klayout 0.26 User Manual (Flat file format)

Compiled from Web documentation at
https://www.klayout.de/doc-qt4/manual/

20190919

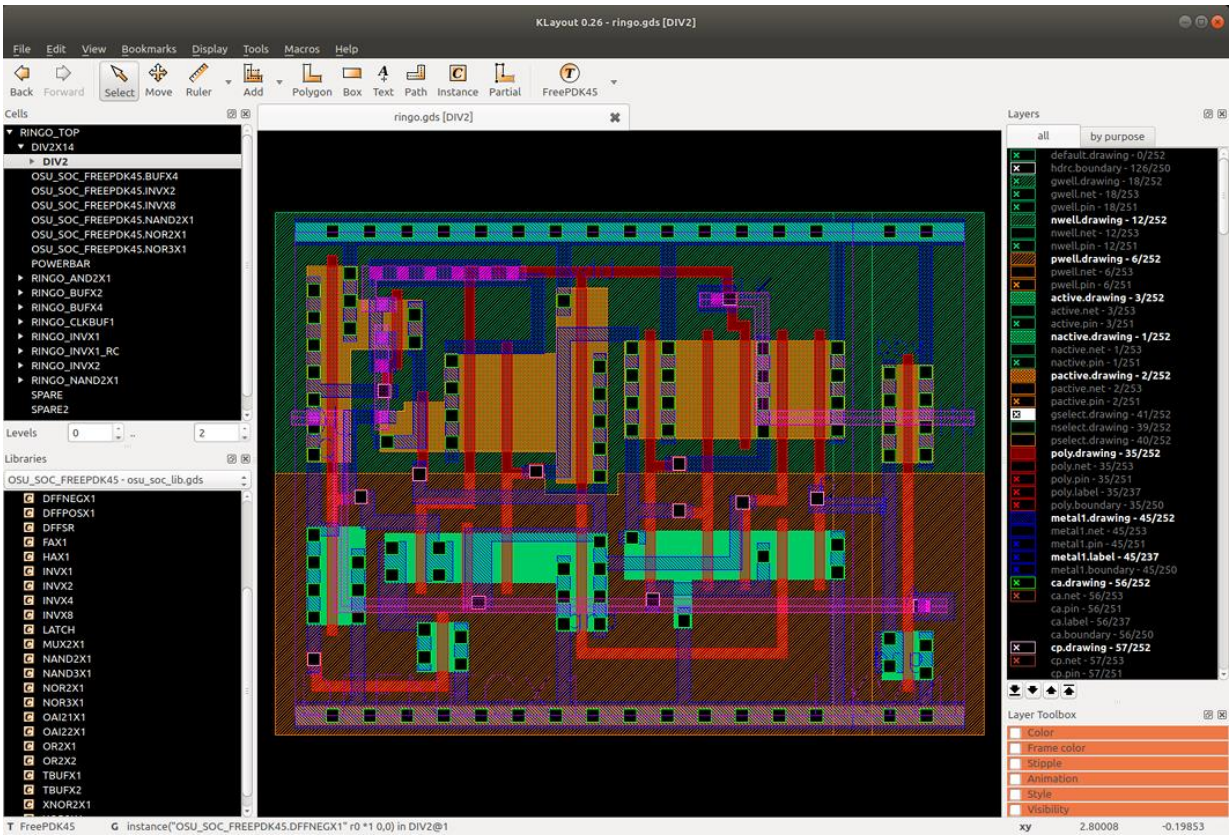
Contents

KLayout Basics

Welcome to KLayout's user manual. This is the manual chapter covering the basic features of KLayout. The following subtopics are available:

The Main Window

The main window is divided into three parts: the left area is the hierarchy browser and navigator, the center part of the canvas and the right part is the layer list with the layer toolbox. The individual components can be rearranged, so the arrangement described is just the default arrangement. You can move a component to a new place by dragging it with it's title bar to some other place or detach it from the main window to form a floating separate window.

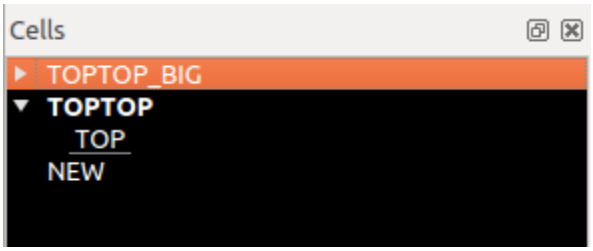


Left Part - The Hierarchy Browser, Library View and Navigator

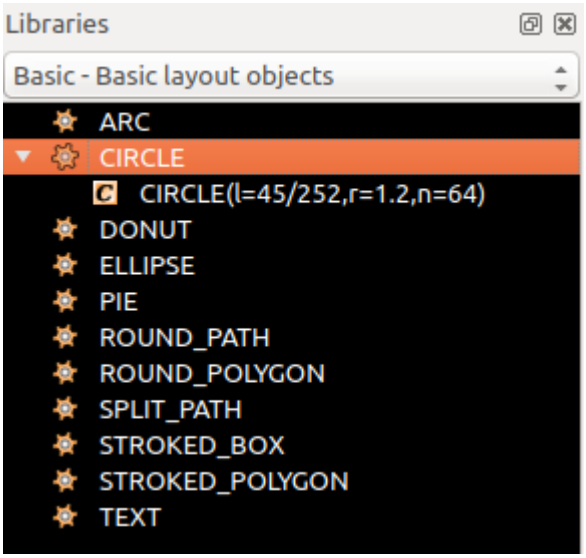
The left panel is the hierarchy browser which shows the cell hierarchy and - optionally - the navigator window that shows an overview over the whole layout.

In the hierarchy browser, cell nodes can be expanded showing the child nodes. The "current cell" is the one shown in the center panel. It is drawn in bold font. One or multiple cells can be selected. The selected cells are the ones, the various functions act on. The "context cell" is the cell which is the "active cell" on which drawing happens. The context usually is the same than the current cell, but by descending into the hierarchy, the child cell of the current cell can be made the context cell. It is shown in underlined font.

