output database.

- pattern_limit *limit_num*

  An optional argument that specifies the maximum number of permutations that are created *for each* pattern. If pattern_limit is not set, the default maximum output is 100 unfolded patterns.

  The first pattern to be output is always an exact match of the unconstrained pattern polygons. By default (linear_step_size or step_size not specified), the unfolding then proceeds as with the step_size algorithm and a default *step_dist* of 1 dbu. If linear_step_size or step_size is specified, the unfolding proceeds with the specified process and using the

provided *step_dist* value. The unfolding process continues until all possible permutations are output, or the pattern limit is met.

The pattern_limit option affects the numeric portion of the name of each output pattern. See "Pattern Naming in the Output Library" on page 256.

- step_size *step_dist* | linear_step_size *step_dist*

  An optional argument choice that specifies to unfold patterns based on a *step_dist* in microns. Both choices reduce the number of patterns that are output. The step_size argument results in the best pattern coverage if the pattern limit is met before unfolding is complete. The linear_step_size argument outputs patterns in a more intuitive order. If neither option is specified, the default is "step_size 1".

  step_size *step_dist*

  The patterns are unfolded such that the original pattern, maximum inside, and maximum outside movements are output first. Then the edges near the midpoint of the allowed edge movement are output. Next, more edges are output at the midpoints between previously output edges. Edge movement is always in multiples of *step_dist*, except for the edges output at the maximum inside and maximum outside movements. This ordering of output patterns ensures that edges at the maximum and minimum range are output first, so that more of the constraint range is covered before reaching the pattern_limit, if one is specified.

  linear_step_size *step_dist*

  The patterns are unfolded in a linear, increasing order. The moving edge of the pattern moves by the specified *step_dist* until the distance to the maximum outside or inside edge is less than the specified value. The moving edge then moves the remaining distance to the maximum constraint value. If a pattern_limit is specified, unfolded patterns at the maximum range may not be output, unlike the case with the step_size option.

  For example, assume a pattern to be unfolded has a single moving edge with a maximum inside movement of 0.04 um and a maximum outside movement of 0.06 um. A *step_dist* of 0.01 um results in eleven output patterns with the moving edge at these locations, depending on the option used:

  step_size 0.01: 0, 0.06, -0.40, 0.01, 0.04, -0.10, 0.05, 0.03, -0.02, 0.02, -0.3

  linear_step_size 0.01: 0, -0.01, -0.02, -0.03, -0.04, 0.01, 0.02, 0.03, 0.04, 0.05, 0.6

- index_prop *prop_name*

  An optional argument that adds a numeric property named *prop_name* with a unique value to each pattern. The property value starts at one with the first pattern that is output and is incremented by one for each subsequent output pattern. If a library contains a property named *prop_name*, the value is overwritten.

Description

The unfold utility reads a pattern matching library (PMDB) with constrained patterns and outputs a series of patterns without constraints (the unfolded patterns).

Patterns without constraints are copied to the output library. Invalid patterns are not copied. The default behavior uses “pattern_limit 100” and “step_size 1”.

The unfold utility can also be run from the Calibre Pattern Matching GUI; see “Unfolding Patterns in a Library Using the GUI” on page 112.

Pattern Naming in the Output Library

Unfolded patterns in the output library are named with this convention:
<unfold_count>_<source_pattern_name>

- *unfold_count* — An integer that starts at one for the first unfolded pattern of each source pattern and is incremented by 1 for each unfolded pattern. That is, for patterns patA and patB, the first unfolded patterns for each pattern are named 1_patA and 1_patB, respectively. The second unfolded patterns are named 2_patA and 2_patB.

  If “pattern_limit *limit_num*” is specified, the minimum number of digits in *unfold_count* is the number of digits in *limit_num*. For example, if limit_num is 500, the first unfolded pattern for patA is named 001_patA. You can force more digits to be used by adding leading zeros to *limit_num*. This feature is useful when the unfolded pattern library is input to the pdl_lib_mgr write_oasis utility, as explained in the following section.

