



SIEMENS EDA

Calibre® Pattern Matching Reference Manual

API Batch Command Support

Software Version 2021.2
Document Revision 14

Unpublished work. © 2021 Siemens

This material contains trade secrets or otherwise confidential information owned by Siemens Industry Software, Inc., its subsidiaries or its affiliates (collectively, "Siemens"), or its licensors. Access to and use of this information is strictly limited as set forth in Customer's applicable agreement with Siemens. This material may not be copied, distributed, or otherwise disclosed outside of Customer's facilities without the express written permission of Siemens, and may not be used in any way not expressly authorized by Siemens.

This document is for information and instruction purposes. Siemens reserves the right to make changes in specifications and other information contained in this publication without prior notice, and the reader should, in all cases, consult Siemens to determine whether any changes have been made. Siemens disclaims all warranties with respect to this document including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement of intellectual property.

The terms and conditions governing the sale and licensing of Siemens products are set forth in written agreements between Siemens and its customers. Siemens' **End User License Agreement** may be viewed at: www.plm.automation.siemens.com/global/en/legal/online-terms/index.html.

No representation or other affirmation of fact contained in this publication shall be deemed to be a warranty or give rise to any liability of Siemens whatsoever.

TRADEMARKS: The trademarks, logos, and service marks ("Marks") used herein are the property of Siemens or other parties. No one is permitted to use these Marks without the prior written consent of Siemens or the owner of the Marks, as applicable. The use herein of third party Marks is not an attempt to indicate Siemens as a source of a product, but is intended to indicate a product from, or associated with, a particular third party. A list of Siemens' trademarks may be viewed at: www.plm.automation.siemens.com/global/en/legal/trademarks.html. The registered trademark Linux[®] is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a world-wide basis.

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 <i>Calibre Release Notes</i> 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 <i>Calibre Release Notes</i> 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 <i>Calibre Release Notes</i> 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 <i>Calibre Release Notes</i> 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

Revision History

Chapter 1

Introduction to the Calibre Pattern Matching Batch Commands 11

Overview of Calibre Pattern Matching Batch Commands 11

Requirements for the Calibre Pattern Matching API. 11

Syntax Conventions 13

Chapter 2

Getting Started with the Calibre Pattern Matching API 15

Starting an Interactive Session for the Pattern Matching Tcl API. 16

Running a Pattern Matching API Script 17

Tcl Basics for Pattern Matching Batch Commands. 18

Chapter 3

Batch Commands for Pattern Matching 21

Library Commands 22

 pmatch::create_pattern_lib 23

 pmatch::get_layer_names 25

 pmatch::get_pattern_names 26

 pmatch::get_src_layer_num 27

 pmatch::load_pattern_lib 29

 pmatch::remove_pattern 31

 pmatch::rename_layers 33

 pmatch::rename_pattern 34

 pmatch::set_active_library 35

 pmatch::set_src_layer_num 36

 pmatch::update_pattern. 38

Pattern Data Retrieval 40

 pmatch::check_pattern 41

 pmatch::get_class 42

 pmatch::get_pattern 43

 pmatch::get_pattern_name 44

 pmatch::get_precision. 45

Pattern Markers 46

 pmatch::add_custom_bbox_marker 47

 pmatch::add_custom_marker_rect 49

 pmatch::add_marker 50

 pmatch::delete_marker 51

 pmatch::get_markers. 52

 pmatch::rename_marker 53

Pattern Geometries. 54

pmatch::get_edges	55
pmatch::get_vertices	57
Pattern Extents	59
pmatch::get_extent_type	60
pmatch::get_extent_bbox	61
pmatch::size_up_extent	62
pmatch::snap_extent	63
Pattern Properties, Keys, and Constraints	64
pmatch::add_int_attr	66
pmatch::add_keys	68
pmatch::add_pattern_orient	69
pmatch::add_properties	70
pmatch::add_text_attribute	72
pmatch::delete_constraint_label	74
pmatch::delete_keys	76
pmatch::delete_properties	78
pmatch::get_cglobal	80
pmatch::get_constraints	81
pmatch::get_constraint_label	83
pmatch::get_int_attr	85
pmatch::get_keys	86
pmatch::get_library_attribute	88
pmatch::get_pattern_orient	89
pmatch::get_properties	90
pmatch::get_property_value	92
pmatch::get_text_attribute	93
pmatch::set_cglobal	94
pmatch::set_constraint_label	96
pmatch::set_library_attribute	98
pmatch::set_pattern_orient	99
Groups and Filters	101
pmatch::filter_add	102
pmatch::filter_create	104
pmatch::filter_delete	105
pmatch::filter_reset	106
pmatch::group_add_filtered	107
pmatch::group_add_pattern	109
pmatch::group_create	111
pmatch::group_delete	113
pmatch::group_reset	114
pmatch::group_save	116
pmatch::list_group_patterns	118
Help Command	120
pmatch::help	121

Index

Third-Party Information

List of Figures

Figure 3-1. Orientations for add_pattern_orient 69

Figure 3-2. Orientations for set_pattern_orient 99

List of Tables

Table 1-1. Syntax Conventions	13
Table 2-1. Tcl Special Characters	19

Chapter 1

Introduction to the Calibre Pattern Matching Batch Commands

You can use the Calibre® Pattern Matching Tcl-based API batch commands to access and modify pattern libraries without using the Calibre Pattern Matching GUI. Using the pattern matching API improves the loading time of large pattern libraries when compared with the Calibre Pattern Matching GUI.

Overview of Calibre Pattern Matching Batch Commands	11
Requirements for the Calibre Pattern Matching API	11
Syntax Conventions	13

Overview of Calibre Pattern Matching Batch Commands

The Calibre Pattern Matching tool includes Tcl API commands that read and modify pattern libraries. You can develop Tcl procedures that automate tasks that would be repetitive to perform using the Calibre Pattern Matching GUI. Pattern matching API scripts are run using the `pdl_lib_mgr` prompt utility.

You can use the Tcl API to traverse pattern libraries and the pattern objects within the library. You can examine the contents of patterns, such as properties, attributes, and markers, and make changes.

Information on running pattern matching, using the Calibre Pattern Matching GUI, and using the pattern library manager utilities is available in the [Calibre Pattern Matching User's Manual](#).

Related Topics

[Tcl Library in Calibre \[Calibre Administrator's Guide\]](#)

Requirements for the Calibre Pattern Matching API

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

- **Licensing**

A Calibre Pattern Matching license is required. See “[Calibre Pattern Matching](#)” in the *Calibre Administrator’s Guide*.

- **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*.

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.
<i>Italic</i>	Italic fonts indicate a user-supplied argument.
Monospace	Monospace fonts indicate a shell command, line of code, or URL. A bold monospace font identifies text you enter.
<u>Underline</u>	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.
...	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 { <i>element_name</i> [‘(‘ <i>model_name</i> ‘)’]} <i>device_layer</i> { <i>pin_layer</i> [‘(‘ <i>pin_name</i> ‘)’] ...} [‘<‘ <i>auxiliary_layer</i> ‘>’ ...] [‘(‘ <i>swap_list</i> ‘)’ ...] [<u>BY NET</u> BY SHAPE]	

Chapter 2

Getting Started with the Calibre Pattern Matching API

You can run Calibre Pattern Matching Tcl API commands interactively from the API command prompt or execute a Tcl script that contains the API commands.

Note



The pattern matching API uses “%” as the prompt character, which is typically used as the Linux[®] C shell prompt. This documentation uses “\$” for the Linux command line prompt.

Output from an API command is shown without the “%” prompt character.

It is not required to use the `pmatch::` namespace with the API commands, although most examples do so.

- To start an interactive API session, enter the following at the Linux command prompt:

```
$ pdl_lib_mgr prompt
// Calibre PDL Lib Manager v2018.2 ...
%
```

“Calibre PDL Lib Manager” is printed with the Calibre version and the standard Calibre banner, then the API “%” prompt character is printed. You enter API commands at the prompt.

- To run a Tcl script named *myScript.tcl*, enter the following at the Linux command prompt:

```
$ pdl_lib_mgr prompt myScript.tcl
```

The Calibre banner is printed, the script is run and any command output is printed, then the API session ends and returns to the Linux prompt:

```
// Calibre PDL Lib Manager v2018.2 ...
...
<script output>
$
```

1. Linux[®] is a registered trademark of Linus Torvalds in the U.S. and other countries.

- To execute the *myScript.tcl* from within the API session, use the Tcl source command:

```
$ pdl_lib_mgr prompt
// Calibre PDL Lib Manager v2018.2 ...
% source myScript.tcl
<script output>
%
```

The script executes and returns to the API session prompt.

Starting an Interactive Session for the Pattern Matching Tcl API.....	16
Running a Pattern Matching API Script.....	17
Tcl Basics for Pattern Matching Batch Commands.....	18

Starting an Interactive Session for the Pattern Matching Tcl API

The example starts an interactive session of the Pattern Matching Tcl API, loads a pattern matching library, and outputs the number of patterns in the library.

Prerequisites

- The correct licenses and environment variables. See “[Requirements for the Calibre Pattern Matching API](#)” on page 11.
- Calibre version 2013.2 or later.
- A pattern matching library.

Procedure

1. At the Linux command line, invoke the pattern matching API.

```
$ pdl_lib_mgr prompt
// Calibre PDL Lib Manager v2018.2 ...
%
```

The “%” pattern matching API prompt appears.

2. At the API prompt, use the [pmatch::load_pattern_lib](#) command to load the library *example_lib.pmdb*. Save the returned library handle to the Tcl variable *x_lib*.

```
% set x_lib [pmatch::load_pattern_lib -path example_lib.pmdb]
Pattern Library DATABASE
```

3. Save the list of pattern names to the *patterns* variable:

```
% set patterns [pmatch::get_pattern_names -lib $x_lib]
p1_exact p4_bcm p5_bcm
```


4. Use the Tcl `llength` command to get the length of the pattern name list:

```
% set num_patterns [llength $patterns]
3
```

5. Use the Tcl `puts` command to print the number of patterns:

```
% puts "There are $num_patterns patterns in this library"
There are 3 patterns in this library
```

6. Exit the API session and return to the Linux prompt:

```
% exit
$
```

Running a Pattern Matching API Script

The script in this example loads a pattern matching library and outputs the number of patterns in the library.

Prerequisites

- The correct licenses and environment variables. See [“Requirements for the Calibre Pattern Matching API”](#) on page 11.
- Calibre version 2013.2 or later.
- A pattern matching library.

Procedure

1. Create a text file containing the API commands. This example uses the same commands as those in [“Starting an Interactive Session for the Pattern Matching Tcl API”](#) on page 16.

```
set x_lib [pmatch::load_pattern_lib -path example_lib.pmdb]
set patterns [pmatch::get_pattern_names -lib $x_lib]
set num_patterns [llength $patterns]
puts "There are $num_patterns patterns in this library"
```


Save the file as *example.tcl*.

2. Execute the script from the Linux command line.

```
$ pdl_lib_mgr prompt example.tcl
// Calibre PDL Lib Manager v2018.2 ...
... < rest of Calibre banner >
There are 3 patterns in this library
$
```

The script executes and prints the `puts` output, then exits.

Tip

 You can also execute a script from within the API session using the Tcl source command:

```
% source example.tcl
```

Tcl Basics for Pattern Matching Batch Commands

If you are not familiar with Tcl coding, this section described basics, such as command syntax and special characters.

Tcl characteristics:

- **Tcl is case sensitive** — The commands in the `pmatch::` namespace must be used as shown.
- **Statements can span multiple lines** — The statement continuation character for multiple line statements is the backslash (`\`) character, which must terminate each continued line. There cannot be any whitespace characters after the backslash.
- **Comments are a type of command** — In Tcl, comments are indicated by `"#"`. It must be the first non-whitespace character of the command; nothing after it on the line is acted upon. However, the comment line is parsed, and mismatched braces and parentheses can cause Tcl errors.
- **Tcl lists** — The list is a basic data structure in Tcl, and is used in several API commands. A Tcl list is an ordered collection of numbers, strings, or other lists.

A Tcl list of single elements can be specified in these equivalent ways:

```
"a b c d"  
{a b c d}  
[list a b c d]
```

A list of lists can be specified in these equivalent ways:


```
{{a1 a2} {b1 b2} {c1 c2} {d1 d2}}  
[list [list a1 a2] [list b1 b2] [list c1 c2] [list d1 d2]]
```

- **Many characters have special meanings in Tcl** — Consult any online reference for Tcl for a complete list of special characters. The following are the special characters used in this chapter:

Table 2-1. Tcl Special Characters

Character	Meaning
;	The semicolon terminates the previous command, allowing you to place more than one command on the same line.
\	The backslash causes backslash substitution, or, when within double quotes, disables substitution for the single character immediately following the backslash. The backslash is also the line continuation character. Be careful not to have whitespace on the same line after a backslash that terminates a line, as this can cause Tcl interpretation errors.
\n	The backslash with the letter “n” creates a new line.
#	The pound sign indicates the start of a comment.
\$	The dollar sign in front of a variable name instructs the Tcl interpreter to access the value stored in the variable.
[]	Brackets cause command substitution, instructing the Tcl interpreter to treat everything within the brackets as the result of a command. Command substitution can be nested within words.
{ }	Braces instruct the Tcl interpreter to treat the enclosed words as a single string. The Tcl interpreter accepts the string as is, without performing any variable substitution or evaluation.
" "	Quotes instruct the Tcl interpreter to treat the enclosed words as a single string. However, if the Tcl interpreter encounters variables or commands within the string in quotes, it first evaluates the variables and commands.

Note

 Braces and square brackets are used in both syntax definitions and Tcl constructs. If a syntax definition includes a Tcl construct with brackets or braces, the Tcl elements are enclosed in single quotes to indicate a literal element. For example, the following syntax indicates the square brackets are literal within a syntax definition:

```
'[ list a b c ]'
```

Related Topics

[Tcl Library in Calibre \[Calibre Administrator's Guide\]](#)

Chapter 3

Batch Commands for Pattern Matching

All supported API batch commands available in Calibre pattern matching are arranged according to functional areas, such as pattern library commands, pattern data retrieval, and pattern markers.

Library Commands	22
Pattern Data Retrieval	40
Pattern Markers	46
Pattern Geometries	54
Pattern Extents	59
Pattern Properties, Keys, and Constraints	64
Groups and Filters	101
Help Command	120

Library Commands

A pattern library is a collection of one or more patterns. A pattern defines the polygons Calibre searches for in a pattern matching run. When the pattern library is complete, the pattern library is compiled into an SVRF DMACRO pattern file for use in the pattern matching run.