In the following example, "TOPTOP_BIG" is selected, "TOPTOP" is the current cell and "TOP" is the context cell:

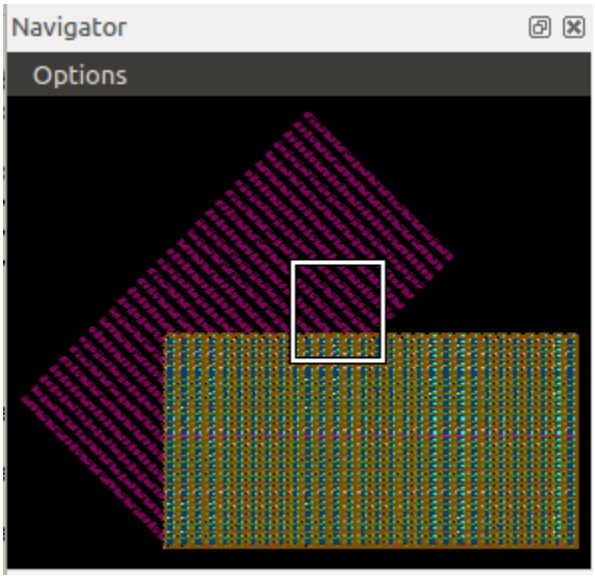


The sub-panel below the hierarchy browser is the library view. This view shows the libraries registered in the system and their content:



The library view is used to browse and place library cells, which can be normal cells or PCells. Read more about the library view here: The Library View.

The navigator is invisible by default but can be activated by checking the "Navigator" menu item in the "View" menu. The navigator shows an overview image of the whole cell and a box indicating the clip shown in the center panel:



Center Part - The Canvas

The center panel is the actual canvas. There, the layout is drawn. To zoom in, click with the right mouse button and drag a rectangle that will become the new area shown. Select items by left-clicking or dragging a selection rectangle with the left mouse button pressed. A variety of edit and display modification feature is supplied, for example the ability to add rulers or background images.

Right Part - The Layer List and Layer Style Controls

The right panel is shows the layer list or layer tree. By default, it shows a plain layer list, but the are manifold ways to configure the list, i.e. grouping, styling, adding tabs to easily switch between different setups etc.

The layer tree specifies what and how layout is drawn. Entries can be deleted or configured freely - this does not affect the layout itself. Essentially, the layer list can be entirely independent from the layout and omit layers from the layout or add (empty) layers that are not actually part of the layout. In this scenario, a layer list typically reflects a set of drawing layers with a certain technology-dependent meaning. This list provides a styled layer view for an otherwise undecorated layout file. The layers can be reordered, so their drawing priority is changed. The top layer will be drawn first, while others will be drawn later. Hence the first layer is likely to become obscured by following layers.

Modification operators can be attached to layers - for example, the drawn layout can be transformed geometrical (i.e. translated, rotated, scaled, mirrored) and layout can be selectively drawn when certain conditions for user properties apply.

Below the layer list, a set of control panels is located. The control panels are minimized per default. They can be expanded by clicking on the header bar. These controls form the "Layer

Toolbox" where you can modify the layer styles. The styles selected in the layer toolbox will affect the layers selected in the layer list.

Multiple layouts can be shown together. Either they can be overlayed or they can be shown in separate views. In this case, a tab panel appears at top of the main window. Selecting a tab switches between the layouts. Layers for different layouts appear as annotated layers in the layer list - for example "1/0@1" for layer 1, datatype 0, first layout and "1/0@2" for layer 1, datatype 0 and second layout.

Loading A File

In the "File" menu, choose

- "Open" to close the current view and open a new layout instead of the currently loaded one
- "Open in Same Panel" to open a new layout in addition to the currently loaded one
- "Open in New Panel" to open a new layout in a new view

Either way, a file selection dialog will appear where a file can be chosen for loading. After choosing the file and clicking "Ok", the file is loaded.

The program will automatically determine the type of the file. Currently, OASIS, GDS2, DXF, CIF, LEF/DEF and a text version of GDS2 are supported. Gerber PCB data can be read with some preparations too (see Import Gerber PCB Files). If the file is gzip/zlib compressed, it will be uncompressed automatically.

Certain options can be specified for the file loader using the reader option pages. To open the reader options dialog, choose "Reader Options" from the "File" menu. This dialog allows specification of certain options for all "Open" actions, for example:

- Confine the reader to a certain set of layers. All other layers are not read.
- Disable reading of text objects. Text objects don't carry geometrical information for masks and can be discarded this way.
- Disable reading of user properties. If properties are not required, the memory consumption can be reduced by disabling properties.
- Certain GDS specific options which mainly control the level of compatibility with other tools.
- Other formats may offer other options too. Specifically rich formats such as DXF or LEF/DEF can be configured in manifold ways. Different tabs show options for different formats or format groups.

Using "File/Reload", the currently loaded file can be re-read from disk. Usually this is not required, because KLayout will automatically check whether the file has changed and offer to load it.

By picking a file from the "Open Recent" list in the "File" menu, a previous file can be loaded again.

Files can be given to KLayout on the command line and are loaded automatically. Multiple files can be specified. They are shown in different pages by default. To load multiple files into the

same page, add a "-s" option to the command line. "http:" or "https:" can be specified on the command line as well. In this case, KLayout will download the files from the given URL.

Files and URL's can be dragged and dropped on the KLayout main window. KLayout will then load and show these files.

Layout files can be associated with technologies. Technologies allow associating a layout with additional data, such as libraries, macros, net tracer settings, layer properties etc. Read About Technology Management for details.

Managing The Panels And Loaded Layouts

Choose "Close" in the "File" menu to remove a layout of a panel and close the panel unless there are still layouts loaded. If multiple layouts are loaded into the current panel, a dialog appears. This allows selecting one or many layouts for closing. "Close All" will close all panels.

Choose "Clone" from the "File" menu to duplicate a panel. A new panel will be created that is an exact copy of the current one. Both, the current and the new panel are views to the same layout. This way, only one copy of the layout is held in memory.

Choose "Pull In Other Layout" to combine other layouts already loaded into the current panel. Basically, KLayout allows viewsing a layout in multiple panels, either on it's own in different configurations or together with other layouts. "Pull In Other Layout" allows configuration of a panel to show another layout which has been loaded into another panel. In that sense it's the reverse of closing one layout from a panel showing multiple layouts.

Choosing A Cell

To show a certain cell, select the cell in the cell hierarchy in hierarchy browser to the left. Then, right-click in the cell tree to bring up the context menu and choose "Show as top" or simply select the cell with the middle mouse button.

To select a cell by name, choose "Select Cell" in the "Display" menu. A dialog will appear that allows selecting a cell by name or choosing it from an alphabetically sorted list. Additionally, this dialog allows navigating the cell tree by moving to one of the child or parent cells.

Choosing A Hierarchy Depth

By default, only the bounding box of the cell selected is shown. This corresponds to zero hierarchy levels being shown. If you select one more level of hierarchy (Levels 0 to 1), this content of the child cells of the current cell are drawn, but grandchildren (children of child cells) are drawn as boxes. Increasing the hierarchy levels will draw more and more details of child cells until all cells below the current cell are drawn in detail.