- *source_pattern_name* — The name of the pattern being unfolded.

Patterns without constraints have the same name as the source pattern.

Controlling Output Order in write_oasis By Using pattern_limit

The write_oasis utility outputs patterns alphabetically, with the alphabetically first pattern in the lower left corner of the OASIS layout. Without the use of the pattern_limit option, the alphabetical sort may not output patterns in the order they were unfolded.

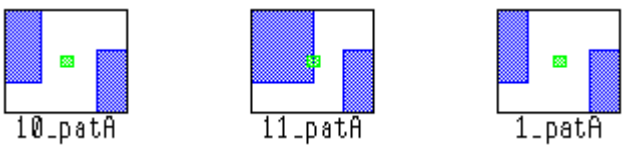
Suppose a library has pattern patA, and the following unfold command is used:

```
pdl_lib_mgr unfold input lib1.pmdb output lib1_u.pmdb step_size 0.2
```

This results in 11 unfolded patterns, which are sorted alphabetically as follows.

10_patA, 11_patA, 1_patA, 2_patA, … 9_patA

Running write_oasis on the unfolded library results in this output for the first row of output patterns:



The following command uses a pattern limit larger than the maximum number of unfolded patterns:

```
pdl_lib_mgr unfold input lib1.pmdb output lib1_u2.pmdb step_size 0.2
            pattern_limit 20
```

The resulting unfolded patterns are sorted alphabetically as follows.

01_patA, 02_patA, 03_patA, … 10_patA, 11_patA

Running write_oasis on the unfolded library results in this output for the first row of output patterns:

Note

The write_oasis utility can also sort output by a property value with the sort_by argument.

Examples

For example, the library *liba.pmdb* has a pattern named pat_2 with one edge that can move out 0.1 microns. The following command unfolds only pat_2 and limits the number of unfolded patterns to three:

```
pdl_lib_mgr unfold input liba.pmdb output liba_unfold_pat2.pmdb \
    pattern_name pat_2 pattern_limit 3
```

These permutations of pat_2 are saved to the output library *liba_unfold_pat2.pmdb*:

1. 1_pat_2: The drawn pattern with no constraints.
2. 2_pat_2: The pattern with the constrained edge at the maximum movement (0.1 microns).
3. 3_pat_2: The pattern with the constrained edge at the midpoint of its allowed movement (0.05 microns).

write_oasis

Generates an OASIS file including all polygon information from pattern layers. The utility can also generate PNG-formatted image files corresponding to each pattern in the library.

Usage

pdl_lib_mgr [-turbo [*number_of_processors*]]
write_oasis input *filename*
[output *filename* | oasis_per_pattern]
[precision *num*]
[space *value*] [columns *num*]
[text_height *size*] [props_as_text] [pattern_name *name*] [name_inside]
[cluster [prop *property_name*] [cluster_min *num*]]
[extent *OASIS_layer_name OASIS_layer_num*]
[{marker *marker_name OASIS_layer_name OASIS_layer_num* [attach_props]}...]
[{layer *index* [poly_layer *OASIS_layer*] [extent_layer *OASIS_layer*]
[constraint *OASIS_layer_name OASIS_layer_num*]
[region {critical | keep_out | dont_care} *OASIS_layer_name OASIS_layer_num*]…}...]
[sort_by *prop_or_attr_name1* {descending | ascending}
[secondary_sorter *prop_or_attr_name2* {descending | ascending}]]
[snapshot [out_dir *directory*] [layer_props *layer_props_file*] [all]
[append_layers_palette]]

Arguments

- -turbo [*number_of_processors*]

  An optional argument set that specifies to use multithreaded parallel processing. The argument set must be specified as shown in the syntax line, before the write_oasis argument.The optional value *number_of_processors* is a positive integer that specifies the number of CPUs to use for multithreaded processing. If *number_of_processors* is not specified, Calibre runs on the maximum number of CPUs available for which you have licenses. For best performance, it is recommended that you avoid specifying this value.

- **input *filename***

  A required argument that specifies the pattern matching database (PMDB). The file extension (*.pmdb*) is optional.