To create a new pattern library, you must have the following:

- Correct licenses for Calibre Pattern Matching
- Environment variables set to run Calibre

For information on creating and using libraries using the Calibre Pattern Matching GUI, refer to the [Calibre Pattern Matching User's Manual](#).

pmatch::create_pattern_lib	23
pmatch::get_layer_names	25
pmatch::get_pattern_names	26
pmatch::get_src_layer_num	27
pmatch::load_pattern_lib	29
pmatch::remove_pattern	31
pmatch::rename_layers	33
pmatch::rename_pattern	34
pmatch::set_active_library	35
pmatch::set_src_layer_num	36
pmatch::update_pattern	38

pmatch::create_pattern_lib

Create a pattern library and save it to the specified file path.

Usage

```
pmatch::create_pattern_lib -path library_path -layernames layer_names_list  
[-overwrite]
```

Arguments

- **-path** *library_name*
A required argument set that specifies the name of the new pattern library. The filename extension *.pmdb* is added automatically if it is not included.
- **-layernames** *layer_names_list*
A required argument set that specifies a Tcl list of layers used in the library.
- **-overwrite**
An optional argument that specifies to overwrite the file at *library_path* if it exists.

Return Values

A pattern library handle, or -1 if the command fails.

Description

Creates a new pattern library with the layer names in *layer_names_list* and saves the library to *library_path*. This command sets the new library as the active pattern library, which makes use of the -lib argument in other commands optional.

If the *library_path* argument is an existing pattern library, the command fails unless -overwrite is specified.

Examples

Example 1

Create a new library and save it to the file *example_lib.pmdb*. The pattern library handle is loaded into the variable *x_lib*.

```
% set x_lib [pmatch::create_pattern_lib -path example_lib.pmdb \  
          -layernames "active m1 contact"]  
Pattern Library DATABASE
```

Example 2

Create a new library, where one of the layer names is specified with a Tcl variable.

```
set lay_2 "poly"  
set lib_handle_new [create_pattern_lib -path newpat_lib \  
          -layernames "active $lay_2 contact"]
```

Related Topics

[pmatch::load_pattern_lib](#)

[pmatch::set_active_library](#)

[pmatch::get_pattern_names](#)

`pmatch::get_layer_names`

Get all pattern layer names in a pattern library.

Usage

`pmatch::get_layer_names` [-lib *library_handle*]

Arguments

- -lib *library_handle*

An optional argument that specifies a pattern library handle returned by [`pmatch::create_pattern_lib`](#) or [`pmatch::load_pattern_lib`](#). The active library is used if this argument is not specified.

Return Values

Tcl list of layer names in the following format:

`{layer_count {layer_names_list} {errors}}`

Returns -1 if the command fails.

Examples

Output the names of the layers in a pattern library.

```
% set x_lib [pmatch::load_pattern_lib -path myLib.pmdb]
Pattern Library DATABASE
% set layer_names [pmatch::get_layer_names]
3 {aaa bbb ccc} {}
```

The library has three pattern layers, with names “aaa”, “bbb”, and “ccc”, and no errors.

Related Topics

[`pmatch::get_pattern_names`](#)

[`pmatch::rename_layers`](#)

`pmatch::get_pattern_names`

Get all pattern names from a pattern library.

Usage

`pmatch::get_pattern_names` [-lib *library_handle*]

Arguments

- -lib *library_handle*

An optional argument that specifies a pattern library handle returned by [pmatch::create_pattern_lib](#) or [pmatch::load_pattern_lib](#). The active library is used if this argument is not specified.

Return Values

Tcl list of all pattern names in the library, or -1 if the command fails.

Examples

Load a pattern-matching library, and output the names of every pattern in the library.

```
% set x_lib [pmatch::load_pattern_lib -path example_lib.pmdb]
Pattern Library DATABASE
% set patterns [pmatch::get_pattern_names -lib $x_lib]
p1_tem p2_tem p3_bcm
```

Related Topics

[pmatch::load_pattern_lib](#)

pmatch::get_src_layer_num

Gets the list of source layer numbers for a pattern library, or the source layer number for a specified pattern layer. The source layer number is the layout layer number associated with a pattern layer.

Usage

pmatch::get_src_layer_num [-lib *library_handle*] [-layer *layer_index*]

Arguments

- -lib *library_handle*

An optional argument set that specifies a pattern library handle returned by [pmatch::create_pattern_lib](#) or [pmatch::load_pattern_lib](#). The active library is used if this argument is not specified.

- -layer *layer_index*

An optional argument set that specifies a layer index in the list of pattern layers for the library. Index values start at 1 for pattern layers.

If -layer is specified, the source layer number for the layer at *layer_index* is returned. Otherwise a list of all source layer numbers is returned.

Return Values

- Without -layer — A Tcl list of source layer numbers ordered by layer index, where { } is returned if there is no source layer number for the layer index.
- With -layer — One source layer number, or { } if there is no source layer number for *layer_index*.

The command returns -1 on error.

Examples

Get all source layers, print the source layer for pattern layer1, set the source layer for pattern layer 2, then remove the source layer for pattern layer 2, using the following Tcl script *sources.tcl*:

```
pmatch::load_pattern_lib -path newlib.pmdb
set src_layer_nums [pmatch::get_src_layer_num]
puts "source layers: $src_layer_nums"

set src_layers_num [pmatch::get_src_layer_num -layer 1]
puts "source layer for first layer: $src_layers_num"
```

```
puts "setting layer 2 source to 13.2"
set ret_val [pmatch::set_src_layer_num -layer 2 -value 13.2]
puts "returned: $ret_val"
# returns empty string on success
set src_layer_nums [pmatch::get_src_layer_num]
puts "new source layers: $src_layer_nums"

puts "setting layer 2 source to to empty"
set ret_val [pmatch::set_src_layer_num -layer 2 -value ""]
puts "returned: $ret_val"
set src_layer_nums [pmatch::get_src_layer_num]
puts "new source layers: $src_layer_nums"
```

Run the script:

pdl_lib_mgr prompt sources.tcl

The following output is produced. The library has two pattern layers. Layer 1 has a source layer number of 2. Layer 2 does not have a source layer number.

```
source layers: 2 {}
source layer for first layer: 2
setting layer 2 source to 13.2
returned:
new source layers: 2 13.2
setting layer 2 source to to empty
returned:
new source layers: 2 {}
```

Related Topics

[pmatch::set_src_layer_num](#)

pmatch::load_pattern_lib

Load a pattern library for reading or both reading and writing.

Usage

pmatch::load_pattern_lib -path *library_path* [-mode {r | rw}]

Arguments

- **-path *library_path***
A required argument that specifies the path to the pattern library.
- **-mode {r | rw}**
An optional argument that specifies to open the file for either read-only (r) or for read/write (rw) access. Read/write access is the default.

Return Values

A pattern library handle, or -1 if the command fails.

Description

Loads a pattern library for reading or both reading and writing. The active pattern library is set to the specified library, which makes use of the -lib argument in other commands optional. The command generates an error if the pattern matching library file does not exist.

The library is opened with exclusive access, so that no other application is able to access the library while it is open.

Examples

Load two pattern matching libraries.

```
% set w_lib [pmatch::load_pattern_lib -path sqlib.pmdb]
Pattern Library DATABASE
% set x_lib [pmatch::load_pattern_lib -path relib.pmdb]
Pattern Library DATABASE
```

Attempt to load a pattern-matching library file that does not exist.

```
% set y_lib [pmatch::load_pattern_lib -path bogus.pmdb]
Invalid PMDB Library path.
-1
```

Load a pattern-matching library, and output the number of patterns in the library.

```
% set z_lib [pmatch::load_pattern_lib -path exlib.pmdb]
Pattern Library DATABASE
% set patterns [pmatch::get_pattern_names]
p1_tem p2_tem p3_bcm
% set num_patterns [llength $patterns]
3
% puts "There are $num_patterns patterns in this library"
There are 3 patterns in this library
```

Related Topics

[pmatch::create_pattern_lib](#)

[pmatch::set_active_library](#)

[pmatch::get_pattern_names](#)

pmatch::remove_pattern

Delete a pattern from a pattern library.

Usage

pmatch::remove_pattern -name *pattern_name* [-lib *library_handle*]

Arguments

- **-name** *pattern_name*
A required argument that specifies the name of the pattern to remove.
- **-lib** *library_handle*
An optional argument that specifies a pattern library handle returned by [pmatch::create_pattern_lib](#) or [pmatch::load_pattern_lib](#). The active library is used if this argument is not specified.

Return Values

Returns 1 if the command is successful, or -1 if the command fails.

Description

Deletes a pattern from a pattern library.

Caution



This operation cannot be undone.

Examples

Remove the first pattern from a pattern library.

```
% pmatch::load_pattern_lib -path example_lib.pmdb
Pattern Library DATABASE
% set pat_names [pmatch::get_pattern_names]
p1_tem p2_tem p3_bcm

% set first_pat_name [lindex $pat_names 0]
p1_tem

% pmatch::remove_pattern -name $first_pat_name
1

% pmatch::get_pattern_names
p2_tem p3_bcm
```

Related Topics

[pmatch::get_pattern](#)

[pmatch::rename_pattern](#)

[pmatch::update_pattern](#)

pmatch::rename_layers

Rename the layers in a pattern library.

Usage

pmatch::rename_layers -layernames *layer_names* [-lib *library_handle*]

Arguments

- **-layernames** *layer_names*
A required argument that specifies a Tcl list of layer names. The number of layer names must match the number of layers in the pattern library.
- **-lib** *library_handle*
An optional argument that specifies a pattern library handle returned by [pmatch::create_pattern_lib](#) or [pmatch::load_pattern_lib](#). The active library is used if this argument is not specified.

Return Values

Returns 1 if the command is successful, or -1 if the command fails.

Description

Renames the layers in a pattern library. The new layer names are used for all patterns in the library.

You cannot use this command to add layers to the pattern library.

Examples

Rename layers in a pattern library. The library has three pattern layers.

```
% set x_lib [pmatch::load_pattern_lib -path example_lib.pmdb]
Pattern Library DATABASE
% pmatch::rename_layers -layernames {g1 g2 r3}
% set layer_names [pmatch::get_layer_names]
3 {g1 g2 r3} {}
```

Related Topics

[pmatch::load_pattern_lib](#)

pmatch::rename_pattern

Rename the specified pattern.

Usage

```
pmatch::rename_pattern -original_name orig_name -new_name new_name  
[-lib library_handle]
```

Arguments

- **-original_name *orig_name***
A required argument that specifies the original name of the pattern to rename.
- **-new_name *new_name***
A required argument that specifies the new name of the pattern.
- **-lib *library_handle***
An optional argument that specifies a pattern library handle returned by [pmatch::create_pattern_lib](#) or [pmatch::load_pattern_lib](#). The active library is used if this argument is not specified.

Return Values

Returns 1 if the command is successful, or -1 if the command fails.

Examples

Rename a pattern in a pattern library.

```
% pmatch::load_pattern_lib -path example_lib.pmdb  
Pattern Library DATABASE  
% set pat_names [pmatch::get_pattern_names]  
p1_tem p2_tem p3_bcm  
  
% set first_pat_name [lindex $pat_names 0]  
p1_tem  
  
% set newname "p_abc"  
% pmatch::rename_pattern -original_name $first_pat_name -new_name $newname  
1  
  
% pmatch::get_pattern_names  
p_abc p2_tem p3_bcm
```

Related Topics

[pmatch::update_pattern](#)

`pmatch::set_active_library`

Specify the active library.

Usage

`pmatch::set_active_library -lib library_handle`

Arguments

- **`-lib library_handle`**

A required argument that specifies a pattern library handle returned by [pmatch::create_pattern_lib](#) or [pmatch::load_pattern_lib](#).

Return Values

Returns 1 if the command is successful, or -1 if the command fails.

Description

Specifies the active library.

Many commands have an optional `-lib` argument to specify the pattern library. If the `-lib` argument is not specified in such commands, the active library is used.

Examples

Change the active pattern matching library.

```
% set t_lib [pmatch::load_pattern_lib -path tem_lib.pmdb]
Pattern Library DATABASE
% set b_lib [pmatch::load_pattern_lib -path bcm_lib.pmdb]
Pattern Library DATABASE

% set patterns [pmatch::get_pattern_names]
p1_bcm p2_bcm

% pmatch::set_active_library -lib $t_lib
1
% set patterns [pmatch::get_pattern_names]
p1_exact p2_tem p3_tem
```

Related Topics

[pmatch::load_pattern_lib](#)

[pmatch::create_pattern_lib](#)

`pmatch::set_src_layer_num`

Set the source layer number for a specified pattern layer. The command overwrites any previously specified source layer number. The source layer number is the layout layer number associated with a pattern layer.

Usage

`pmatch::set_src_layer_num` [-lib *library_handle*]
-layer *layer_index* **-value** *src_layer_number*

Arguments

- **-lib** *library_handle*
An optional argument set that specifies a pattern library handle returned by [`pmatch::create_pattern_lib`](#) or [`pmatch::load_pattern_lib`](#). The active library is used if this argument is not specified.
- **-layer** *layer_index*
A required argument set that specifies a layer index in the list of pattern layers for the library. Index values start at 1 for pattern layers.
- **-value** *src_layer_number*
A required argument set that specifies the source layer number to associate with *layer_index*. The *src_layer_number* can be specified as an integer layer number, or a layer number and datatype separated by a decimal point. For example: 25, or 30.1.

Return Values

Returns an empty string if the command is successful, or -1 if the command fails.

Examples

Get all source layer numbers, print the source layer number for pattern layer1, set the source layer number for pattern layer 2, then remove the source layer number for pattern layer 2, using the following Tcl script *sources.tcl*:

```
pmatch::load_pattern_lib -path newlib.pmdb
set src_layer_nums [pmatch::get_src_layer_num]
puts "source layers: $src_layer_nums"

set src_layers_num [pmatch::get_src_layer_num -layer 1]
puts "source layer for first layer: $src_layers_num"
```

```
puts "setting layer 2 source to 13.2"
set ret_val [pmatch::set_src_layer_num -layer 2 -value 13.2]
puts "returned: $ret_val"
# returns empty string on success
set src_layer_nums [pmatch::get_src_layer_num]
puts "new source layers: $src_layer_nums"

puts "setting layer 2 source to to empty"
set ret_val [pmatch::set_src_layer_num -layer 2 -value ""]
puts "returned: $ret_val"
set src_layer_nums [pmatch::get_src_layer_num]
puts "new source layers: $src_layer_nums"
```

Run the script:

pdl_lib_mgr prompt sources.tcl

The following output is produced. The library has two pattern layers. Layer 1 has a source layer number of 2. Layer 2 does not have a source layer number.

```
source layers: 2 {}
source layer for first layer: 2
setting layer 2 source to 13.2
returned:
new source layers: 2 13.2
setting layer 2 source to to empty
returned:
new source layers: 2 {}
```

Related Topics

[pmatch::get_src_layer_num](#)

pmatch::update_pattern

Update the specified pattern with current pattern data.