To select more hierarchy levels

- Select "Full Hierarchy" from the "Display" menu or press the "*" key to show all hierarchy levels
- Select "Box Only" from the "Display" menu or press the "0" key to show only the bounding box (the default)
- Select "Top Level Only" from the "Display" menu or press the "1" key to show the top level elements
- Select "Increment Hierarchy" from the "Display" menu or press the "+" key to show one more hierarchy level
- Select "Decrement Hierarchy" from the "Display" menu or press the "-" key to show one hierarchy level less
- Use the hierarchy level entry fields below the cell list to change the current minimum or maximum level

Configuring The Cell List

Two modes are provided for the cell list: a tree view (the default) and a flat cell list. To switch to flat mode, check the "Flat Cell List" option in the cell panel's context menu.

If multiple layouts are loaded, the cell lists of the individual layouts are shown separately. The drop-down box above the cell lists will select the current cell tree to show. An alternative mode is available in which the cell lists are shown beside each other in the cell tree panel. This mode is enabled by choosing the "Split Mode" option in the cell panel's context menu. In split mode, you can click on the headers to select the current cell tree. The current cell tree plays a role in some cases, for example for the layout operations available in the "Layout" submenu of the "Edit" menu.

In addition, three sorting modes are provided: alphabetically by name and by cell size (bounding box area), descending and ascending. The cell size is supposed to reflect the design level: library and leaf cells are usually small which macro blocks are usually large. By using cell size sorting in ascending order, the leaf cells will be shown first. To change the sorting, check the corresponding option in the "Sorting" submenu of the cell panel's context menu.

The Library View

Beside the cell list, a library view is provided as a support view for the layout panel. This sub-panel displays the libraries available and allows browsing the cells and PCells inside a library.

By default, the library view is shown below the cell tree. You can rearrange the views by dragging them at their title bar and docking them in other places of the main window. To reset the window arrangement to the default configuration, use "Restore Window" from the "View" menu.

The library view shows the cells and PCells of one library. To select a library, choose it from the selection box at the top of the library view.

PCells are shown with a small "gear" icon in the library view. If PCells are instantiated, the variants in use are shown as entries below the PCell entry.

In edit mode, cells can conveniently be placed by dragging them from the library view to the layout canvas. If a PCell variant is dragged, another instance of this PCell variant is created. If a PCell master is dragged, KLayout will pop up the PCell parameter definition dialog on drop.

Hiding Cells

Independent of the hierarchy levels shown, cells can be hidden. In this case, the cell is now shown but rather the bounding box is shown. To do so, select the cell from the cell list and choose "Hide" from the context menu. To show a cell again, choose "Show". To make all cells visible again, choose "Show All".

Zooming Into The Layout

Select the zoom area with the right mouse button in the layout canvas.

Press the button, drag the box to the desired position and release the button.

To zoom in (enlarge) drag the box right and down. To zoom out (shrink) drag the box up and left. To choose a new center, single-click the new center point with the right mouse button.

Additionally, these functions are available from the menu or by hotkeys:

- Pan to the left, right, top or bottom using the arrow keys or choosing one of these functions from the "Display" menu.
- Fill the whole selected cell into the window by pressing "F2" or choosing "Zoom Fit" from the "Display" menu
- Zoom in or out by a fixed amount by pressing "Enter" or "Shift+Enter" or choosing "Zoom In" or "Zoom Out" from the "Display" menu
- Zoom in and out by using the mouse wheel if available. The current mouse location will stay fixed, while the surrounding layout will be enlarged or reduced in size
- Press "Shift" while dragging the mouse with the right mouse button pressed will drag the layout similar to what happens in recent map service web applications

Global Rotation and Flip

KLayout offers an option to flip or rotate the whole window. This is a useful option for example to view mask data which a mirrored image of the chip.

To access this option, choose the desired transformation from one of the options available in the "Global Transformation" submenu of the "Display" menu.

Global Rotation and Flip

KLayout offers an option to flip or rotate the whole window. This is a useful option for example to view mask data which a mirrored image of the chip.

To access this option, choose the desired transformation from one of the options available in the "Global Transformation" submenu of the "Display" menu.

Returning To A Previous View

Choose "Last State" from the "Display" menu to return to the last window shown. Choose "Next state" to switch to a more recent state again.

Bookmarking Views

Views (window, cell) can be bookmarked for later retrieval. Choose "Bookmark This View" from the "Bookmarks" menu. A name is required to be entered for the bookmark. The bookmark will then appear in the "Goto Bookmarks" list.

The list of bookmarks defined can be loaded or saved by using the "Load Bookmarks" and "Save Bookmarks" functions from the "Bookmarks" menu.

The bookmark list is available as a dockable tool window as well: check the "View/Bookmark List" option to enable this dock window. The bookmark list by default is shown at the bottom right side of the layout view.

Dockable Bookmark List

To navigate to a bookmark from the dockable bookmark list, double-click the entry. From the context menu (right mouse click) you can select these functions:

- **Follow Selection**: if this option is checked, the selected bookmark will immediately change the view accordingly. With this option, you can browse the bookmark list with the arrow keys while the view updates automatically.
- **Manage bookmarks**: opens the bookmark management dialog (same as from the "Bookmarks" menu).
- **Load bookmarks** and **Save bookmarks**: loads or saves the bookmarks to a file (same function as from the "Bookmarks" menu).

Descending Into A Cell With Context

A cell can be shown either isolated (this is the default, if the cell is the current cell), embedded (as a subcell of the current cell) or as the current cell in the context of another direct or indirect parent cell. In the latter mode, the cell is highlighted while the context cell is shown in dimmed or another, user-defined color.

To highlight a cell in a context, first choose the context cell. Then select a shape or a cell instance within the cell to show in the context and choose "Descend" from the "Display" menu or press "Ctrl+D". Now, the first child cell leading to the selected shape is highlighted, while the surrounding shapes of the parent cell (the previous current cell) is shown in dimmed colors. Choose "Descend" repeatedly to descend further into the hierarchy until the selected shape or

instance is on the level of the current cell. The current cell is show underlined in the cell tree, while the context cell is shown in bold font in the cell tree as usual.

The reverse of this operation is "Ascend" (or "Ctrl+A") available from the "Display" menu.

The way how the context layout is shown can be adjusted in the setup dialog on the "Background" tab.

Choosing A Layer Color

Select the layer or the layers for which to change the color and open the color chooser panel in the layer panel to the right. If the color chooser is not visible, select the small check box in the "Color" header bar. Then the color chooser is expanded.

To change the color, click on the desired color. To select a color not offered in the list, select the "More Colors" button. A color choose dialog will open.