- output *filename* | oasis_per_pattern

  An optional argument choice that specifies the OASIS output. One of the argument choices is required unless snapshot is specified.

  output *filename* — Specifies to write all patterns to a single OASIS file named *filename*.

  oasis_per_pattern — Specifies to output one OASIS file per pattern. The files are named *pattern_name.oas* and are created in the working directory. If the precision argument is not specified, each output file uses the pattern precision by default.

The output and oasis_per_pattern arguments cannot be specified together. The snapshot argument may be specified with either argument.

- precision *num*

  An optional argument that specifies the output file precision.

  The default is the lowest common multiple of precision values in your library; this value prevents truncation of any geometries. If oasis_per_pattern is specified, each output file uses the pattern precision by default.

  A warning is issued if the precision setting causes any pattern geometry to be truncated or if a pattern becomes invalid.

  Caution

  Use the lowest common multiple of precision values in your library to ensure that information is not lost.

- space *value*

  An optional argument set that specifies the default space between patterns in microns. The default is the maximum width or height across all patterns. Set “space 0” for the minimum spacing.

- text_height *size*

  An optional argument set that specifies the text height in microns. It is used to determine the placement coordinate (x, y) of text below the pattern. The default text height is one quarter of the pattern to pattern spacing.

  Note

  You must set the text height in your layout viewer to the same value.

- props_as_text

  An optional argument that specifies to write property names and values as text below the pattern name. This option is ignored if oasis_per_pattern is specified. Property information is not included for properties that are defined in the library but do not have an assigned value for a particular pattern.

- pattern_name *name*

  An optional argument set that specifies to output only the pattern with the given pattern name. The pattern is output at the position the pattern was originally defined rather than starting at the 0,0 origin.

  This argument also affects the output from the snapshot and oasis_per_pattern arguments. The cluster option is ignored if it is specified with pattern_name, and a warning is issued.

- name_inside

  An optional argument that controls the placement of the pattern name. The pattern name is placed in the center of the pattern extent if this option is specified.

If the pattern has a custom extent with a non-rectangular shape, it is virtually divided into trapezoidal sections and the name is written in the center of the first section.

By default, the name text is placed below the pattern for readability. Use the name_inside option for tools that read OASIS files and need to associate the pattern name with the pattern extent. The output is less readable since the name text is placed on top of any geometries that are in the middle of the pattern extent. The text_height argument is ignored if name_inside is specified.

- columns *num*

  An optional argument set that specifies the number of columns to be used for the output. By default, patterns are output with roughly the same number of rows and columns.

- cluster [prop *property_name*] [cluster_min *num*]

  An optional argument set that outputs *predefined* clusters of similar patterns. By default, the write_oasis utility looks for clusters defined with a property name of “cluster”. You can specify a different property name using the “prop” option. The cluster_min option specifies the minimum number of patterns in a cluster. The default is 1.

  The cluster option is ignored if it is specified with the pattern_name option, and a warning is issued.

  Each cluster output is placed in its own OASIS cell, which is referenced from the top cell. The cell name is selected from one of the patterns in the cluster. The patterns in the cluster output are sorted from largest to smallest. Patterns which are not part of a cluster are placed in the top cell.

  Tip

  Before executing write_oasis with the cluster option, run the “pdl_lib_mgr cluster” utility to define clusters of similar patterns.

- extent *OASIS_layer_name OASIS_layer_num*

  An optional argument set that defines an output layer for global extent information. This layer is used for pattern names if the name_inside argument is specified. The OASIS output layer number must not be duplicated by any other OASIS layer number.

  If the layer argument set is specified, the extent argument set must also be specified in order to output the global extent.

- {marker *marker_name OASIS_layer_name OASIS_layer_num* [attach_props]}...

  An optional argument set that defines the output layer for the marker *marker_name*. This argument set can be repeated for multiple markers. The OASIS output layer number must not be duplicated by any other OASIS layer number.