Usage

pmatch::update_pattern -pattern *pattern_handle* [-lib *library_handle*]

Arguments

- **-pattern *pattern_handle***
A required argument set that specifies a pattern handle, which is the first element returned from [pmatch::get_pattern](#).
- **-lib *library_handle***
An optional argument that specifies a pattern library handle returned by [pmatch::create_pattern_lib](#) or [pmatch::load_pattern_lib](#). If not specified, the active library is used.

Return Values

Returns 1 if the command is successful, or -1 if the command fails.

Description

Updates the specified pattern in the pattern library. If the pattern exists in the pattern library, all data associated with the pattern is replaced with the current pattern data. If the pattern is new, it is added to the library.

Changes made to patterns using the API are saved in memory, but are not persistent until the `pmatch::update_pattern` command is called.

Note



If you are running from a shell that has the Calibre Pattern Matching GUI open, the GUI is not updated automatically by this command. You need to reload the library if you want the GUI to use the updated pattern.

Examples

Update one pattern in a library, then save the updated pattern.

```
% pmatch::load_pattern_lib -path example_lib.pmdb
Pattern Library DATABASE
% set pat_names [pmatch::get_pattern_names]
p1_tem p2_tem p3_bcm

% set patx [pmatch::get_pattern -name [lindex $pat_names 0]]
{PDL Pattern} {}
% set patx_handle [lindex $patx 0]
PDL Pattern
```

```
# Modify the pattern using pmatch::add_keys
% pmatch::add_keys -pattern $patx_handle -keys_list {k1 k2}
1

# Save the changes.
% pmatch::update_pattern -pattern $patx_handle
1
```

Related Topics

[pmatch::rename_pattern](#)

Pattern Data Retrieval

These commands return pattern data for later processing.

To retrieve pattern data, you must first load a pattern library using the [pmatch::create_pattern_lib](#) or [pmatch::load_pattern_lib](#) command.

For information on creating and using libraries using the Calibre Pattern Matching GUI, refer to the [Calibre Pattern Matching User's Manual](#).

pmatch::check_pattern	41
pmatch::get_class	42
pmatch::get_pattern	43
pmatch::get_pattern_name	44
pmatch::get_precision	45

pmatch::check_pattern

Check a pattern for compilation errors and warnings.

Usage

pmatch::check_pattern -pattern *pattern_handle*

Arguments

- **-pattern** *pattern_handle*

A required argument specifying the pattern handle.

Return Values

List of two sublists, with the first containing the compiler errors and the second containing the warnings.

Examples

Check all patterns in a pattern library for compilation errors and warnings. The example library has three patterns. The three lines of double braces at the end are the output of the `check_pattern` command, indicating that there are no errors or warnings.

```
% set x_lib [pmatch::load_pattern_lib -path example_lib.pmdb]
Pattern Library DATABASE
% set pat_names [pmatch::get_pattern_names -lib $x_lib] \
    p1_tem p2_tem p3_bcm

% foreach patternName $pat_names {
    set patx [pmatch::get_pattern -name $patternName]
    set patx_handle [lindex $patx 0]
    puts [pmatch::check_pattern -pattern $patx_handle]
}
{} {}
{} {}
{} {}
```

Related Topics

[pmatch::get_pattern](#)

[pmatch::get_precision](#)

pmatch::get_class

Get the pattern class for the specified pattern.

Note



Pattern classes are described in “[Patterns](#)” in the *Calibre Pattern Matching User’s Manual*.

Usage

pmatch::get_class -pattern *pattern_handle* [-lib *library_handle*]

Arguments

- **-pattern *pattern_handle***
A required argument that specifies a pattern handle, which is the first element returned from [pmatch::get_pattern](#).
- **-lib *library_handle***
An optional argument that specifies a pattern library handle returned by [pmatch::create_pattern_lib](#) or [pmatch::load_pattern_lib](#). The active library is used if this argument is not specified.

Return Values

The pattern class on success, or -1 if the command fails. The pattern class is one of the following: TEM1, TEM2, XBAR, BCM1, BCM2, or INVALID.

Examples

Get the pattern class for the first pattern in the library.

```
% pmatch::load_pattern_lib -path example_lib.pmdb
Pattern Library DATABASE
% set pat_names [pmatch::get_pattern_names]
p1_tem p2_tem p3_bcm

# Get the handle for the first pattern in the pattern library
% set patx [pmatch::get_pattern -name [lindex $pat_names 0]]
{PDL Pattern} {}
% set patx_handle [lindex $patx 0]
PDL Pattern

% puts [pmatch::get_class -pattern $patx_handle]
TEM1
```

pmatch::get_pattern

Load a single pattern from a pattern library.

Usage

pmatch::get_pattern -name *pattern_name* [-lib *library_handle*]

Arguments

- **-name *pattern_name***
A required argument that specifies the name of the pattern to load.
- **-lib *library_handle***
An optional argument that specifies a pattern library handle returned by [pmatch::create_pattern_lib](#) or [pmatch::load_pattern_lib](#). The active library is used if this argument is not specified.

Return Values

A Tcl list of two elements:

- The first is a pattern handle if the command is successful, or -1 if the command fails.
- The second is a list containing any errors or warnings that result from parsing the pattern. The string is empty if the first element is -1.

Examples

Example 1

Check all patterns in a pattern library for compiler errors and warnings.

```
set pat_lib [pmatch::load_pattern_lib -path myLib2_error.pmdb]
set pattern_list [pmatch::get_pattern_names -lib $pat_lib]
foreach name $pattern_list {

    set pat [pmatch::get_pattern -name $name]
    set pat_handle [lindex $pat 0]
    set pat_errs [lindex $pat 1]
    if {$pat_errs != ""} {
        puts "pattern $name has error or warning: $pat_errs"
    }

}
```

Related Topics

[pmatch::get_pattern_names](#)

[pmatch::check_pattern](#)

`pmatch::get_pattern_name`

Get the name of the pattern for the specified pattern handle.

Usage

`pmatch::get_pattern_name` **-pattern** *pattern_handle* [-lib *library_handle*]

Arguments

- **-pattern** *pattern_handle*
A required argument that specifies a pattern handle, which is the first element returned from [pmatch::get_pattern](#).
- **-lib** *library_handle*
An optional argument that specifies a pattern library handle returned by [pmatch::create_pattern_lib](#) or [pmatch::load_pattern_lib](#). The active library is used if this argument is not specified.

Return Values

A pattern name, or -1 if the command fails.

Examples

Get the pattern name for the first pattern in the library.

```
% pmatch::load_pattern_lib -path example_lib.pmdb
Pattern Library DATABASE
% set pat_names [pmatch::get_pattern_names]
p1_tem p2_tem p3_bcm

# Get the handle for the first pattern in the pattern library
% set patx [pmatch::get_pattern -name [lindex $pat_names 0]]
{PDL Pattern} {}
% set patx_handle [lindex $patx 0]
PDL Pattern

% puts [pmatch::get_pattern_name -pattern $patx_handle]
p1_tem
```

Related Topics

[pmatch::get_pattern_names](#)

pmatch::get_precision

Get the precision of a pattern.

Usage

pmatch::get_precision -pattern *pattern_handle*

Arguments

- **-pattern** *pattern_handle*
A required argument specifying the pattern handle.

Return Values

The precision value of the pattern.

Examples

Check the precision values of all patterns in a pattern library.

```
% set x_lib [pmatch::load_pattern_lib -path example_lib.pmdb]
Pattern Library DATABASE
% set pat_names [pmatch::get_pattern_names -lib $x_lib]
p1_tem p2_tem p3_bcm

% foreach patternName $pat_names {
    set patx [pmatch::get_pattern -name $patternName]
    set patx_handle [lindex $patx 0]
    puts [pmatch::get_precision -pattern $patx_handle]
}
1000.0
1000.0
1000.0
```

The three patterns each have a precision of 1000.0.

Related Topics

[pmatch::get_pattern](#)

[pmatch::check_pattern](#)

Pattern Markers

These commands enable the processing of pattern marker layers. Marker layers are output layers generated by the pattern engine for pattern matches. Markers can be added to a pattern to show where a match was found.

To retrieve pattern markers, you must first load a pattern library using the [pmatch::create_pattern_lib](#) or [pmatch::load_pattern_lib](#) command.

For information on creating and using libraries using the Calibre Pattern Matching GUI, refer to the [Calibre Pattern Matching User's Manual](#).

pmatch::add_custom_bbox_marker	47
pmatch::add_custom_marker_rect	49
pmatch::add_marker	50
pmatch::delete_marker	51
pmatch::get_markers	52
pmatch::rename_marker	53

pmatch::add_custom_bbox_marker

Add a custom marker that is undersized from the pattern bounding box.

Usage

```
pmatch::add_custom_bbox_marker -pattern pattern_handle -name marker_name  
-size_under value
```

Arguments

- **-pattern *pattern_handle***
A required argument set that specifies a valid pattern handle.
- **-name *marker_name***
A required argument set that specifies the new custom marker name. The new marker name should not exist in the pattern.
- **-size_under *value***
A required argument set that specifies the marker edge shift from the pattern bounding box. Specify a positive value for *value* in user units.

Return Values

Returns 1 if the command is successful, or -1 if the command fails.

Description

Adds a custom marker to a pattern. The marker edges are undersized from the pattern bounding box edges by the specified **-size_under** value. The custom marker is always a rectangle. If the **-size_under** value is too large, the command fails and a custom marker is not created.

The pattern bounding box encloses all layer geometries and extents, but does not include regions that lie outside of the pattern extent.

All patterns in the library must have the same number of markers. Therefore, if you use this command, you must apply it to all patterns in the library.

Examples

Add custom markers to the patterns in a library.

```
foreach name $pattern_names_list {  
  set pat_handle [pmatch::get_pattern -name $name]  
  set pat_handle [lindex $pat_handle 0]  
  pmatch::add_custom_bbox_marker -pattern $pat_handle -name bbox_custom \  
    -under_size 0.02  
  pmatch::update_pattern -pattern $pat_handle  
}
```

Related Topics

[pmatch::get_pattern](#)

[pmatch::rename_pattern](#)

pmatch::add_custom_marker_rect

Add a rectangular custom marker inside the pattern bounding box.

Usage

```
pmatch::add_custom_marker_rect -pattern pattern_handle -name marker_name  
-ur_coord '{x1 y1}' -ll_coord '{x2 y2}'
```

Arguments

- **-pattern *pattern_handle***
A required argument that specifies a valid pattern handle.
- **-name *marker_name***
A required argument that specifies the new custom marker name.
- **-ur_coord '{x1 y1}'**
A required argument that specifies the upper right coordinates of the custom rectangular marker in user units.
- **-ll_coord '{x2 y2}'**
A required argument that specifies the lower left coordinates of the custom rectangular marker in user units.

Return Values

Returns 1 if the command is successful, or -1 if the command fails.

Description

Adds a custom marker to a pattern. The marker geometry is defined by the upper right coordinates and lower left coordinates. The custom marker is always a rectangle.

If the marker extends outside of the pattern bounding box, the marker is clipped and a warning message is issued. The pattern bounding box encloses all layer geometries and extents, but does not include regions that lie outside of the pattern extent.

All patterns in the library must have the same number of markers. Therefore, if you use this command, you must apply it to all of the patterns in the library.

Examples

Add custom rectangular markers to patterns in a library.

```
foreach name $pattern_names_list {  
  set pat_handle [pmatch::get_pattern -name $name]  
  set pat_handle [lindex $pat_handle 0]  
  pmatch::add_custom_marker_rect -pattern $pat_handle -name custom_rect \  
    -ur_coord {0.6 5887.6} -ll_coord {0.007 5887.5}  
  pmatch::update_pattern -pattern $pat_handle  
}
```

pmatch::add_marker

Add a marker to a pattern. Marker types of bbox, bbox10, drawn, or matched can be specified.

Usage

```
pmatch::add_marker -pattern pattern_handle -layer layer_index -type marker_type  
-name marker_name
```

Arguments

- **-pattern *pattern_handle***
A required argument that specifies a valid pattern handle.
- **-layer *layer_index***
A required argument that specifies a layer index. Specify “-layer 0” for layer-independent marker types such as bbox and bbox10. Specify a pattern layer index for layer-dependent markers such as matched, drawn, bbox(*layer*), or bbox10(*layer*). The index for pattern layers starts at 1.
- **-type *marker_type***
A required argument that specifies a the marker type, which can be bbox, bbox10, drawn, or matched.
- **-name *marker_name***
A required argument that specifies the marker name.

Return Values

Returns 1 if the command is successful, or -1 if the command fails.

Examples

Add a marker of type bbox to all the patterns in a pattern list. The marker is given the name “newbox”.

```
foreach name $pattern_names_list {  
  set pat_handle [pmatch::get_pattern -name $name]  
  set pat_handle [lindex $pat_handle 0]  
  set newglobalbbox [pmatch::add_marker -pattern $pat_handle -layer 0 \  
    -type bbox -name newbox]  
}
```

Related Topics

[pmatch::get_markers](#)

[pmatch::add_custom_bbox_marker](#)

[pmatch::delete_marker](#)

pmatch::delete_marker

Delete a marker from a pattern.

Usage

pmatch::delete_marker -pattern *pattern_handle* -name *marker_name*

Arguments

- **-pattern** *pattern_handle*
A required argument that specifies a valid pattern handle.
- **-name** *marker_name*
A required argument that specifies the marker name.

Return Values

Returns 1 if the command is successful, or -1 if the command fails.