To choose the color of the frame to draw around the shapes, without changing the fill color, use the "Frame Color" chooser panel.

Layers can be "dimmed" by making their color darker or brighter so they contrast less with the background. To do so, choose "Dark" or "Bright" from the color panel. Pressing the button multiple times makes the colors darker or brighter each time. The brightness or darkness can be reset with the "Reset" button.

Telling Used From Unused Layers

In some applications, the layer list will grow very large and keeping track of the important layers may be hard. KLayout provides support for that task in two ways: KLayout checks whether a layer carries any information and displays the layers in a different way in the layer list, if it is empty.

Two ways of checking the information content of a layer are provided: either a layer is said to be empty if the current cell does not have any shapes on it. Alternatively, a layer can be identified to be empty by checking if any shape is shown in the current view (more precisely if any shape's bounding box overlaps with the current view rectangle). The latter mode can be selected in the layer list's context menu with the option "Test For Shapes in View".

If a layer is determined to be empty, it is either grayed out or it is now shown at all. The latter option keeps the layer list short and is selected with "Hide Empty Layers" from the layer list's context menu.

Choosing A Line Style

Line styles control how the outline of the shapes is drawn. The default is a solid line. Available line styles are dotted lines, dashes lines and custom styles. To choose a certain line style, select the layer or the layers for which to change the style and choose the line style from the "Style" panel.

New styles can be created with the "Custom Style" button. A line style editor will come up that allows creating and editing of new styles. The predefined pattern cannot be changed. To apply a new style, select "More" from the style selection panel and choose the new style from the list. The custom styles are saved with the layer display properties.

Animating Layers

Layers can be animated, i.e. made blinking or the fill pattern scroll. Select the layer or the layers for which to change the animation style and choose the animation style from the "Animation" panel. For blinking mode, two phases can be selected: "Blink" and "/Blink". Choosing different phases for two layers makes the layers appear alternatively.

Changing The Display Style

The line width of the element's frame can be changed by using the width buttons on the "Style" panel after having selected the layers to apply the change on. "0px" removes the line, "1px" draws a single-pixel wide line (the default), "2px" a somewhat thicker line and so on.

"Simple" is the normal draw mode while "Marked" draws a cross on each vertex of the element. The cross size is constant so the shapes stay visible even on large scale where the elements would otherwise become single pixels.

A layer can be configured to draw a diagonal cross on rectangles. To enable this style, select "Cross" from the "Styles" toolbox. To disable it, choose "No Cross". The cross drawing will add to the fill pattern and applies to rectangles only.

Changing The Layer Visibility

The selected layers can be made invisible by choosing the "Hide" option on the "Visibility" panel. Choosing "Show" makes the layers visible again.

Also, double-clicking a visible layer in the layer list toggles the layer's visibility.

To make a layer "transparent" (i.e. let the other layers show through), select "Transp." on the "Visibility" panel. To make it opaque again, select "Opaque" (the default).

Valid And Invalid Layers

Starting with version 0.23, KLayout offers "invalid" layers: invalid layers can be visible, but don't participate in selection and snapping. They just act as some kind of background drawing.

Layers can be made invalid by choosing "Make invalid" from the layer's context menu. Layers can be made valid again by choosing "Make valid". Invalid layers are marked with a small "x" in the layer list.

Organizing Layers Hierarchically

Layers can be organized hierarchically. For example, certain layers can be grouped together. Choose "Group" in the context menu of the layer list (right-click the layer list). The selected layers will be replaced by a tree node that represents these layers. Click on the tree node to expand or collapse this group.

Once layers are grouped, they can be hidden or made visible with a single double-click on the node representative. The node representative also controls the appearance of the layers in the group: if a color or style is assigned to the representative, it overrides the respective style of all layers contained in the group. This way for example, the color of the layers contained in the

group can be changed at once. To remove a color override of a node representative, set the color to "None".

To resolve a group, select the group representative and choose "Ungroup" from the context menu.

A variety of automatic grouping methods is provided. For example, the "Regroup views by layout index" from the layer context menu will collect all layers and put them into one group per layout shown in the panel.

Using Multiple Layer Setups With Tabs

With version 0.21, a new feature was introduced. Using tabs in the layer panel it is very simple to switch between different setups.

A layer tab can be created by choosing "New Tab" from the "Tabs" submenu in the layer panel's context menu (right mouse button click). A new tab will appear at the top of the layer properties panel. Initially this tab will be a copy of the current setup. Any edits on the layer properties will apply to this tab only. When switching to a different tab, the layout view will reflect the new tab's settings. That way, different setups can be prepared and easily exchanged.

When the layer properties are saved, the layer properties file will contain all tabs. Thus, a multi-page setup can easily be stored and retrieved.

The initial title of the tab will be the tab number. The title can be set with the "Rename Tab" function in the "Tabs" submenu of the layer panel's context menu. To remove a tab, choose "Remove Tab".

Removing And Adding Layers To The Layer Set

The layers shown in the layer list are rather "pointers" to the actual layout layers and not representing the actual layers. Because of this, these layers are more precisely referred to as "views". Layers can be removed and created again without affecting the actual layout data.

To create a layer, choose "Insert View" from the layer context menu (right mouse button click on the layer list). Then, an input dialog prompts for the source specification. The source

specification tells, from which actual data layer to take the displayed data from. The most simple form of a source specification is "layer/datatype" (i.e. "5/0") or the layer name, if an OASIS layer name is present. This specification can be enhanced by a layout index. The first layout loaded in the panel is referred to which "@1" or by omitting this specification. The source specification "10/5@2" therefore refers to layer 10, datatype 5 of the second layout loaded in the panel.

Source specifications can be wildcarded. That means, either layer, datatype or layout index can be specified by "*". In this case, such a layer must be contained in a group and the group parent must provide the missing specifications. For example, if a layer is specified "10/*" and the parent is specified "*/5", the effective layer looked for will be "10/5". Unlike the behaviour for the display styles, the children override (or specialize) the parent's definition in the case of the source specification.

The layer list can be cleaned up to remove layer views that do not correspond to actual layout layers using the function "Clean up views" from the context menu. Similar, layers that are present in the layout but for which there is no view can be added using the "Add other views" method.

Transforming Views And Property Selectors

The source specification described in the section before is much more powerful than just allowing to describe the data source. In addition to that, the layer can be geometrically transformed and the display can be confined to shapes that belong to a certain class described by a property selector.