  For per layer bbox and bbox10 markers, the marker geometry is determined from the layer extent if one is defined, otherwise the geometry is determined from the pattern extent.

  - attach_props — An optional argument that specifies to attach property information as an OASIS property to the layer *OASIS_layer_name*. OASIS properties in a layout

database can be read with the DFM Read operation. Property information is not included for properties that are defined in the library but do not have an assigned value for a particular pattern. See "Example 4" on page 264.

- {layer *index* [poly_layer *OASIS_layer_num*] [extent_layer *OASIS_layer_num*] [constraint *OASIS_layer_name OASIS_layer_num*] [region {critical | keep_out | dont_care} *OASIS_layer_name OASIS_layer_num*]…}...

  An optional argument set that defines output layers for pattern geometries, layer extents, constraints, and regions. The assigned OASIS layer number must not be duplicated by any other OASIS layer number. Layer extents, constraints, and regions are not output if the layer argument set is not specified.

  - layer *index* — Specifies the pattern layer the other options apply to, where *index* is the ordinal of the pattern layer, starting at 1. For example, if a library has three pattern layers, *index* is 3 for the last listed pattern layer. The list of pattern layers can be viewed in the Calibre Pattern Matching GUI by choosing **File > Library Attributes** and selecting the **Layers** category.
  - poly_layer *OASIS_layer_num* — Specifies to output pattern geometries for layer *index* to *OASIS_layer_num*, where *OASIS_layer_num* is an integer.
  - extent_layer *OASIS_layer_num* — Specifies to output layer extents for layer *index* to *OASIS_layer_num*.
  - constraint *OASIS_layer_name OASIS_layer_num* — Specifies to output constraint information for layer *index* to the specified OASIS layer. The allowed movement area of each edge due to a cglobal value or edge constraints is output.
  - region {critical | keep_out | dont_care} *OASIS_layer_name OASIS_layer_num*

    Specifies to output the indicated region (critical, keep_out, or dont_care) for layer *index* to the specified OASIS layer. The region argument set can be specified once for each region type that exists for layer *index*. See "Region Layers" on page 50 for information on regions.

  The layer argument set can be repeated once for each pattern layer. For example, for a library with two pattern layers, a layer extent for pattern layer 1, and critical and keep_out regions for pattern layer 2, the following layer argument sets can be specified:

  ```
  layer 1 poly_layer 101 extent_layer 201 \
  layer 2 poly_layer 102 region critical crit_2 302 \
                         region keep_out ko_2 402
  ```

  If the layer argument set is specified, the global extent is not output unless the extent argument set is also specified.

- sort_by *prop_or_attr_name1* {descending | ascending} [secondary_sorter *prop_or_attr_name2* {descending | ascending}]

  An optional argument set that specifies a sorting method for the placement of patterns in the OASIS layout. A primary sorting method and optional secondary sorting method are used.

Sorting is done based on the values of custom properties or attributes. The property or attribute must exist in the pattern library. Types long and double are supported for both properties and attributes. Attributes also support text values.

- sort_by *prop_or_attr_name1* — Specifies the primary sorting method, where *prop_or_attr_name1* is the name of a custom property or attribute to sort by.
- descending | ascending — Specifies the sorting method. For descending, the pattern with the largest property or attribute value is placed in the lower left corner of the OASIS layout. For ascending, the pattern with the smallest property or attribute value is placed in the lower left corner of the OASIS layout.
- secondary_sorter *prop_or_attr_name2* — Specifies the secondary sorting property or attribute name. The secondary sorting method is used when two patterns have the same value for the primary sorting property or attribute.

  If *prop_or_attr_name2* is the same as *prop_or_attr_name1*, the secondary_sorter arguments are ignored.

If two patterns have the same value for the primary sorting property or attribute, or there is no value set, the secondary sorting method is applied, if specified. If secondary_sorter is not specified, the patterns are sorted by the pattern name.

If the name specified by *prop_or_attr_name1* or *prop_or_attr_name2* exists as both a property and an attribute in the library, the property is used for sorting.

See “Example 5” on page 264.