Examples

This example shows how to delete markers with the name Custom from the patterns in a library.

```
foreach name $pattern_names_list {  
    set pat_handle [pmatch::get_pattern -name $name]  
    set pat_handle [lindex $pat_handle 0]  
    set markers [pmatch::get_markers -pattern $pat_handle]  
    pmatch::delete_marker -pattern $pat_handle -name Custom  
}
```

Related Topics

[pmatch::add_marker](#)

[pmatch::add_custom_bbox_marker](#)

[pmatch::get_markers](#)

pmatch::get_markers

Get the list of markers in a pattern. The information returned for each marker includes the marker name, corresponding layer number, and marker type.

Usage

pmatch::get_markers -pattern *pattern_handle*

Arguments

- **-pattern** *pattern_handle*

A required argument specifying a pattern handle, which is the first element returned from [pmatch::get_pattern](#).

Return Values

Tcl list of sublists, where each sublist has the format:

{layer_id marker_type marker_name}

The elements are defined as follows:

- *layer_id* — The layer number for the marker, or 0 if a layer number is not applicable, such as for a bbox type marker.
- *marker_type* — One of {bbox, bbox10, drawn, matched, custom}. For information on marker types, see “[Marker Layer](#)” in the *Calibre Pattern Matching User’s Manual*.
- *marker_name* — The marker name.

Examples

Get the markers for a pattern.

```
% pmatch::load_pattern_lib -path example_lib.pmdb
Pattern Library DATABASE
% set pat_names [pmatch::get_pattern_names]
p1_tem p2_tem p3_bcm

% set patx [pmatch::get_pattern -name [lindex $pat_names 0]]
{PDL Pattern} {}
% set patx_handle [lindex $patx 0]
PDL Pattern

% pmatch::get_markers -pattern $patx_handle
{0 bbox10 Marker}
```

pmatch::rename_marker

Rename a specified marker in a pattern.

Usage

```
pmatch::rename_marker -pattern pattern_handle  
-original_name original_marker_name -new_name new_marker_name
```

Arguments

- **-pattern** *pattern_handle*
A required argument specifying the pattern handle.
- **-original_name** *original_marker_name*
A required argument that specifies the current name of the marker.
- **-new_name** *new_marker_name*
A required argument that specifies the new name for the marker. Marker names can contain letters, numbers, and underscores.

Return Values

Returns a value of -1 if the command fails, and 1 if the command is successful.

Examples

Rename the marker named “Marker” for each pattern.

```
set a_lib [pmatch::load_pattern_lib -path lib1.pmdb]  
set patterns [pmatch::get_pattern_names]  
  
foreach name $patterns {  
    set pat_x [pmatch::get_pattern -name $name]  
    set pat_handle [lindex $pat_x 0]  
    set return_val [pmatch::rename_marker -pattern $pat_handle \  
        -original_name Marker -new_name Marker_2]  
  
    if {$return_val == -1} {  
        puts "The marker was not renamed in pattern $name."  
    }else {  
        puts "The marker was successfully renamed in pattern $name."  
    }  
    pmatch::update_pattern -pattern $pat_handle  
}
```

Pattern Geometries

There commands process pattern geometry vertices and edges.

To retrieve pattern geometries, you must first load a pattern library using the [pmatch::create_pattern_lib](#) or [pmatch::load_pattern_lib](#) command.

For information on creating and using libraries using the Calibre Pattern Matching GUI, refer to the [Calibre Pattern Matching User's Manual](#).

pmatch::get_edges	55
pmatch::get_vertices	57

pmatch::get_edges

Get the list of edges for a specified pattern layer.

Usage

pmatch::get_edges -pattern *pattern_handle* -layer *layer_index*

Arguments

- **-pattern** *pattern_handle*
A required argument specifying the pattern handle.
- **-layer** *layer_index*
A required argument specifying the pattern layer index, which is an integer greater than or equal to 1.

Return Values

Tcl list of sublists, with the sublist for each edge having the following format:

```
{is_real_flag edge_id vertex1_id vertex2_id}
```

The values are defined as follows:

- *is_real_flag* — 0 or 1 to indicate if this edge is virtual or real (respectively)
- *edge_id* — The edge ID as in the pattern
- *vertex1_id* — ID of the first vertex on the edge
- *vertex2_id* — ID of the second vertex on the edge

Description

Gets the list of edges for the specified pattern layer. The edges are identified as real or virtual. Virtual edges are coincident with the pattern's extent. Real edges are completely enclosed in the pattern's extent. A real edge defines the placement of an edge for a pattern that is matched.

Examples

For one pattern, get the list of edges for layer 1.

```
% pmatch::load_pattern_lib -path example_lib.pmdb
Pattern Library DATABASE
% set pat_names [pmatch::get_pattern_names]
p1_tem p2_tem p3_bcm
```

```
% set patx [pmatch::get_pattern -name [lindex $pat_names 0]]
{PDL Pattern} {}
% set patx_handle [lindex $patx 0]
PDL Pattern

% pmatch::get_edges -pattern $patx_handle -layer 1
{0 1 1 2} {1 2 2 3} {1 3 3 4} {0 4 4 5} {1 5 5 6} {1 6 6 1}
```

For all patterns in a pattern library, get the list of edges on layer 1.

```
% pmatch::load_pattern_lib -path example_lib.pmdb
Pattern Library DATABASE
% set pat_names [pmatch::get_pattern_names]
p1_tem p2_tem p3_bcm

% foreach patternName $pat_names {
    set patx [pmatch::get_pattern -name [lindex $patternName 0]]
    set patx_handle [lindex $patx 0]
    set edges [pmatch::get_edges -pattern $patx_handle -layer 1]
    puts "$patternName: $edges"
}
p1_tem: {0 1 1 2} {1 2 2 3} {1 3 3 4} {0 4 4 5} {1 5 5 6} {1 6 6 1}
p2_tem: {1 1 1 2} {0 2 2 3} {1 3 3 4} {0 4 4 1} {1 5 5 6} {0 6 6 7}
        {1 7 7 8} {0 8 8 5}
p3_bcm: {1 1 1 2} {0 2 2 3} {1 3 3 4} {0 4 4 1} {1 5 5 6}
        {0 6 6 7} {1 7 7 8} {0 8 8 5}
```

Related Topics

[pmatch::get_vertices](#)

pmatch::get_vertices

Get the list of vertex IDs and coordinates for a specified pattern layer

Usage

pmatch::get_vertices -pattern *pattern_handle* -layer *layer_index*

Arguments

- **-pattern *pattern_handle***
A required argument specifying the pattern handle.
- **-layer *layer_index***
A required argument specifying the pattern layer number, which is an integer starting with 1.

Return Values

Tcl list of sublists, each having the format:

{*vertex_id coordinate_x coordinate_y*}

The values *coordinate_x* and *coordinate_y* are in user units.

Examples

For one pattern, get the vertex ID and coordinates for each vertex on layer 1.

```
% pmatch::load_pattern_lib -path example_lib.pmdb
Pattern Library DATABASE
% set pat_names [pmatch::get_pattern_names]
p1_tem p2_tem p3_bcm

% set patx [pmatch::get_pattern -name [lindex $pat_names 0]]
{PDL Pattern} {}
% set patx_handle [lindex $patx 0]
PDL Pattern

% pmatch::get_vertices -pattern $patx_handle -layer 1
{1 -121000 11200} {2 -119000 11200} {3 -119000 15000} {4 -114100 15000}
{5 -114100 17000} {6 -121000 17000}
```

For all patterns in a pattern library, get the vertex ID and coordinates for each vertex on layer 1.

```
% pmatch::load_pattern_lib -path example_lib.pmdb
Pattern Library DATABASE
% set pat_names [pmatch::get_pattern_names]
p1_tem p2_tem p3_bcm
```

```
% foreach patternName $pat_names {  
  set patx [pmatch::get_pattern -name [lindex $patternName 0]]  
  set patx_handle [lindex $patx 0]  
  set verts [pmatch::get_vertices -pattern $patx_handle -layer 1]  
  puts "pattern $patternName: $verts"  
}  
pattern p1_tem: {1 -121000 11200} {2 -119000 11200} {3 -119000 15000}  
             {4 -114100 15000} {5 -114100 17000} {6 -121000 17000}  
pattern p2_tem: {1 -58200 0} {2 -44700 0} {3 -44700 2000} {4 -58200 2000}  
             {5 -58200 15000} {6 -44700 15000} {7 -44700 17000} {8 -58200 17000}  
pattern p3_bcm: {1 -76800 -73000} {2 -66200 -73000} {3 -66200 -71000}  
             {4 -76800 -71000} {5 -76800 -59000} {6 -66200 -59000} {7 -66200 -57000}  
             {8 -76800 -57000}
```

Related Topics

[pmatch::get_edges](#)

Pattern Extents

The extent layer defines the borders of the pattern and isolates the pattern from surrounding geometries. Extents can be defined for the pattern as a whole or for individual layers.

The polygons that fall within the pattern's extent are evaluated for a match while polygons outside the extent are not evaluated. By default, the pattern extent is rectangular. An extent defines the limits of the pattern.

Note



Extents are Manhattan geometries, containing only 90 degree angles.

See “[Extent Layer](#)” in the *Calibre Pattern Matching User's Manual* for more information.

pmatch::get_extent_type	60
pmatch::get_extent_bbox	61
pmatch::size_up_extent	62
pmatch::snap_extent	63

pmatch::get_extent_type

Get the extent type for the pattern or for a pattern layer.

Usage

pmatch::get_extent_type -pattern *pattern_handle* [-layer *layer_index*]

Arguments

- **-pattern** *pattern_handle*

A required argument that specifies a pattern handle.

- -layer *layer_index*

An optional argument that specifies the layer index to get the extent type for. The index for pattern layers starts at 1.

If the -layer option is not used, the default layer index is 0, which gets the type of the pattern extent.

Return Values

Returns one of the following strings:

- auto — The extent is the bounding box of the layer geometries.
- custom — The extent is user-defined.
- none — The layer inherits the pattern extent.

Examples

Outputs the pattern extent type for all patterns in a library.

```
foreach name $patterns {
  set pat_x [pmatch::get_pattern -name $name]
  set pat_handle [lindex $pat_x 0]
  set pat_extent_type [pmatch::get_extent_type -pattern $pat_handle]
  if {$pat_extent_type == -1} {
    puts "Did not get extent type for pattern $name."
  } else {
    puts "The extent type for pattern $name is: $pat_extent_type."
  }
}
```

pmatch::get_extent_bbox

Get the coordinates of the bounding box of the extent for a pattern or specified pattern layer.

Usage

pmatch::get_extent_bbox -pattern *pattern_handle* [-layer *layer_index*]

Arguments

- **-pattern** *pattern_handle*

A required argument that specifies a pattern handle.

- **-layer** *layer_index*

An optional argument that specifies a layer index. The index for pattern layers starts at 1. The index for the pattern extent is 0.

If the -layer option is not used, the default layer index is 0, which gets the bounding box of the pattern extent.

Return Values

Outputs a Tcl list containing the coordinates of the extent bounding box in user units. Two coordinates pairs are output, the lower left and upper right vertices of the bounding box:

`{x_lowerLeft y_lowerLeft x_upperRight y_upperRight}`

Description

Gets the coordinates of the extent bounding box for a pattern or specified pattern layer. If the specified layer has no explicit extent, it inherits the values from the pattern extent.

Examples

Get the extent bounding box coordinates for all patterns and print the extent coordinates for each pattern.

```
foreach name $patterns {
    set pat_x [pmatch::get_pattern -name $name]
    set pat_handle [lindex $pat_x 0]
    set pat_extent [pmatch::get_extent_bbox -pattern $pat_handle]
    if {$pat_extent == -1} {
        puts "Did not successfully get extent bounding box coordinates."
    } else {
        puts "The extent bounding box coordinates are: $pat_extent."
    }
}
```

pmatch::size_up_extent

Sizes up the extent of the specified layer.

Usage

pmatch::size_up_extent -pattern *pattern_handle* -layer *layer_index* -dbu_value *value*

Arguments

- **-pattern *pattern_handle***
A required argument that specifies a pattern handle.
- **-layer *layer_index***
A required argument that specifies which layer extent to size up. The index for pattern layers starts at 1. If *layer_index* is 0, then the pattern extent is sized up.
- **-dbu_value *value***
A required argument that specifies a length in dbu by which to size up the extent.

Return Values

Returns a 1 if successful, or a -1 if it fails.

Description

Sizes up the extent of a specified layer by the specified value. You can specify auto or custom extents. If the specified layer does not have a defined extent, a new extent is not added. This command applies to exact patterns; that is, patterns that do not have constraints or cglobal defined.

Examples

In this example, the pattern extent is sized up by 5 dbu.

```
set pat_handle [pmatch::get_pattern -name primary]
set pat_handle [lindex $pat_handle 0]
pmatch::size_up_extent -pattern $pat_handle -layer 0 -dbu_value 5
pmatch::update_pattern -pattern $pat_handle
```

pmatch::snap_extent

Change the custom extent of a specified layer into an auto extent.

Usage

pmatch::snap_extent -pattern *pattern_handle* -layer *layer_index*

Arguments

- **-pattern *pattern_handle***
A required argument specifying a pattern handle.
- **-layer *layer_index***
A required argument that specifies a layer index. The index for pattern layers starts at 1. If *layer_index* is 0, then the pattern extent is snapped.

Return Values

Returns a 1 if the command is successful, or -1 if the command fails.

Description

Changes the custom extent of a specified layer into an auto extent, if the layer extent exists.

If the specified layer has a single rectangular polygon, the new extent is the polygon extent sized up by 1 dbu. This is done so that the resulting pattern is valid.

If the specified layer does not have an explicitly defined custom extent, then no new extent is added. This only applies to exact patterns (patterns with no constraints or cglobal specified).

Examples

Snap the extent of a pattern for all patterns in a library.

```
foreach name $pattern_names_list {  
  set pat_handle [pmatch::get_pattern -name $name]  
  set pat_handle [lindex $pat_handle 0]  
  pmatch::snap_extent -pattern $pat_handle -layer 0  
  pmatch::update_pattern -pattern $pat_handle  
}
```

Pattern Properties, Keys, and Constraints

These commands retrieve pattern properties, keys, and constraints.

Pattern properties give you the ability to operate on output markers based on property names and values using other commands, or to output the property directly to a Calibre DFM RDB.

Keys enable you to select specific patterns that are used in the pattern matching run. If no keys are defined, all patterns are matched. A pattern may specify zero or more keys.

Constraints enable you to specify the distance an edge can shift and still be considered a match. TEM patterns support both single-edge constraints and edge to edge constraints. Single-edge constraints enable specific edges to vary from the original location. Edge to edge constraints enable any two parallel edges to vary from the original pattern, and the two edges can be on different layers.