A geometrical transformation is specified by appending a transformation in round brackets to the layer/datatype source specification. The format of this transformation is (in any order):

([<dx>,<dy>] [r<angle> | m<angle>] [*<mag>])

For example, "(r90)" specifies a rotation by 90 degree counter-clockwise. "(0,100.0 m45 *0.5)" will shrink the layout to half the size, flip at the 45 degree-axis (swap x and y axes) and finally shift the layout by 100 micron upwards.

A detailed explanation of the transformation syntax can be found in Transformations in KLayout.

(this section is inlined, next)

Transformations accumulate over the layer hierarchy. This means, that if a layer is transformed and the layer is inside a group whose representative specifies a transformation as well, the resulting transformation is the combination of the layer's transformation (first applied) and the group representative's transformation.

Multiple transformations can be present. In this case, the layout is shown in multiple instances.

A particular application is to regroup layers by layout index and assign a transformation to the group representative belonging to a certain layout such that the layouts get aligned.

The property selector is specified in square brackets. A selector combines several expressions of the form "<property>==value>" or "<property>!=<value>" with operators "&&", "||", "!" and allows usage to round brackets to prioritize the evaluation of these operators:

```
[ <expr> ]
```

In GDS2 files, the property is always named with a integer value which is written with a single hash characters (i.e. "#43". The value of a GDS property is always a string. A string is either written as a text atom or can be enclosed in single or double quotes. The following is an example for a valid property selector for GDS files:

```
10/5 [#43==X]
```

With this source specification, the layer will show all shapes from layer 10, datatype 5 which have a user property with number 43 and value string "X". A more complex example is this:

```
10/5 [!(#43==X&&(#2==Y||#2==U))]
```

With OASIS files, the properties can be named with a string. In this case, the property selector can be "[prop==X]" for example. In addition, the value can be a an integer or a double value. This is reflected by the choice of the value: "[prop==#200]" will check, if the property named "prop" has an integer value which is 200. In the same fashion, "[prop==##0.5]" checks, if the property "prop" has a double value and this is 0.5.

Property selectors combine over a layer hierarchy. This means, that if a group representative specifies a property selector and a layer in this group specifies a selector as well, only those shapes will be shown that meet both criteria.

Transformations in KLayout

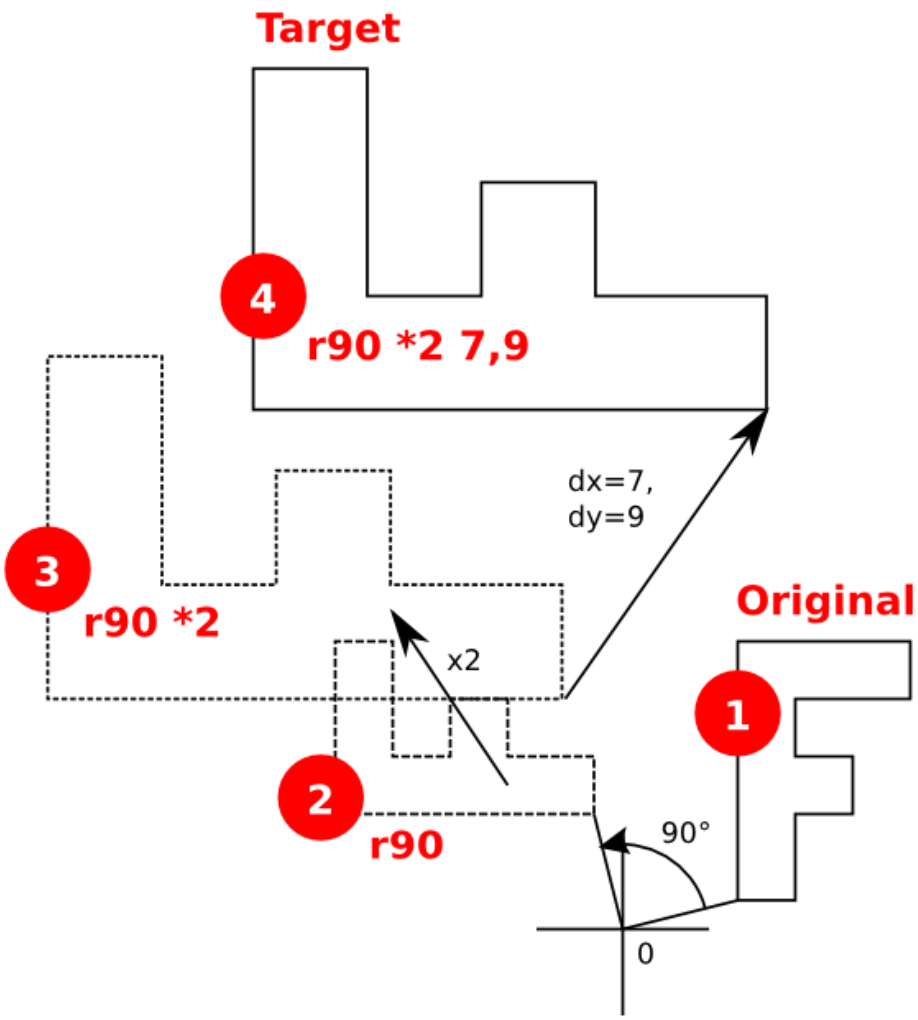
KLayout supports a subset of affine transformations with the following contributions:

- **Rotation and/or mirroring:** rotation by a given angle or mirroring at a given axis.
- **Scaling:** magnification by the given factor.
- **Translation:** a displacement by the given vector.

The execution order is "displacement after rotation, mirroring and scaling". Transformations are used for example to describe the instantiation of a cell. The content of a cell appears in the parent cell after the given transformation has been applied to the content of the cell.

The transformations supported by KLayout cover the transformations employed within GDS2, OASIS and other layout formats. KLayout does not support shearing currently.

The following figure illustrates the effect of the transformation "r90 *2 7,9". This notation specifies a transformation composed of a rotation by 90 degree, a scaling with factor 2 and a displacement by 7 units in x- and 9 units in y-direction. In that example, the "F" shape is first rotated by 90 degree around the origin. Because the "F" is already displaced from the origin, this will also move the "F" shape. The shape then is scaled. Again it will move because every point of the polygon moves away from the origin. Finally it is displaced by the given displacement vector.



The notation shown here is used in many places within KLayout. It is basically composed of the following parts which are combined putting one or more blanks in between. The order the parts are specified is arbitrary: the displacement is always applied after the rotation.