These arguments are not supported with sort_by: pattern_name, cluster, oasis_per_pattern, and snapshot

- snapshot [out_dir *directory*] [layer_props *layer_props_file*] [all] [append_layers_palette]

  An optional argument set that specifies to generate one image file per pattern. The files are in Portable Network Graphics (PNG) format and named <*pattern_name*>.*png*. By default only pattern layers and the pattern extent are included in the image.

  - out_dir *directory* — Specifies to create the PNG files in a separate directory named *directory*, which must exist. If out_dir is not specified the PNG files are created in the working directory.
  - layer_props *layer_props_file* — Specifies to use a Calibre DESIGNrev layerprops file for the layer properties (color, line width, fill pattern, and layer name). If layer_props is not specified, the default layer properties used by the Calibre Pattern Matching GUI are used.
  - all — Specifies to include marker and region layers in the images, in addition to the pattern layers and pattern extent.
  - append_layers_palette — Specifies to include a layer palette with the snapshot. The layer palette has an image of the layer color and fill pattern, and the layer name.

The output or oasis_per_pattern argument may be specified with snapshot, but is not required. The extent, marker, and layer argument sets have no effect on the snapshot output.

Description

The write_oasis utility generates a graphical OASIS file corresponding to the patterns in the input library. The output mode is determined by the output or oasis_per_pattern arguments. The output argument causes patterns to be written to a single OASIS file. The oasis_per_pattern argument outputs one OASIS file per pattern.

The utility can also generate PNG-formatted image files corresponding to each pattern in the library. The PNG output can be generated instead of the OASIS output or in addition to it.

OASIS Generation

By default, polygon information from all pattern layers and the global extent layer is output to the OASIS file. The utility examines the input library to find the maximum width or height across patterns, and then roughly equal rows and columns of patterns are output. You can specify the number of columns with the columns argument.

Pattern names are written out below the pattern on the extent layer by default; the placement can be changed with the name_inside argument. Pattern names are not included in the output generated by the oasis_per_pattern argument.

If the pattern library assigns layer numbers to pattern layers, the layer numbers are used for the corresponding output OASIS layers, unless the layer argument set specifies the OASIS layer numbers to use for each pattern layer.

You can specify to output markers, layer extents, regions, and constraints with the marker and layer argument sets.

You can limit the output to a specific pattern with the pattern_name argument.

PNG File Generation

The snapshot argument set causes a PNG image to be created for each pattern in the library. See the argument description for details.

Examples

Example 1

Run the cluster utility to create a library with the cluster property. Write out an OASIS file with similar patterns clustered together.

```
pdl_lib_mgr cluster input mix.pmdb output mix_clustered.pmdb
   line_ends length1 ">" 0.05 length2 ">=" 0.7 with_length "==" 0.05
   inside 0.1 outside 0.1 other_edges inside 0.001 outside 0.001

pdl_lib_mgr write_oasis input mix_clustered.pmdb output mix_clustered.oas
   cluster
```

Example 2

Output the marker named “Marker” to the OASIS layer named “marker” on layer 15. Generate one OASIS file per pattern.

```
pdl_lib_mgr write_oasis input myLib.pmdb marker Marker marker 15
   oasis_per_pattern
```

Example 3

Generate PNG images for each pattern in the pattern library and save the image files to the directory *myLibimages*. Also create an OASIS file named *myLib.oas* that includes all patterns.

```
pdl_lib_mgr write_oasis input myLib.pmdb output myLib.oas
   snapshot out_dir myLibimages
```

Example 4

Write properties in the input pattern library to the OASIS file, both as text and as OASIS properties attached to marker layer polygons. Two marker layer polygons are output, but properties are only attached to the layer for MarkerA.

```
pdl_lib_mgr write_oasis input myLib.pmdb output myLib.oas props_as_text
   marker MarkerA markA 15 attach_props
   marker MarkerB markB 16
```

Example 5

This example demonstrates two uses of the sort_by argument.