For information on creating and managing pattern libraries using the Calibre Pattern Matching GUI, refer to the [Calibre Pattern Matching User's Manual](#).

pmatch::add_int_attr	66
pmatch::add_keys	68
pmatch::add_pattern_orient	69
pmatch::add_properties	70
pmatch::add_text_attribute	72
pmatch::delete_constraint_label	74
pmatch::delete_keys	76
pmatch::delete_properties	78
pmatch::get_cglobal	80
pmatch::get_constraints	81
pmatch::get_constraint_label	83
pmatch::get_int_attr	85
pmatch::get_keys	86
pmatch::get_library_attribute	88
pmatch::get_pattern_orient	89
pmatch::get_properties	90
pmatch::get_property_value	92
pmatch::get_text_attribute	93
pmatch::set_cglobal	94
pmatch::set_constraint_label	96
pmatch::set_library_attribute	98

pmatch::set_pattern_orient	99
---	-----------

pmatch::add_int_attr

Adds an integer attribute to the specified pattern.

Usage

```
pmatch::add_int_attr [-lib library_handle] -pattern_name pattern_name  
-attr_name attr_name -attr_val attr_val
```

Arguments

- **-lib *library_handle***
An optional argument set that specifies a pattern library handle returned by [pmatch::create_pattern_lib](#) or [pmatch::load_pattern_lib](#). The active library is used if this argument set is not specified.
- **-pattern_name *pattern_name***
A required argument set that specifies the name of the pattern.
- **-attr_name *attr_name***
A required argument set that specifies the name of the integer attribute to add to the pattern.
- **-attr_val *attr_val***
A required argument set that specifies the integer value for the attribute.

Return Values

Returns a 1 if the command is successful, or -1 if the command fails.

Description

Adds an integer attribute to the specified pattern.

This command updates the PMDB file directly, therefore a `pmatch::update_pattern` command is not necessary in order to save the changes. For proper functioning when making other changes to the same pattern, the command should be executed *before* a `pmatch::get_pattern` command or *after* a `pmatch::update_pattern` command.

Caution



Do not execute `pmatch::add_int_attr` between a `pmatch::get_pattern` command and a `pmatch::update_pattern` command—the added integer attribute will not be saved.

Examples

Add an integer attribute to all patterns within an active library, and output whether the command was successful.

```
set pat_lib [pmatch::load_pattern_lib -path myLib.pmdb]
set pattern_list [pmatch::get_pattern_names -lib $pat_lib]

foreach name $pattern_list {
    set ret_val [pmatch::add_int_attr -pattern_name $name \
        -attr_name myAttr -attr_val 123]
    if {$ret_val == -1} {
        puts "The integer attribute was not successfully added."
    } else {
        puts "The integer attribute was successfully added."
    }
    # make other pattern changes
    set pat [pmatch::get_pattern -name $name]
    set pat_handle [lindex $pat 0]
    pmatch::add_keys -pattern $pat_handle -keys_list {k1 k2}
    ## do not use pmatch::add_int_attr after get_pattern,
    ## change will not be saved
    pmatch::update_pattern -pattern $pat_handle
}
```

pmatch::add_keys

Add a specified list of keys to a pattern in the active library.

Usage

pmatch::add_keys -pattern *pattern_handle* -keys_list *keys_list*

Arguments

- **-pattern** *pattern_handle*
A required argument specifying the pattern handle.
- **-keys_list** *keys_list*
A required argument specifying a Tcl list of key values.

Return Values

Returns 1 if the command is successful, or -1 if the command fails.

Examples

Add keys to each pattern in a library.

```
% pmatch::load_pattern_lib -path example_lib.pmdb
Pattern Library DATABASE
% set pat_names [pmatch::get_pattern_names]
p1_tem p2_tem p3_bcm

% set i 0
0
% foreach patternName $pat_names {
    set patx [pmatch::get_pattern -name [lindex $patternName 0]]
    set patx_handle [lindex $patx 0]
    set key_names [list mykey_$i]
    pmatch::add_keys -pattern $patx_handle -keys_list $key_names
    set keys [pmatch::get_keys -pattern $patx_handle]
    puts "pattern $patternName: $keys"
    incr i
}
pattern p1_tem: mykey_0
pattern p2_tem: mykey_1
pattern p3_bcm: mykey_2

% pmatch::update_pattern -pattern $patx_handle
1
```

Related Topics

[pmatch::get_keys](#)

[pmatch::delete_keys](#)

pmatch::add_pattern_orient

Specifies to match the pattern in the given orientation. All other orientation values for the pattern are unchanged.

Usage

pmatch::add_pattern_orient -pattern *pattern_handle* -orient *orientation*

Arguments

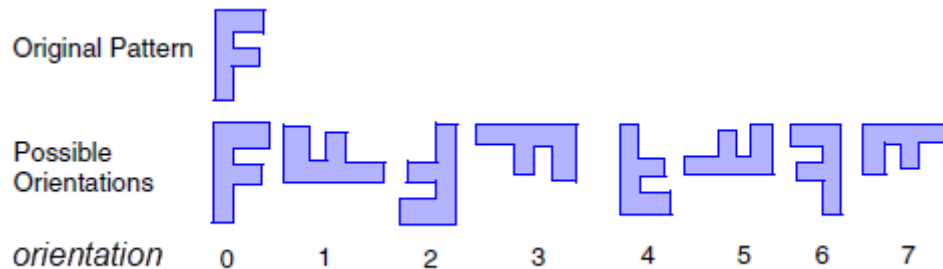
- **-pattern *pattern_handle***

A required argument specifying the pattern handle.

- **-orient *orientation***

A required argument that specifies the orientation, where *orientation* is an integer value in the range $0 \leq \text{orientation} \leq 7$. The possible orientations are shown in the following figure:

Figure 3-1. Orientations for add_pattern_orient



Return Values

Returns 1 for success, -1 for failure.

Description

Specifies that the pattern should match in the given orientation. The specified orientation is set to 1 (match) and all other orientation values for the pattern are unchanged. The orientations correspond to what is seen in the Pattern Matching GUI in the Orientation section of the **Attributes** tab.

Use [pmatch::set_pattern_orient](#) to set a given orientation to 1 (match) and set all other orientations to 0 (not matched).

Examples

See the example with [pmatch::set_pattern_orient](#).

Related Topics

[Pattern Orientation \[Calibre Pattern Matching User's Manual\]](#)

pmatch::add_properties

Adds a specified list of properties to a pattern in the active library.

Usage

pmatch::add_properties -pattern *pattern_handle* -prop_list *prop_list*

Arguments

- **-pattern** *pattern_handle*

A required argument specifying the pattern handle.

- **-prop_list** *prop_list*

A required argument specifying a Tcl list of sublists, each sublist having the format:

`{prop_name prop_value [prop_type]}`

The optional *prop_type* entry can be either “float” or “integer”, where “float” is the default if not specified.

Return Values

Returns 1 if the command is successful, or -1 if the command fails.

Examples

Add properties to a pattern.

```
% pmatch::load_pattern_lib -path example_lib.pmdb
Pattern Library DATABASE
% set pat_names [pmatch::get_pattern_names]
p1_tem p2_tem p3_bcm

% set patx [pmatch::get_pattern -name [lindex $pat_names 0]]
{PDL Pattern} {}
% set patx_handle [lindex $patx 0]
PDL Pattern

% pmatch::get_properties -pattern $patx_handle
{cal 1.0 float}

# Add a property to a pattern and update the pattern in the library
% pmatch::add_properties -pattern $patx_handle -prop_list {{prop_x 1}}
1
% pmatch::update_pattern -pattern $patx_handle
1
```

```
% pmatch::get_properties -pattern $patx_handle
{cal 1.0 float} {prop_x 1.0 float}

% pmatch::add_properties -pattern $patx_handle -prop_list {{p1 1} {p2 10}}
1
% pmatch::update_pattern -pattern $patx_handle
1

% pmatch::get_properties -pattern $patx_handle
{cal 1.0 float} {p1 1.0 float} {p2 10.0 float} {prop_x 1.0 float}

% foreach patternName $pat_names {
  set patx [pmatch::get_pattern -name $patternName]
  set patx_handle [lindex $patx 0]
  puts [pmatch::add_properties -pattern $patx_handle -prop_list {{p3 1}}]
}
1
1
1
1
```

Related Topics

[pmatch::delete_properties](#)

[pmatch::get_properties](#)

[pmatch::get_property_value](#)

pmatch::add_text_attribute

Adds a text attribute to the specified pattern.

Usage

```
pmatch::add_text_attribute [-lib library_handle] -pattern_name pattern_name  
-attr_name attr_name -attr_val attr_val
```

Arguments

- **-lib *library_handle***
An optional argument set that specifies the pattern library handle returned by [pmatch::create_pattern_lib](#) or [pmatch::load_pattern_lib](#).
- **-pattern_name *pattern_name***
A required argument set that specifies the name of the pattern.
- **-attr_name *attr_name***
A required argument that specifies the name of the attribute.
- **-attr_val *attr_val***
A required argument set that specifies the value of the text attribute. If *attr_val* includes spaces, it should be enclosed in quotes.

Return Values

Returns a value of 1 on success, and a value of -1 on failure.

Description

Adds a text attribute to the specified pattern.

This command updates the PMDB file directly, therefore a `pmatch::update_pattern` command is not necessary in order to save the changes. For proper functioning when making other changes to the same pattern, the command should be executed *before* a `pmatch::get_pattern` command or *after* a `pmatch::update_pattern` command.

Caution



Do not execute `pmatch::add_text_attribute` between a `pmatch::get_pattern` command and a `pmatch::update_pattern` command—the added text attribute will not be saved.

Examples

Add a text attribute to the specified pattern.


```
set pat_lib [pmatch::load_pattern_lib -path myLib.pmdb]
set pattern_list [pmatch::get_pattern_names -lib $pat_lib]

foreach name $pattern_list {
    set ret_val [pmatch::add_text_attribute -pattern_name $name \
        -attr_name pat_attr -attr_val worst_patterns]
    if {$ret_val == -1} {
        puts "The text attribute was not added to the pattern."
    } else {
        puts "The text attribute was added to the pattern."
    }
    # make other pattern changes
    set pat [pmatch::get_pattern -name $name]
    set pat_handle [lindex $pat 0]
    pmatch::add_keys -pattern $pat_handle -keys_list {k1 k2}
    ## do not use pmatch::add_text_attribute after get_pattern,
    ## change will not be saved
    pmatch::update_pattern -pattern $pat_handle
}
```

pmatch::delete_constraint_label

Deletes the label for a specified constraint, or deletes all instances of a specified label in the pattern.

Usage

pmatch::delete_constraint_label -pattern *pattern_handle*
 {-id *constraint_id* | -label *label*}

Arguments

- **-pattern *pattern_handle***
A required argument set specifying the pattern handle.
- **-id *constraint_id* | -label *label***
A required argument choice that specifies the constraint label to remove.
 - id *constraint_id*** — Specifies to remove the constraint label for the constraint specified by *constraint_id*. The constraint ID is the second list element in the return value from [pmatch::get_constraints](#).
 - label *label*** — Specifies to remove all instances of the specified constraint label.

Return Values

Returns 1 if the command is successful, or -1 if the command fails.

Examples

Get all constraints, then delete the label for the first constraint, using the following Tcl script *constraints.tcl*:

```
pmatch::load_pattern_lib -path one_patt.pmdb
set pat_names [pmatch::get_pattern_names]
set patx [pmatch::get_pattern -name [lindex $pat_names 0]]
set patx_handle [lindex $patx 0]
puts "constraints in first pattern"
set constr [pmatch::get_constraints -pattern $patx_handle -labels]
puts "$constr"

puts "\ndeleting the label for constraint 1"
set ret_val [pmatch::delete_constraint_label -pattern $patx_handle -id 1]
puts "returned: $ret_val"
puts "label for constraint 1:"
set label [pmatch::get_constraint_label -pattern $patx_handle -id 1]
puts "$label"
```

Run the script:

pdl_lib_mgr prompt constraints.tcl

The following output is produced. The pattern has two single edge constraints. The first constraint has a label “label1”, while the second constraint does not have a label:

```
constraints in first pattern
{SingleEdge 1 label1 e5 {>= 2.251 <= 2.501}} {SingleEdge 2 e1 {>= 0.801 <=
1.201}}
```

```
deleting the label for constraint 1
returned: 1
label for constraint 1:
```

Related Topics

[pmatch::get_constraints](#)

[pmatch::set_constraint_label](#)

[pmatch::get_constraint_label](#)

pmatch::delete_keys

Deletes a specified list of keys from a pattern in the active library.

Usage

pmatch::delete_keys -pattern *pattern_handle* -keys_list *keys_list*

Arguments

- **-pattern** *pattern_handle*
A required argument that specifies the pattern handle.
- **-keys_list** *keys_list*
A required argument that specifies a Tcl list of key values.

Return Values

Returns 1 if the command is successful, or -1 if the command fails.

Examples

Delete a key from a pattern.

```
% pmatch::load_pattern_lib -path example_lib.pmdb
Pattern Library DATABASE
% set pat_names [pmatch::get_pattern_names]
p1_tem p2_tem p3_bcm

% set patx [pmatch::get_pattern -name [lindex $pat_names 0]]
{PDL Pattern} {}
% set patx_handle [lindex $patx 0]
PDL Pattern

% pmatch::get_keys -pattern $patx_handle
mykey_0

pmatch::delete_keys -pattern $patx_handle -keys_list {junk}
Trying to remove nonexistent key: junk
-1

pmatch::delete_keys -pattern $patx_handle -keys_list {{mykey_0}}
Key 'mykey_0' deleted from pattern 'p1_tem'.
1

# Update pattern in library to reflect changes
% pmatch::update_pattern -pattern $patx_handle
1
```

Related Topics

[pmatch::get_keys](#)

[pmatch::add_keys](#)

pmatch::delete_properties

Deletes specified properties from a pattern in the active library.

Usage

pmatch::delete_properties -pattern *pattern_handle* -prop_names_list *prop_list*

Arguments

- **-pattern** *pattern_handle*
A required argument specifying the pattern handle.
- **-prop_names_list** *prop_list*
A required argument specifying a Tcl list of sublists, each having the format:
{*prop_name*}

Return Values

Returns 1 if the call is successful, or -1 if the call fails.