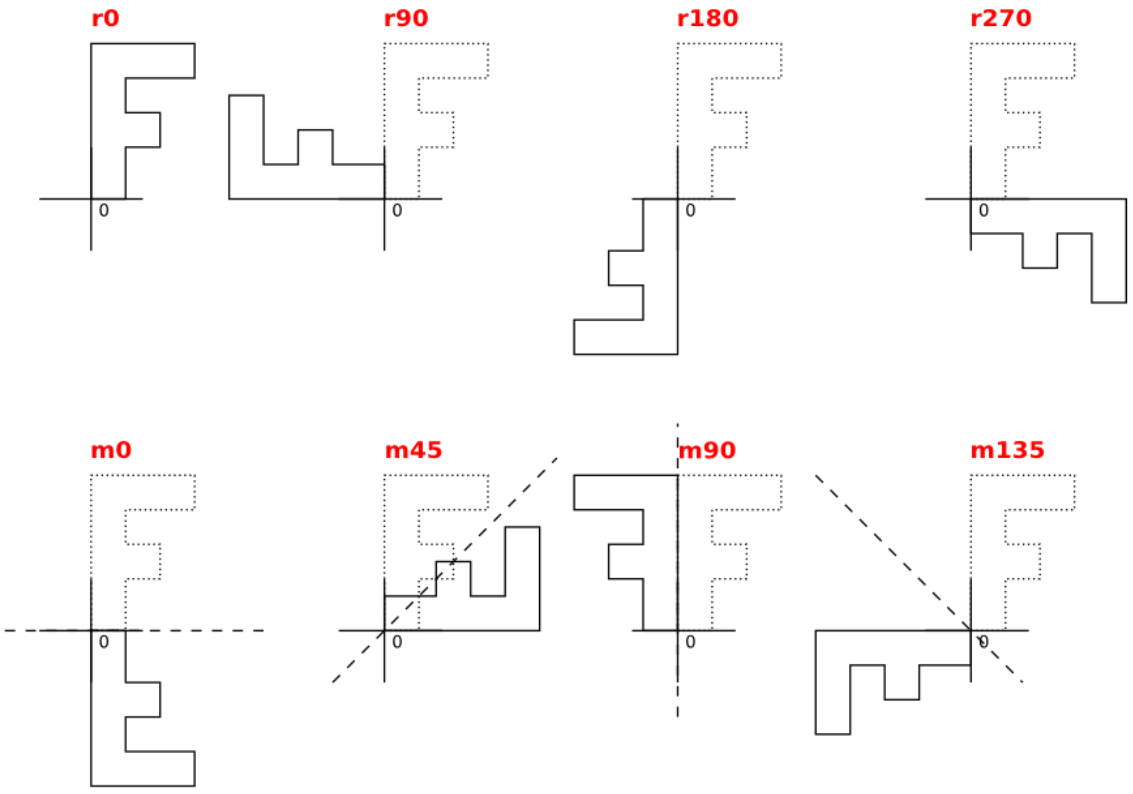
- <x>**,**<y>: A displacement (applied after rotation and scaling) in micron units. If no displacement is specified, "0,0" is assumed.
- **r**<a> or **m**<a>: A rotation by angle "a" (in degrees) or mirroring at the "a" axis (the x axis rotated by "a" degree). If no rotation or mirroring is specified, no rotation is assumed.
- ***<s>: A scaling by the factor "s". If no scaling is specified, no scaling is assumed.

Here are some examples:

- **0,100**: shift 100 units up.
- **r90**: rotation by 90 degree counterclockwise (positive in the mathematical sense).
- **m0**: mirroring at the x-axis.
- **m45 100,-200**: swap x and y (mirror at 45 degree axis), shift 100 units to the right and 200 units down.
- **r22.5 *1.25**: rotate by 22.5 degree and scale by factor 1.25.

The distance units are usually micron. In some cases (i.e. transformations inside a database), the unit is database units and dx and dy are integer values.

Mirroring and rotation are exclusive and mirroring includes a rotation. In fact, a mirror operation at a certain axis is identical to a mirror operation at the x-axis, followed by a rotation by twice the angle "a". The following figure illustrates rotation and mirroring with the eight basic transformations involving rotations by multiples of 90 degree:



KLayout is not restricted to these basic operations. Arbitrary angles are supported (i.e. "r45" or "m22.5"). Usually however, this implies grid snapping and other issues. This also is true for arbitrary scaling values. KLayout is also more effective when using simple transformations involving only rotations by multiples of 90 degree and do not use scaling.

Specifying Explicit Hierarchy Levels For One Layer Or A Layer Group

By default, only the hierarchy levels that are selected in the hierarchy level selection boxes are shown, i.e. if levels 0 to 1 are selected, just the top level shapes and instances are shown. This selection can be modified for certain layers or layer groups. To specify a different hierarchy selection for a certain layer, use an optional source specification element, the hierarchy level selector:

#[<lower-level>][. . <upper-level>]

Upper and lower level can be omitted. In this case, the respective level is not overridden. The upper level can be '*' which means: every level that is available. If just one level and no ".." is given, it is taken as upper level and the lower level is set to zero.

Some examples might illustrate this:

`#*` Display all hierarchy levels
`#0..1` Display top level only
`#..5` Override upper level with 5
`#2..` Override lower level with 2
`#..*` Override upper level setting by "all levels"

Modifications of this notation are provide in order to support more use cases. Instead of specifying a single number for the level, the following alternative notations are supported:

`(1)` Relative specification: Hierarchy level 1 related to the current cell's level. The effective specification differs in "Descend" mode where the current cell is on a lower hierarchy level than the context cell which is the top cell drawn.
`<1` Constrained specification: Hierarchy level 1 or less if the upper or lower default level set in the user interface is less.
`>1` Constrained specification: Hierarchy level 1 or greater if the upper or lower default level set in the user interface is greater.
`(>1)` Combined specification: Hierarchy level 1 related to the current cell's level or less.
`>*` Equals the currently set maximum hierarchy level.

For example:

`#(0)..(1)` The top level of the current cell (works also in "Descend" mode).
`#>0..<1` Everything exactly on top level unless the top level is not selected in the controls.
`#>1..<*` Everything below the context cell's top level unless not selected by the user interface controls.
`#(>1)..<*` Same than before but related to the current cell, not the context cell.

Loading And Saving The Layer Sets

The visual layer properties can be saved to a file using the "Save Layer Properties" function from the "File" menu. This list can be loaded again using the "Load Layer Properties" function.

Creating A Screenshot

To save the canvas as a PNG file, choose "Screenshot" from the "File" menu or press the "Print" key. A file dialog box will appear in which the file can be specified where the screenshot is saved to.

Doing Measurements

A measurement can be performed by clicking on the ruler icon in the toolbar and selecting "Ruler" from the drop-down options. Left-click on a point in the layout and then left-click again to specify the second point. A ruler will be shown that indicates the distance measured.