Case A: Sort the patterns by the dup_count property in each pattern. Specify descending so that the pattern with the highest dup_count value is placed in the lower left of the output OASIS layout. Specify props_as_text to print all property names and values below the pattern name.

```
pdl_lib_mgr write_oasis input lib_a2.pmdb output a2test.oas
   sort_by dup_count descending props_as_text
```

The SVRF commands DFM Pattern Capture and DFM Pattern Classify both have a DUP_COUNT keyword that adds a property with a duplicate pattern count.

Case B: Assume the pattern library has a unique property name “idx” for each pattern. The patterns are sorted by this property.

```
pdl_lib_mgr write_oasis input lib_a3.pmdb output a3test.oas
   sort_by idx ascending
```

Tip

These commands can add an index property to a pattern library:

- DFM Pattern Capture SVRF command, INDEX_PROP keyword
- unfold utility, index_prop argument
- merge utility, index argument

PMDB_info

Reports information about the input pattern library (PMDB).

Usage

pdl_lib_mgr PMDB_info input *filename* [details [markers]] [log_file *filename*]

Arguments

- **input *filename***

  A required argument that specifies the pattern matching database (PMDB). The file extension (.pmdb) is optional.

- details

  An optional argument that displays the pattern name, properties, keys, orientation, layer count, and count by pattern type.

- markers

  An optional argument that adds marker information to pattern information when the details option is specified. The unique marker count per library and marker names per pattern are added.

- log_file *filename*

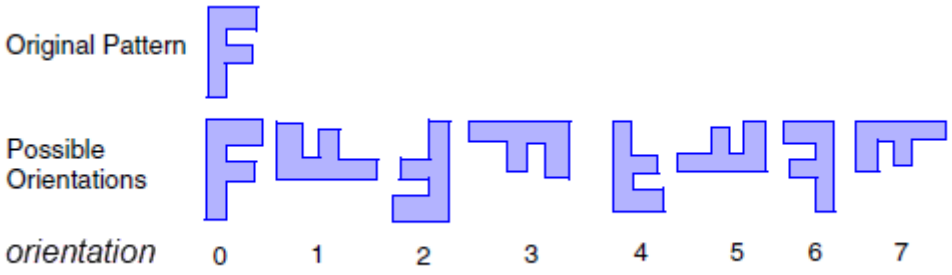
  An optional argument that specifies a log file to store the information output by the PMDB_info command.

Description

This utility reports information regarding the specified pattern library, such as the version that created the database.

When the details argument is specified, the reported information includes the orient_detail property, which gives the allowed orientations of a match to the pattern. The reported value is a string containing the setting (0 or 1) for the eight possible pattern orientations. A pattern that can match in all orientations has a value of 11111111.

Figure 5-11. orient_detail



Examples

```
pdl_lib_mgr PMDB_info input library.pmdb
```

Calibre MTflex Support for pdl_lib_mgr cluster

Calibre MTflex functionality provides a parallel processing data architecture for running multithreaded Calibre tools on distributed networked computers. When you run pdl_lib_mgr cluster with Calibre MTflex functionality, Calibre creates hierarchical threads and runs these threads across the network to remote computers. Calibre MTflex is only supported for the cluster utility.

To invoke pdl_lib_mgr cluster with Calibre MTflex support, do one of the following:

- Use -remote and specify each machine

  ```
  pdl_lib_mgr -turbo -remote machine1,machine2,machine3 cluster ...
  ```

- Use -remotefile and specify a configuration file

  ```
  pdl_lib_mgr -turbo -remotefile mtflex.config cluster ...
  ```

  where *mtflex.config* is the configuration file specifying the remote machines. For example:

  ```
  //mtflex.config file
  REMOTE HOST machine1
  REMOTE HOST machine2
  REMOTE HOST machine3
  ```

For more detailed information, see “Calibre MTflex Processing” and “Creating a Configuration File” in the in the *Calibre Administrator’s Guide.*

Index

— Q —

— R —

— S —

— T —

— U —

— V —

Third-Party Information

Details on open source and third-party software that may be included with this product are available in the *<your_software_installation_location>/legal* directory.