Examples

Delete properties from a pattern.

```
% pmatch::load_pattern_lib -path example_lib.pmdb
Pattern Library DATABASE
% set pat_names [pmatch::get_pattern_names]
p1_tem p2_tem p3_bcm

% set patx [pmatch::get_pattern -name [lindex $pat_names 0]]
{PDL Pattern} {}
% set patx_handle [lindex $patx 0]
PDL Pattern

% pmatch::get_properties -pattern $patx_handle
{cal 1.0 float} {p1 1.0 float} {p2 10.0 float} {prop_x 1.0 float}

pmatch::delete_properties -pattern $patx_handle -prop_names_list {{junk}}
Trying to remove nonexistent property: junk
-1

pmatch::delete_properties -pattern $patx_handle -prop_names_list {{p1}}
Property 'p1' deleted from pattern 'p1_exact'.
1

# Update pattern in library to reflect changes
% pmatch::update_pattern -pattern $patx_handle
1
% pmatch::get_properties -pattern $patx_handle
{cal 1.0 float} {p2 10.0 float} {prop_x 1.0 float}
```

Related Topics

[pmatch::add_properties](#)

[pmatch::get_properties](#)

[pmatch::get_property_value](#)

pmatch::get_cglobal

Get the value of the cglobal constraint for a pattern.

Usage

pmatch::get_cglobal -pattern *pattern_handle*

Arguments

- **-pattern *pattern_handle***
A required argument specifying the pattern handle.

Return Values

If the call is successful, it returns the cglobal value in user units. The command returns -1 if the call fails.

Examples

Get the cglobal value for a pattern.

```
% pmatch::load_pattern_lib -path example_lib.pmdb
Pattern Library DATABASE
% set pat_names [pmatch::get_pattern_names]
p1_tem p2_tem p3_bcm

% set patx [pmatch::get_pattern -name [lindex $pat_names 0]]
{PDL Pattern} {}
% set patx_handle [lindex $patx 0]
PDL Pattern

% pmatch::get_cglobal -pattern $patx_handle
0.0
```

Related Topics

[pmatch::set_cglobal](#)

[Global Constraints \[Calibre Pattern Matching User's Manual\]](#)

pmatch::get_constraints

Gets a list of the constraints in a pattern.

Usage

pmatch::get_constraints -pattern *pattern_handle* [-labels]

Arguments

- **-pattern *pattern_handle***
A required argument specifying the pattern handle.
- **-labels**
An optional argument that includes constraint labels, if present, in the output.

Return Values

A Tcl list of sublists with the following formats, depending on the constraint type. The label entry is output only if -labels is included and a constraint label exists. Marker constraints are not reported.

- For cglobal constraints: {cglobal *value*} {cglobal_per_layer {*layer value*}...}
The *value* is multiplied by the pattern precision.
- For single edge constraint: {SingleEdge *cID label eID Expr*}
- Edge to edge constraint: {Measure *cID label eID eID Expr*}
- Non-directional edge to edge constraint: {Range *cID label eID eID Expr*}
- Relational constraint: {Relational *cID label e2eID e2eID opID*}

where:

- *cID* — The constraint ID.
- *eID* — An edge ID. Edge ID's are assigned internally.
- *Expr* — A mathematical expression describing the constraint.
- *e2eID* — An edge to edge constraint ID. Used in relational constraints.
- *opID* — An integer indicating the operator for a relational constraint: 0: ==, 1: !=, 2: <, 3: <=, 4: >=, 5: >

Examples

Get the constraints for a pattern, using the following Tcl script *constraints.tcl*:

```
pmatch::load_pattern_lib -path one_patt.pmdb
set pat_names [pmatch::get_pattern_names]
set patx [pmatch::get_pattern -name [lindex $pat_names 0]]
set patx_handle [lindex $patx 0]
puts "constraints in first pattern"
set constr [pmatch::get_constraints -pattern $patx_handle -labels]
puts "$constr"
```

Run the script:

pdl_lib_mgr prompt constraints.tcl

The following output is produced. The pattern has two single edge constraints. The first constraint has a label “label1”, while the second constraint does not have a label:

```
constraints in first pattern
{SingleEdge 1 label1 e5 {>= 2.251 <= 2.501}} {SingleEdge 2 e1 {>= 0.801 <= 1.201}}
```

Related Topics

[pmatch::set_constraint_label](#)

[pmatch::get_constraint_label](#)

[pmatch::delete_constraint_label](#)

`pmatch::get_constraint_label`

Gets the label for a constraint.

Usage

`pmatch::get_constraint_label` -pattern *pattern_handle* -id *constraint_id*

Arguments

- **-pattern *pattern_handle***
A required argument set specifying the pattern handle.
- **-id *constraint_id***
A required argument set specifying the constraint ID. The constraint ID is the second list element in the return value from [pmatch::get_constraints](#).

Return Values

Returns the label, or an empty string if the label does not exist.

Examples

Get all constraints, then print the label for the first constraint, using the following Tcl script *constraints.tcl*:

```
pmatch::load_pattern_lib -path one_patt.pmdb
set pat_names [pmatch::get_pattern_names]
set patx [pmatch::get_pattern -name [lindex $pat_names 0]]
set patx_handle [lindex $patx 0]
puts "constraints in first pattern"
set constr [pmatch::get_constraints -pattern $patx_handle -labels]
puts "$constr"
puts "\ngetting label for constraint 1"
set label [pmatch::get_constraint_label -pattern $patx_handle -id 1]
puts "$label"
```

Run the script:

`pdl_lib_mgr prompt constraints.tcl`

The following output is produced. The pattern has two single edge constraints. The first constraint has a label “label1”, while the second constraint does not have a label:

```
constraints in first pattern
{SingleEdge 1 label1 e5 {>= 2.251 <= 2.501}} {SingleEdge 2 e1 {>= 0.801 <=
1.201}}

getting label for constraint 1
label1
```

Related Topics

[pmatch::get_constraints](#)

[pmatch::set_constraint_label](#)

[pmatch::delete_constraint_label](#)

pmatch::get_int_attr

Gets the value of an integer attribute for a pattern.

Usage

```
pmatch::get_int_attr [-lib library_handle] -pattern_name pattern_name  
-attr_name attr_name
```

Arguments

- **-lib *library_handle***
An optional argument that specifies the pattern library handle returned by [pmatch::create_pattern_lib](#) or [pmatch::load_pattern_lib](#).
- **-pattern_name *pattern_name***
A required argument specifying the name of the pattern.
- **-attr_name *attr_name***
A required argument specifying the name of the attribute.

Return Values

Outputs the attribute value for the pattern, or -1 in case of error.

Examples

Gets the integer attribute of all patterns within an active library, and outputs a message stating whether or not the command was successful.

```
foreach name $patterns {  
  set pat_x [pmatch::get_pattern -name $name]  
  set pat_handle [lindex $pat_x 0]  
  set pat_attr [pmatch::add_int_attr -pattern_name $name -attr_name \  
    pat_attr -attr_val 123]  
  set pat_attr_out [pmatch::get_int_attr -pattern_name $name -attr_name \  
    pat_attr]  
  if {$pat_attr_out == -1} {  
    puts "Get integer attribute was not successful."  
  } else {  
    puts "Get integer attribute was successful."  
  }  
}
```

pmatch::get_keys

Gets the list of keys for a pattern.

Usage

pmatch::get_keys {**-pattern** *pattern_handle* | **-pattern_name** *name* [-lib *library_handle*]}

Arguments

- **-pattern** *pattern_handle* | **-pattern_name** *name* [-lib *library_handle*]

A required argument choice that specifies which pattern to get the keys from.

- **-pattern** *pattern_handle*

Specifies a pattern handle, which is the first element returned from [pmatch::get_pattern](#).

- **-pattern_name** *name* [-lib *library_handle*]

Specifies a pattern name.

The optional -lib argument specifies a pattern library handle returned by [pmatch::load_pattern_lib](#) or [pmatch::create_pattern_lib](#). If a pattern library handle is not specified the active library is used.

Return Values

Tcl list of key values or -1 on failure.

Examples

Example 1

Output the keys for each pattern in a library using the pattern name as input. The active library is used by default.

```
% pmatch::load_pattern_lib -path example_lib.pmdb
Pattern Library DATABASE
% set pat_names [pmatch::get_pattern_names]
p1_tem p2_tem p3_bcm

% foreach patternName $pat_names {
    set keys [pmatch::get_keys -pattern_name $patternName]
    puts "pattern $patternName: $keys"
}
pattern p1_tem: c1 c2
pattern p2_tem:
pattern p3_bcm: mykey
```

Example 2

Output the keys for each pattern in a library using the pattern handle as input.

```
% pmatch::load_pattern_lib -path example_lib.pmdb
Pattern Library DATABASE
% set pat_names [pmatch::get_pattern_names]
p1_tem p2_tem p3_bcm

% foreach patternName $pat_names {
    set patx [pmatch::get_pattern -name [lindex $patternName 0]]
    set patx_handle [lindex $patx 0]
    set keys [pmatch::get_keys -pattern $patx_handle]
    puts "pattern $patternName: $keys"
}
pattern p1_tem: c1 c2
pattern p2_tem:
pattern p3_bcm: mykey
```

Related Topics

[pmatch::add_keys](#)

[pmatch::delete_keys](#)

pmatch::get_library_attribute

Gets the value of a global attribute with the specified name.

Usage

pmatch::get_library_attribute [-lib *library_handle*] -name *attribute_name*

Arguments

- -lib *library_handle*
An optional argument that specifies the pattern library handle returned by [pmatch::create_pattern_lib](#) or [pmatch::load_pattern_lib](#). The active library is used if this argument is not specified.
- -name *attribute_name*
A required argument specifying the name of the attribute.

Return Values

Returns the attribute value on success, or an empty string on failure.

Examples

Get the global attribute for the library.

```
set a_lib [pmatch::load_pattern_lib -path pat1.pmdb]
set patterns [pmatch::get_pattern_names]
set num_patterns [llength $patterns]
set lib_attr [pmatch::set_library_attribute -name errors -value 2]
set lib_attr_out [pmatch::get_library_attribute -name errors]
puts "library attribute $lib_attr_out"
```


`pmatch::get_pattern_orient`

Gets the orientation settings for a pattern.

Usage

`pmatch::get_pattern_orient` -pattern *pattern_handle*

Arguments

- **`-pattern pattern_handle`**
A required argument specifying the pattern handle.

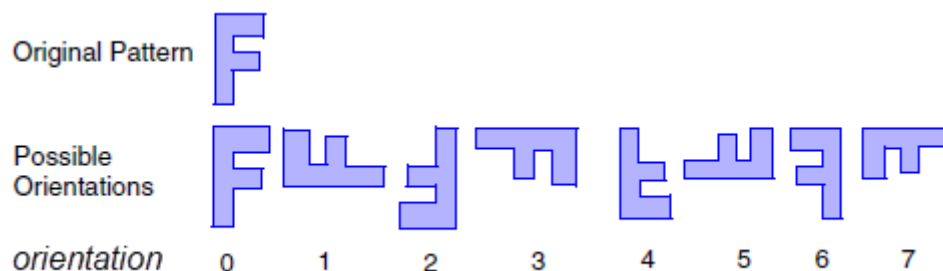
Return Values

A string with the orientation values, or -1 if the command fails.

Description

Returns the orientation settings for a pattern. The orientations correspond to what is seen in the Pattern Matching GUI in the Orientation section of the **Attributes** tab.

The returned value is a string containing the setting (0 or 1) for the eight possible pattern orientations.



For example, the following returned string has only orientation 0 set, so the pattern only matches in the original orientation.

```
10000000
```

Use [`pmatch::set_pattern_orient`](#) and [`pmatch::add_pattern_orient`](#) to set pattern orientations.

Examples

```
set orient_val [pmatch::get_pattern_orient -pattern $myPat_handle]
puts "Orientation setting: $orient_val "
```

See [`pmatch::set_pattern_orient`](#) for a complete example.

Related Topics

[Pattern Orientation \[Calibre Pattern Matching User's Manual\]](#)

pmatch::get_properties

Gets the properties in a pattern. The pattern may be specified with a pattern handle or a pattern name.

Usage

```
pmatch::get_properties  
  {-pattern pattern_handle | {-pattern_name name [-lib library_handle]}}  
  [-no_types]
```

Arguments

- **-pattern** *pattern_handle* | {**-pattern_name** *name* [-lib *library_handle*]}
- A required argument choice that specifies a pattern.
 - **-pattern** *pattern_handle*
Specifies a pattern handle, which is the first element returned from [pmatch::get_pattern](#).
 - **-pattern_name** *name* [-lib *library_handle*]
Specifies a pattern name.
The optional -lib argument specifies a pattern library handle returned by [pmatch::load_pattern_lib](#) or [pmatch::create_pattern_lib](#). If a pattern library handle is not specified the active library is used.
- **-no_types**
An optional argument that specifies to not include the property type in the output.

Return Values

Tcl list of sublists, each having the format: {*prop_name prop_value prop_type*}

prop_type can be “float” or “integer”.

If the -no_types option is used, the output format is: {*prop_name prop_value*}

Returns -1 on failure.

Examples

Get the properties for the first pattern in the active library.

```
% pmatch::load_pattern_lib -path example_lib.pmdb
Pattern Library DATABASE
% set pat_names [pmatch::get_pattern_names -lib $x_lib]
p1_tem p2_tem p3_bcm

% set patx [pmatch::get_pattern -name [lindex $pat_names 0]]
{PDL Pattern} {}
% set patx_handle [lindex $patx 0]
PDL Pattern

% pmatch::get_properties -pattern $patx_handle
{cal 1.0 float}

% pmatch::get_properties -pattern_name [lindex $pat_names 0] -no_types
{cal 1.0}
```

Related Topics

[pmatch::add_properties](#)

[pmatch::delete_properties](#)

[pmatch::get_property_value](#)

`pmatch::get_property_value`

Gets the value of the specified property in a pattern.

Usage

```
pmatch::get_property_value -pattern_name pattern_name  
-property_name property_name
```

Arguments

- **`-pattern_name`** *pattern_name*
A required argument that specifies the pattern name.
- **`-property_name`** *property_name*
A required argument that specifies the name of the property.

Return Values

Returns the property value if the command is successful, or NULL if the command fails.

Description

Gets the value of the specified property in a pattern. Failure occurs when the named pattern is not found in the pattern library or the property name is not found in the pattern.