A more convenient way is provided with the single-click measurement ruler. Select "Measure" from the drop-down options of the ruler symbol. In this mode, a single click will set a ruler to the specified position. This feature will look for edges in the vicinity of the ruler and set the ruler to connect the neighboring edges. The ruler is attached perpendicular to the edge next to the initial point.

Rulers can be configured in manifold ways. Use "Rulers And Annotations Setup" in the "Edit" menu to open the ruler configuration dialog. A ruler can be made to snap to edges of objects by selecting "Snap to edge/vertex". Ruler orientations can be constrained by using the "Angle Constraint" options. The number of rulers can be limited using the "Limit number of annotations" setting.

While drawing or moving one point of a ruler, the direction constraint can be overridden with the Shift and Ctrl keys: pressing Shift while moving the mouse will enforce orthogonal constraint, Ctrl will enforce diagonal constraint while pressing both will release any direction constraint.

All rulers can be cleared using the "Clear all rulers" function from the "Edit" menu.

Ruler dragging can be canceled with the "Esc" key or using the "Cancel" function from the "Edit" menu.

Rulers can be moved by selecting "Move" mode with the speedbar buttons in the toolbar or "Move" from the "Mode" submenu in the "Edit" menu. Then left-click and drag the ruler or the ruler end point that should be changed.

Rulers can be deleted selectively by selecting a ruler in "Select" mode and pressing "Delete".

Rulers can be modified in a variety of ways. For example, rulers can be shown as arrows. To edit the properties of a ruler, double-click the ruler or select it and use "Properties" from the "Edit" menu. See Ruler Properties for a description of the properties.

(this section is inlined, next)

Multiple templates can be configured to be available for rulers. Each template defined will be shown in the "Ruler" mode toolbar button's drop-down menu. If a template is selected, new rulers produced from this template will inherit the template's properties. Templates are managed in the ruler setup page ("Setup" from the "File" menu) or "Ruler And Annotation Setup" from the "Edit" menu.

Ruler Properties

These are the properties that can be configured for rulers:

- **Labels:** depending on the outline of the ruler, up to three labels an be present. Each label can be configure individually to either show a text or the measurement values. The main label is always present, X and Y labels are only present, if the ruler has an explicit vertical or horizontal component (all outline styles except "diagonal"). For the main label the position of the label can be specified ("P" setting): the label can be made to appear on the first or the second point or in the middle of the ruler. The Alignment of the labels can be specified too: whether the appear left or right-aligned or centered.
- **Style:** the style determines how the ruler or it's components are drawn. This can be "ruler-like" (with ticks), arrow style, a plain line or with cross markers at the end.
- **Outline:** the outline determines how the two points forming the ruler are connected to render the ruler shape. This is either just one line ("diagonal"), a horizontal and a vertical line (in some outline styles combined with the diagonal line) or a box given by the two points of the ruler. A special outline is the ellipse which draws an ellipse inside the box defined by the ruler.
- **Angle constraint:** the orientation of the ruler can be restricted in several ways, i.e. just being horizontal. By default, the ruler uses the global setting. It can however be configured to provide it's own constraint.
- **Object snapping:** each ruler can be configure to snap to the closest object edge or vertex. By default, the rulers use the global setting. It may be disabled however for each ruler.
- **Mode:** in normal mode, two clicks are required to define a ruler: to set the first point and to set the second one. In "Single click" mode, a single click will set both points to the same. In "Auto measure" mode, the points will be determined by looking for edges in the vicinity of the click point and adjusting the points accordingly.

The "Label format" is an arbitrary text with embedded expressions that may represent a measurement value. Each such expression starts with a dollar sign, followed by the expression

string. The expression syntax supportis the basic operations ("*", "/", "+", "-" ..), bitwise operations ("|", "&", ..), the conditional operator ("x:y?z") as well as some functions, i.e. "abs", "sqrt", "exp". It includes a "sprintf" function. These are some examples:

- **$X**: The value of the X variable (the horizontal distance, see below for a complete list of variables).
- **$(sprintf('%.2f',X))**: The value of the 'X' variable formatted as two digit fixed precision value.
- **$(abs(X)+abs(Y))**: The manhattan distance of the ruler.
- **$min(X,Y)**: The minimum of X and Y.

A description of the expression syntax and the functions available can be found in About Expressions.

(this section is inlined at the end of the Basics section)

This is a list of all variables available:

- **D:** The length of the ruler in micron units.
- **L:** The manhattan length of the ruler in micron units.
- **U:** The x-position of the ruler's first point in micron units.
- **V:** The y-position of the ruler's first point in micron units.
- **P:** The x-position of the ruler's second point in micron units.
- **Q:** The y-position of the ruler's second point in micron units.
- **X:** The horizontal extension of the ruler in micron units.
- **Y:** The vertical extension of the ruler in micron units.
- **A:** The area enclosed by the ruler (if it was a box) in square millimeters.

Adding Images

For some applications it is necessary to show flat pixel data together with the layout. That can either be a SEM image taken or some output of a simulation tool. KLayout provides a way to add images to the display and show them below the drawn layout.

Currently, images can be read from any commonly used image format available in Qt (i.e. PNG, JPG, TIF ...). Color and monochrome images are supported. Internally an image is stored as a matrix of float values and it is possible to write custom importers using RBA.

To add an image, use the "Add Image" function from the "Edit" menu. An image property dialog will appear where the image can be specified. Choose an image using the "Browse" button next to the file name box.

An image has a variety of properties which mainly affect the way it is displayed:

- **Pixel size:** The size of one pixel in micron units. This affects the total size of the image.
- **Center:** This is the point where the center of the image is placed (in micron units).
- **Rotation:** An arbitrary angle by which the image is rotated.
- **Shear/Perspective Tilt:** shear and perspective tilt angles. Although it is possible to specify these angles explicitly it is far easiert to use the landmark adjustment feature to align an image with a layout.
- **Mirror flag:** If this option is checked, the image is mirrored at the bottom edge before it is rotated.
- **Pixel value range:** The pixel value corresponding to minimum and maximum. For normal 8 bit image formats, these values are 0 and 255. They can be adjusted which allows brighten or darken images. For float images (i.e. simulation data), this value should reflect the bounds of the output values, i.e. 0.0 and 1.0 for normalized data.
- **Color mapping:** For monochrome images, the values are converted to colors with a mapping function. The image properties page contains a tab for specifying an arbitrary mapping of data values to colors. This is achieved by placing color sample points on the data range axis and assigning colors to them. Double click at the axis to set new points, click on them to select them and adjust their color with the color box. Select and press "Del" to delete a sample point.
- **Brightness, Contrast and Gamma:** Three sliders for changing these values are provided on the respective tab.
- **RGB channel gains:** Additionally, each color channel can be weighted with a given factor on the respective tab.