Examples

Get property values from a set of assigned pattern properties.

```
foreach name $pattern_name_list {  
  set pat_handle [pmatch::get_pattern -name $name]  
  set pat_handle [lindex $pat_handle 0]  
  set first_prop [get_property_value -pattern_name $name \  
    -prop_name prop1]  
  set second_prop [get_property_value -pattern_name $name \  
    -prop_name prop2]  
  set third_prop [get_property_value -pattern_name $name \  
    -prop_name prop3]  
}
```

Related Topics

[pmatch::add_properties](#)

[pmatch::delete_properties](#)

[pmatch::get_properties](#)

pmatch::get_text_attribute

Gets the value of the specified text attribute for a pattern.

Usage

```
pmatch::get_text_attribute [-lib library_handle] -pattern_name pattern_name  
-attr_name attr_name
```

Arguments

- **-lib *library_handle***
An optional argument that specifies the pattern library handle returned by [pmatch::create_pattern_lib](#) or [pmatch::load_pattern_lib](#). The active library is used if this argument is not specified.
- **-pattern_name *pattern_name***
A required argument specifying the name of the pattern.
- **-attr_name *attr_name***
A required argument specifying the name of the attribute.

Return Values

Outputs 1 if the command is successful, or -1 in case of error.

Examples

Get the text attribute for a pattern.

```
foreach name $patterns {  
  set pat_x [pmatch::get_pattern -name $name]  
  set pat_handle [lindex $pat_x 0]  
  set text_attr [pmatch::add_text_attribute -pattern_name $name \  
    -attr_name pat_attr -attr_val worst_patterns]  
  set text_attr_out [pmatch::get_text_attribute -pattern_name $name \  
    -attr_name pat_attr]  
  if {$text_attr == -1} {  
    puts "Get text attribute was not successful."  
  } else {  
    puts "Get text attribute was successful."  
  }  
}
```

pmatch::set_cglobal

Sets the global constraint (cglobal) value for a pattern. The command overwrites any previously specified cglobal value.

Usage

pmatch::set_cglobal -pattern *pattern_handle* -value *cglobal_value*

Arguments

- **-pattern *pattern_handle***
A required argument specifying the pattern handle.
- **-value *cglobal_value***
A required argument specifying the cglobal_value in user units. If *cglobal_value* is set to 0, then cglobal is removed from the pattern.

Return Values

Returns 1 if the command is successful, or -1 if the command fails.

Examples

Example 1

Set the cglobal value for a pattern.

```
% pmatch::load_pattern_lib -path example_lib.pmdb
Pattern Library DATABASE
% set pat_names [pmatch::get_pattern_names]
p1_tem p2_tem p3_bcm

% set patx [pmatch::get_pattern -name [lindex $pat_names 0]]
{PDL Pattern} {}
% set patx_handle [lindex $patx 0]
PDL Pattern

% pmatch::get_cglobal -pattern $patx_handle
0 0

% pmatch::set_cglobal -pattern $patx_handle -value 2
1

% pmatch::get_cglobal -pattern $patx_handle
2 0

# Update pattern in library to reflect changes
% pmatch::update_pattern -pattern $patx_handle
1
```

Example 2

Assign all patterns in a library a cglobal constraint.

```
% pmatch::load_pattern_lib -path example_lib.pmdb
Pattern Library DATABASE
% set pat_names [pmatch::get_pattern_names]
p1_tem p2_tem p3_bcm

% foreach patternName $pat_names {
    set patx [pmatch::get_pattern -name [lindex $patternName 0]]
    set patx_handle [lindex $patx 0]
    pmatch::set_cglocal -pattern $patx_handle -value 3
    set onecglocal [pmatch::get_cglocal -pattern $patx_handle]
    puts "pattern $patternName: $onecglocal"
}
pattern p1_tem: 3 0
pattern p2_tem: 3 0
pattern p3_bcm: 3 0
```

Related Topics

[pmatch::get_cglocal](#)

[Global Constraints \[Calibre Pattern Matching User's Manual\]](#)

pmatch::set_constraint_label

Defines a label for a constraint.

Usage

pmatch::set_constraint_label -pattern *pattern_handle* -id *constraint_id* -label *label*

Arguments

- **-pattern *pattern_handle***
A required argument specifying the pattern handle.
- **-id *constraint_id***
A required argument specifying the constraint ID. The constraint ID is the second list element in the return value from [pmatch::get_constraints](#).
- **-label *label***
A required argument specifying the constraint label. The label must be 12 characters or less in length.

Return Values

Returns 1 if the command is successful, or -1 if the command fails.

Examples

Get all constraints, then set a label for the second constraint, using the following Tcl script *constraints.tcl*:

```
pmatch::load_pattern_lib -path one_patt.pmdb
set pat_names [pmatch::get_pattern_names]
set patx [pmatch::get_pattern -name [lindex $pat_names 0]]
set patx_handle [lindex $patx 0]
puts "constraints in first pattern"
set constr [pmatch::get_constraints -pattern $patx_handle -labels]
puts "$constr"

puts "\nadding label to constraint 2"
set ret_val [pmatch::set_constraint_label -pattern $patx_handle \
    -id 2 -label "label2"]
puts "returned: $ret_val"
puts "constraint 2 is:"
set label [pmatch::get_constraint_label -pattern $patx_handle -id 2]
puts "$label"
```

Run the script:

pdl_lib_mgr prompt constraints.tcl

The following output is produced. The pattern has two single edge constraints. The first constraint has a label “label1”, while the second constraint does not initially have a label.


```
constraints in first pattern
{SingleEdge 1 label1 e5 {>= 2.251 <= 2.501}} {SingleEdge 2 e1 {>= 0.801 <=
1.201}}
```

```
adding label to constraint 2
returned: 1
constraint 2 is:
label2
```

Related Topics

[pmatch::get_constraints](#)

[pmatch::get_constraint_label](#)

[pmatch::delete_constraint_label](#)

pmatch::set_library_attribute

Sets the value of a global (library-level) attribute.

Usage

```
pmatch::set_library_attribute [ -lib library_handle ]  
    -name attribute_name -value attribute_value
```

Arguments

- **-lib *library_handle***
An optional argument that specifies a pattern library handle returned by [pmatch::create_pattern_lib](#) or [pmatch::load_pattern_lib](#). The active library is used if this argument is not specified.
- **-name *attribute_name***
A required argument specifying the name of the attribute.
- **-value *attribute_value***
A required argument specifying the value of the attribute.

Return Values

Outputs 1 if the command is successful, or -1 in case of error.

Examples

Set an integer attribute for a library.

```
set a_lib [pmatch::load_pattern_lib -path pat1.pmdb]  
set patterns [pmatch::get_pattern_names]  
set num_patterns [llength $patterns]  
set lib_attr [pmatch::set_library_attribute -name errors -value 2]  
if {$lib_attr == -1} {  
    puts "The library attribute was not set."  
} else {  
    puts "The library attribute was set."  
}
```

pmatch::set_pattern_orient

Specifies to match the pattern in the given orientation only. All other orientation values are set to 0 (not matched).

Usage

pmatch::set_pattern_orient -pattern *pattern_handle* -orient *orientation*

Arguments

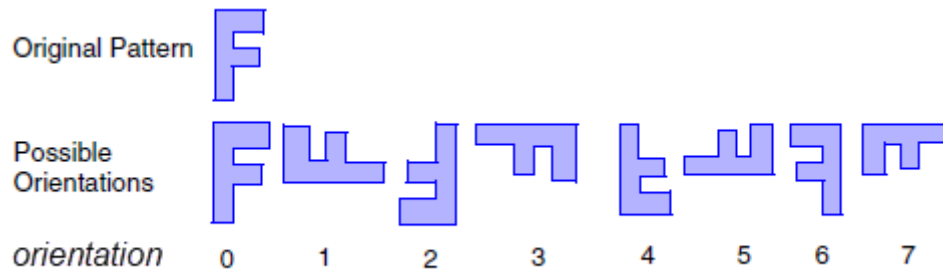
- **-pattern *pattern_handle***

A required argument specifying the pattern handle.

- **-orient *orientation***

A required argument that specifies the orientation, where *orientation* is an integer value in the range $0 \leq \text{orientation} \leq 7$. The possible orientations are shown in the following figure:

Figure 3-2. Orientations for set_pattern_orient



Return Values

Returns 1 for success, -1 for failure.

Description

Specifies that the pattern should match in the given orientation only. The specified orientation is set to 1 (match) and all other orientation values for the pattern are set to 0. The orientations correspond to what is seen in the Pattern Matching GUI in the Orientation section of the **Attributes** tab.

Use [pmatch::add_pattern_orient](#) to add a given orientation to the pattern and leave all other orientations unchanged.

Examples

The example uses [pmatch::add_pattern_orient](#) and [pmatch::set_pattern_orient](#) to set the matched orientations for all patterns in a library. The command [pmatch::get_pattern_orient](#) is used to print out the orientation values.

```
set handle [load_pattern_lib -path ./seedpatt.pmdb -mode rw]
set_active_library -lib $handle
set pat_names [pmatch::get_pattern_names]

foreach patternName $pat_names {
    set myPat [pmatch::get_pattern -name $patternName]
    set myPat_handle [lindex $myPat 0]
    puts "Pat Name : $patternName"

    set orient_val [pmatch::get_pattern_orient -pattern $myPat_handle]
    puts "Original Orientations: $orient_val "

    # set to match in orientation 1 only
    pmatch::set_pattern_orient -pattern $myPat_handle -orient 1
    # add matching in orientations 3, 5, and 7
    pmatch::add_pattern_orient -pattern $myPat_handle -orient 3
    pmatch::add_pattern_orient -pattern $myPat_handle -orient 5
    pmatch::add_pattern_orient -pattern $myPat_handle -orient 7

    set orient_val [pmatch::get_pattern_orient -pattern $myPat_handle]
    puts "New Orientations: $orient_val\n"
    pmatch::update_pattern -pattern $myPat_handle
}
```

Related Topics

[Pattern Orientation \[Calibre Pattern Matching User's Manual\]](#)

Groups and Filters

Group commands are used to collect and manipulate a group of patterns that are a subset of the patterns in a pattern library. Pattern filters select patterns with specific keys and/or properties.

A pattern group is associated with a specific pattern library. A pattern group only exists during the Pattern Matching API session. You can add patterns to the group by pattern name or create filters to select the patterns that are added. The patterns within a group can be saved to a new library; however, group names are not saved to the pattern library file.

To create and use pattern groups, you must first load a pattern library using the [pmatch::create_pattern_lib](#) or [pmatch::load_pattern_lib](#) command.

For information on creating and managing libraries using the Calibre Pattern Matching GUI, refer to the [Calibre Pattern Matching User's Manual](#).

pmatch::filter_add	102
pmatch::filter_create	104
pmatch::filter_delete	105
pmatch::filter_reset	106
pmatch::group_add_filtered	107
pmatch::group_add_pattern	109
pmatch::group_create	111
pmatch::group_delete	113
pmatch::group_reset	114
pmatch::group_save	116
pmatch::list_group_patterns	118

pmatch::filter_add

Adds key name or property conditions to a pattern filter.

Usage

```
pmatch::filter_add -filter_name filter_name  
  { {-key_name key_name }  
    | {-property_name property_name [-value operator1 val1 [operator2 val2]]} }
```

Arguments

- **-filter_name *filter_name***
A required argument set that specifies the name of a pattern filter that was created with [pmatch::filter_create](#).
- { **-key_name *key_name*** }
 | { **-property_name *property_name*** [-value *operator1 val1* [*operator2 val2*]] }
A required argument choice that specifies the filter parameters. Either **-key_name** or **-property_name** must be specified.
 - key_name *key_name***
Specifies the name of the key to filter on.
 - property_name *property_name*** [-value *operator1 val1* [*operator2 val2*]]
Specifies the name of the property to filter on.
The -value argument set species an optional constraint on the property value. If a property condition is added without specifying a constraint, the condition is satisfied by any pattern that includes the specified property name.

The following table lists the supported constraint operators:

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

If two operator-value pairs are used to specify a range, *operator1* must be > or >=, *operator2* must be < or <=, and *val1* must be less than *val2*. For example:
-value > 5.0 <= 7.0.

Return Values

Returns 1 if the call is successful, or -1 if the call fails.

Description

Adds a key name or property condition to the specified pattern filter.

Multiple conditions can be added to a filter by issuing the `pmatch::filter_add` command multiple times for the named filter. The conditions are combined with AND logic, meaning that a pattern must satisfy all conditions in the filter.

Filtered patterns are added to a named group with the [pmatch::group_add_filtered](#) command.

Examples

Filter on a key name and a property condition.

```
% pmatch::load_pattern_lib -path example_lib.pmdb
Pattern Library DATABASE
% set pat_names [pmatch::get_pattern_names]
p1_tem p2_tem p3_bcm

% set myfil "temp_fltr1"
temp_fltr1

# Create a filter
% pmatch::filter_create -filter_name $myfil
Filter "temp_fltr1" created.
1

# Filter on patterns with key name "k1"
% pmatch::filter_add -filter_name $myfil -key_name k1
Added constraint to filter temp_fltr1.
1

# Filter on patterns with property p1 less than 0.01
% pmatch::filter_add -filter_name $myfil -property_name p1 -value < 0.01
Added constraint to filter temp_fltr1.
1
```

Related Topics

[pmatch::filter_create](#)

[pmatch::filter_reset](#)

[pmatch::filter_delete](#)

[pmatch::group_add_filtered](#)

pmatch::filter_create

Creates a pattern filter.

Usage

pmatch::filter_create -filter_name *filter_name*

Arguments

- **-filter_name** *filter_name*

A required argument that specifies the name for the new pattern filter.

Return Values

Returns 1 if the call is successful, or -1 if the call fails.

Description

Creates a pattern filter. The filter conditions are set with the [pmatch::filter_add](#) command.

Pattern filters are independent of any pattern library. Filter conditions are applied with the [pmatch::group_add_filtered](#) command.

Examples

Create a new filter.

```
% pmatch::filter_create -filter_name myfilter
Filter "myfilter" created.
1
```

Related Topics

[pmatch::filter_reset](#)

[pmatch::filter_delete](#)

[pmatch::filter_add](#)

[pmatch::group_add_filtered](#)

`pmatch::filter_delete`

Deletes a pattern filter.

Usage

`pmatch::filter_delete` **`-filter_name`** *filter_name*

Arguments

- **`-filter_name`** *filter_name*

A required argument that specifies the name of a pattern filter created with the [`pmatch::filter_create`](#) command.

Return Values

Returns 1 if the call is successful, or -1 if the call fails.

Examples

Delete a filter.

```
% pmatch::load_pattern_lib -path example_lib.pmdb
Pattern Library DATABASE
% set pat_names [pmatch::get_pattern_names]
p1_tem p2_tem p3_bcm

% pmatch::filter_create -filter_name myfilter
Filter "myfilter" created.
1

...

% pmatch::filter_delete -filter_name myfilter
Filter "myfilter" deleted.
1
```

Related Topics

[`pmatch::filter_create`](#)

[`pmatch::filter_reset`](#)

[`pmatch::filter_add`](#)

`pmatch::filter_reset`

Erases all filter settings for a pattern filter.

Usage

`pmatch::filter_reset` **`-filter_name`** *filter_name*

Arguments

- **`-filter_name`** *filter_name*

A required argument that specifies the name of a pattern filter created with the [pmatch::filter_create](#) command.

Return Values

Returns 1 if the call is successful, or -1 if the call fails.

Examples

Reset a filter.

```
% pmatch::load_pattern_lib -path example_lib.pmdb
Pattern Library DATABASE
% set pat_names [pmatch::get_pattern_names]
p1_tem p2_tem p3_bcm

% pmatch::filter_create -filter_name myfilter
Filter "myfilter" created.
1

...

% pmatch::filter_reset -filter_name myfilter
Filter "myfilter" reset.
1
```

Related Topics

[pmatch::filter_create](#)

[pmatch::filter_delete](#)

[pmatch::filter_add](#)

[pmatch::group_add_filtered](#)

pmatch::group_add_filtered

Applies the specified filter to the library associated with the pattern group and adds the selected patterns to the group.

Usage

pmatch::group_add_filtered -group_name *group_name* -filter_name *filter_name*

Arguments

- **-group_name** *group_name*
A required argument that specifies the name of a pattern group created with the [pmatch::group_create](#) command.
- **-filter_name** *filter_name*
A required argument that specifies the name of a pattern filter created with the [pmatch::filter_create](#) command.

Return Values

Returns 1 if the call is successful, or -1 if the call fails.

Description

Applies the specified filter to the pattern library associated with the pattern group. All patterns that satisfy the filter conditions are added to the named group.

You must load a pattern library in order to use groups; see [pmatch::create_pattern_lib](#) and [pmatch::load_pattern_lib](#).

Examples

Create a group and a filter, and apply the filter to the group. Save the group to a new library.

```
% pmatch::load_pattern_lib -path example_lib.pmdb
Pattern Library DATABASE
% set pat_names [pmatch::get_pattern_names]
p1_tem p2_tem p3_bcm

% set mygroup "temp_grp"
temp_grp
% set myfilter "temp_fltr"
temp_fltr

# Create a group and a filter
% pmatch::group_create -group_name $mygroup
Group temp_grp created.
1
% pmatch::filter_create -filter_name $myfilter
Filter "temp_fltr" created.
1
```

```
# Add a condition for the key name "k1" to the filter
pmatch::filter_add -filter_name $myfilter -key_name k1
Added constraint to filter temp_fltr.
1

# Apply the filter to the group
pmatch::group_add_filtered -group_name $mygroup -filter_name $myfilter
1

# Save the group to a new library
pmatch::group_save -group_name $mygroup -path lib_k1.pmdb
1
```

Related Topics

[pmatch::group_create](#)

[pmatch::group_reset](#)

[pmatch::group_delete](#)

[pmatch::group_add_pattern](#)

[pmatch::list_group_patterns](#)

[pmatch::group_save](#)

[pmatch::filter_add](#)

pmatch::group_add_pattern

Adds a pattern to a group.

Usage

pmatch::group_add_pattern -group_name *group_name* -pattern_name *pattern_name*

Arguments

- **-group_name *group_name***
A required argument that specifies the name of a pattern group created with the [pmatch::group_create](#) command.
- **-pattern_name *pattern_name***
A required argument that specifies the name of the pattern to add to the group. The pattern must exist within the library associated with the group.

Return Values

Returns 1 if the call is successful, or -1 if the call fails.

Examples

Add a pattern to a pattern group.

```
% pmatch::load_pattern_lib -path example_lib.pmdb
Pattern Library DATABASE
% set pat_names [pmatch::get_pattern_names]
p1_tem p2_tem p3_bcm

% set patname1 [lindex $pat_names 0]
p1_tem
% set mygroup "temp_grp"
temp_grp

% pmatch::group_create -group_name $mygroup
Group temp_grp created.
1

% pmatch::group_add_pattern -group_name $mygroup -pattern_name $patname1
Pattern named "p1_tem" inserted into group temp_grp.
1
```

Related Topics

[pmatch::group_create](#)

[pmatch::group_reset](#)

[pmatch::group_delete](#)

[pmatch::group_add_filtered](#)

[pmatch::list_group_patterns](#)

[pmatch::group_save](#)

pmatch::group_create

Creates a pattern group.

Usage

```
pmatch::group_create -group_name group_name [-lib library_handle]
```

Arguments

- **-group_name *group_name***
A required argument that specifies the name of the pattern group.
- **-lib *library_handle***
An optional argument that specifies a pattern library handle returned by [pmatch::create_pattern_lib](#) or [pmatch::load_pattern_lib](#). If this option is not specified, the active pattern library is used.

Return Values

Returns 1 if the call is successful, or -1 if the call fails.

Description

Creates a new pattern group. The group is associated with the library specified by the -lib argument, or with the active library if -lib is not specified. A pattern group is always associated with a particular pattern library.

Use the [pmatch::group_add_pattern](#) and [pmatch::group_add_filtered](#) commands to add patterns to the group.

Examples

Create a new group.

```
% set x_lib [pmatch::load_pattern_lib -path example_lib.pmdb]
Pattern Library DATABASE
% set pat_names [pmatch::get_pattern_names]
p1_tem p2_tem p3_bcm

% pmatch::group_create -group_name mygroup -lib $x_lib
Group mygroup created.
1
```

Related Topics

[pmatch::group_reset](#)

[pmatch::group_delete](#)

[pmatch::group_add_pattern](#)

[pmatch::group_add_filtered](#)

[`pmatch::list_group_patterns`](#)

[`pmatch::group_save`](#)

pmatch::group_delete

Deletes the specified pattern group.

Usage

pmatch::group_delete -group_name *group_name*

Arguments

- **-group_name** *group_name*

A required argument that specifies the name of the pattern group to delete. Pattern groups are created with the [pmatch::group_create](#) command.

Return Values

Returns 1 if the call is successful, or -1 if the call fails.

Examples

Delete a group.

```
% set x_lib [pmatch::load_pattern_lib -path example_lib.pmdb]
Pattern Library DATABASE
% set pat_names [pmatch::get_pattern_names]
p1_tem p2_tem p3_bcm

% pmatch::group_create -group_name mygroup -lib $x_lib
Group mygroup created.
1

...

% pmatch::group_delete -group_name mygroup
Group mygroup deleted.
1
```

Related Topics

[pmatch::group_create](#)

[pmatch::group_reset](#)

[pmatch::group_add_pattern](#)

[pmatch::group_add_filtered](#)

[pmatch::list_group_patterns](#)

[pmatch::group_save](#)

`pmatch::group_reset`

Deletes all patterns in a pattern group.

Usage

`pmatch::group_reset -group_name group_name`

Arguments

- **`-group_name group_name`**

A required argument that specifies the name of a pattern group created with the [`pmatch::group_create`](#) command.

Return Values

Returns 1 if the call is successful, or -1 if the call fails.

Examples

Delete the patterns in a group.

```
% set x_lib [pmatch::load_pattern_lib -path example_lib.pmdb]
Pattern Library DATABASE
% set pat_names [pmatch::get_pattern_names]
p1_tem p2_tem p3_bcm

% set patname1 [lindex $pat_names 0]
p1_tem

% set mygroup "temp_grp"
temp_grp

% pmatch::group_create -group_name $mygroup
Group temp_grp created.
1

% pmatch::group_add_pattern -group_name $mygroup -pattern_name $patname1
Pattern named "p1_tem" inserted into group temp_grp.
1
% pmatch::group_reset -group_name mygroup
Group mygroup reset.
1
```

Related Topics

[pmatch::group_create](#)

[pmatch::group_delete](#)

[pmatch::group_add_pattern](#)

[pmatch::group_add_filtered](#)

[pmatch::list_group_patterns](#)

[pmatch::group_save](#)

pmatch::group_save

Saves the patterns in a pattern group to a new pattern library.

Usage

pmatch::group_save -group_name *group_name* -path *library_path*

Arguments

- **-group_name** *group_name*
A required argument that specifies the name of a pattern group created with the [pmatch::group_create](#) command.
- **-path** *library_path*
A required argument that specifies the path of the new pattern library file.

Return Values

Returns 1 if the call is successful, or -1 if the call fails.

Description

Saves the patterns in the specified pattern group to a new pattern library (PMDB). Pattern group names are not saved to the pattern library file.

A pattern group exists in memory during the Pattern Matching API session and is not saved when you exit the API session.

Examples

Save the patterns in a group to a new pattern library file.

```
% pmatch::load_pattern_lib -path example_lib.pmdb
Pattern Library DATABASE
% set pat_names [pmatch::get_pattern_names]
p1_tem p2_tem p3_bcm

% set mygroup "temp_grp"
temp_grp
% set myfilter "temp_fltr"
temp_fltr
```

```
# Create a group and a filter
% pmatch::group_create -group_name $mygroup
Group temp_grp created.
1
% pmatch::filter_create -filter_name $myfilter
Filter "temp_fltr" created.
1
# Add a filter condition for the key name "k1"
pmatch::filter_add -filter_name $myfilter -key_name k1
Added constraint to filter temp_fltr.
1

# Apply the filter to the pattern group. This adds patterns with
# key name "k1" to the group.
pmatch::group_add_filtered -group_name $mygroup -filter_name $myfilter
1

# Save the patterns in the group to a new library
pmatch::group_save -group_name $mygroup -path lib_k1.pmdb
1
```

Related Topics

[pmatch::group_create](#)

[pmatch::group_reset](#)

[pmatch::group_delete](#)

[pmatch::group_add_pattern](#)

[pmatch::group_add_filtered](#)

[pmatch::list_group_patterns](#)

`pmatch::list_group_patterns`

Lists all patterns in a pattern group.

Usage

`pmatch::list_group_patterns -group_name group_name`

Arguments

- **`-group_name group_name`**

A required argument that specifies the name of a pattern group created with the [`pmatch::group_create`](#) command.

Return Values

Returns a Tcl list of pattern names, or -1 if the call fails.

Examples

Add patterns to a new group and list the patterns in the group.

```
% pmatch::load_pattern_lib -path example_lib.pmdb
Pattern Library DATABASE
% set pat_names [pmatch::get_pattern_names]
p1_tem p2_tem p3_bcm

% set patname1 [lindex $pat_names 0]
p1_tem
% set patname2 [lindex $pat_names 1]
p2_tem
% set mygroup "temp_grp"
temp_grp

% pmatch::group_create -group_name $mygroup
Group temp_grp created.
1

% pmatch::group_add_pattern -group_name $mygroup -pattern_name $patname1
Pattern named "p1_tem" inserted into group temp_grp.
1
% pmatch::group_add_pattern -group_name $mygroup -pattern_name $patname2
Pattern named "p2_tem" inserted into group temp_grp.

% pmatch::list_group_patterns -group_name $mygroup
p1_tem p2_tem
```

Related Topics

[`pmatch::group_create`](#)

[`pmatch::group_reset`](#)

[`pmatch::group_delete`](#)

[`pmatch::group_add_pattern`](#)

`pmatch::group_add_filtered`

`pmatch::group_save`

Help Command

This command lists all of the available API commands. If a command name is specified, it prints a help message about this command.

For information on creating and managing libraries using the Calibre Pattern Matching GUI, refer to the [Calibre Pattern Matching User's Manual](#).

pmatch::help..... **121**

pmatch::help

Gets usage information for a API command or outputs the list of API commands.

Usage

pmatch::help [-command *command_name*]

Arguments

- -command *command_name*

An optional argument that specifies a pattern matching API command name for which to return a help message.

Return Values

Returns 1 if the call is successful, or -1 if the call fails.

Description

When the -command option is specified, usage information for the specified API command is printed. If the -command option is not specified, the list of available API commands is printed.

Examples

List usage information for the [pmatch::get_keys](#) command.

```
% pmatch::help -command get_keys
pmatch::get_keys: Returns the keys list defined in the pattern.
Usage:
pmatch::get_keys -pattern <pattern_handle>
Output: List of keys
```


— Symbols —

`[]`, 13

`{}`, 13

`|`, 13

— A —

`add_custom_bbox_marker`, 47

`add_keys`, 68

`add_marker`, 50

`add_properties`, 70

— B —

Bold words, 13

— C —

`check_pattern`, 41

Command reference, 13

Courier font, 13

`create_pattern_lib`, 23

— D —

`delete_keys`, 76

`delete_marker`, 51

`delete_properties`, 78

Double pipes, 13

— F —

`filter_add`, 102

`filter_create`, 104

`filter_delete`, 105

`filter_reset`, 106

— G —

`get_cglobal`, 80

`get_class`, 42

`get_constraints`, 74, 81, 83, 96

`get_edges`, 55

`get_extent_type`, 60

`get_keys`, 86

`get_layer_names`, 25

`get_markers`, 52

`get_pattern`, 43

`get_pattern_name`, 44

`get_pattern_names`, 26

`get_precision`, 45

`get_properties`, 90

`get_property_value`, 92

`get_vertices`, 57

`group_add_filtered`, 107

`group_add_pattern`, 109

`group_create`, 111

`group_delete`, 113

`group_reset`, 114

`group_save`, 116

— H —

Heavy font, 13

help, 121

— I —

Italic font, 13

— L —

`list_group_patterns`, 118

`load_pattern_lib`, 29

— M —

Minimum keyword, 13

— P —

Parentheses, 13

Pipes, 13

— Q —

Quotation marks, 13

— R —

`remove_pattern`, 31

`rename_layers`, 33

`rename_pattern`, 34

— S —

`set_active_library`, 35

set_cglobal, [27](#), [36](#), [94](#)

Slanted words, [13](#)

snap_extent, [63](#)

Square parentheses, [13](#)

— U —

Underlined words, [13](#)

update_pattern, [38](#)

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.