Once an image is placed, it can be moved and resized using the "Move" function. The images properties can be adjusted using the "Properties" function from the Edit menu or double-clicking at the image.

With KLayout 0.22, it is possible to define landmarks which can be set at arbitrary positions in the image and aligned with corresponding layout features by dragging them to the desired target location. See Using Landmarks To Align Images for details.

An arbitrary number of images can be placed on the layout view. To store the setup, save the session using the "File/Save Session" function.

Using Landmarks To Align Images

"Landmarks" are arbitrary positions in the image. You can define such positions graphically in an image and then, within the layout view, use the defined landmarks as handles to align an image with a layout. Depending on the number of landmarks defined, you can use this feature to compensate the shift of an image, rotation, shear or even perspective distortion.

Landmarks are defined in the landmark editor. Open the landmark editor by pressing the "Define" button in the image properties dialogs.

The landmarks editor shows the image in the left panel and a list of the landmarks already defined int the right panel. You can zoom around the image view using the same methods than in KLayout's layout view (mouse wheel, middle mouse button, zoom box with right mouse button).

To add a landmark choose "Add" mode. Then click at the desired position of the new landmark. It will be shown in the image as a small target cross symbol. To delete a landmark, choose "Delete" mode and click on the landmark to delete. You can also select a landmark from the list and enter Delete mode. In that case, the currently selected landmarks are deleted.

To move a landmark, enter "Move" mode and move the landmark. Please note that the move operations follows KLayouts "pick and place" philosophy, so you have to click with the mouse twice: once to pick the landmark and second to drop it. The advantage of this approach is that you can zoom while dragging the landmark.

It does not make much sense to define more than 4 landmarks because for more the transformation derived from the landmarks is overdetermined. For best accuracy it is important to select landmark positions that span a large region. In particular, landmarks should not be too close and three landmarks should not form a line.

Dependent on the number of landmarks, KLayout can adjust the following parameters of the image:

- **1 Landmark**: displacement
- **2 Landmarks**: displacement, rotation and magnification
- **3 Landmarks**: displacement, rotation, magnification and shear
- **4 Landmarks**: displacement, rotation, magnification, shear and perspective distortion

Once the landmarks are defined, they can be used to adjust the image's display transformation. To do so, enter "Move" move in KLayout. When the mouse is over the image, the landmark symbols are shown and can be picked with the mouse. When you pick up a landmark and move the mouse, KLayout will adjust the image transformation and display the image frame according to the new transformation. When you drop the landmark, the image will be transformed accordingly.

Depending on the number of landmarks, KLayout can adjust either some or all transformation parameters such as displacement, angle etc. such that the other landmarks stay on the same position. In some cases that may not be desired. For example, if four landmarks are defined and one of them is moved, the result may be an extreme perspective distortion. Usually one would do a coarse adjustment by roughly adjusting the position and magnification and use the perspective transformation only to fine-adjust the image finally.

Hence, KLayout allows reduction of the degree of freedom it has when adjusting the transformation. While you drag a landmark, you can use these modifier keys:

- **Shift**: Only adjust displacement
- **Ctrl**: Adjust displacement, rotation and magnification

- **Shift+Ctrl**: Adjust displacement, rotation, magnification and shear

Browsing Shapes

A simple shape browser allows browsing all shapes on a layer. To do so, select the layer or layers to browse in the layer list and choose "Browse shapes" from the "Tools" menu.

A browser dialog will appear that lists the cells, shapes and cell instances. Selecting a cell will display all shapes in the cell in the middle list and the cell's instances with respect to the top cell in the right list.

If a shape is selected, the layout canvas highlights this shape by drawing a marker box around the shape and zooming to the shape. How the shape is shown can be configured on the "Configure" tab of the shape browser dialog or on the respective page in the "Setup" dialog.

Another feature that among many other things allows inspection of shapes is the "Search and replace" function (see Search and Replace in the Advanced Editing section).

(this section is also inlined, next)

Search and Replace

- Find
- Delete
- Replace
- Custom queries

KLayout offers a "search and replace" function which provides a basic search and search+replace, but also a very generic and powerful extended feature called "custom queries" (see below for that). The search and replace dialog can be found in the "Edit" menu under "Search and replace".

The dialog provides four tabs in the left panel: "Find", "Delete", "Replace" and "Custom". The functionality of these four tabs is explained below.

The left side of the dialog will hold results for operations which deliver a result, for example the "Find" operation. The result is a list which displays various items, depending on the nature and parameters of the operation. If the item represents a layout object, for example a shape or a cell instance, the items selected in the list are highlighted in the layout to indicate their position. The

length of the list is limited to avoid performance degradation for very long lists. The number of items shown can be configured on the configuration page.

The configuration page which allows configuration of the search and replace dialog's behavior is shown when the "Configure" button at the left bottom corner is pressed. It can be found as well in the setup dialog under "Browsers", "Search Result Browser".

Find

The functionality of the "Find" tab is simple: Various conditions can be specified and all objects matching that condition are listed when the "Find" button is pressed.

The parameters of that function involve: Object type, cell scope and object specific conditions. The object type is either "Instances" or a shape object. For shapes, the shape type can be confined to "Box", "Polygon", "Path" or "Text" which enables specific features.

The cell context can be one of:

- **Current cell**: look in the current cell and none else. Child cells are ignored.
- **Current cell and below**: look into the current cell and all child cells, where all instances of the children are considered.
- **All cells**: look into every cell individually, but don't consider the way the cells are instantiated.

Depending on the object type various parameters are available to be included in the condition. Each condition applies to one specific parameter and is usually composed of an operator (less, less than, equal ...) and a value against which the value of the parameter is checked. Length and area values are given in micron or square micron units.

String values can be matched against glob pattern using the tilde ("~") match and non-match ("!~") operators. Glob pattern are the ones used for file names on the command line and use "*" for an arbitrary sequence of characters and "?" for a single arbitrary character. Here are some examples for glob pattern:

A*	The string must start with a capital "A"
A	The string must contain a capital "A" somewhere
ABC?	"ABC" followed by any character
N{AND,OR}	"NAND" or "NOR"
A[0-9]	"A" followed by a digit
A[^0-9]	"A" followed by a non-digit

If the value field is left empty, no check is made on that parameter. All conditions which are checked must be fulfilled to make the object listed on the results page.

Delete

Similar to the find page, this function asks for an object type, a cell context and object specific conditions.

If the "Delete All" button is pressed, all selected objects are deleted.

If the "Select+Delete" button is pressed, the selected objects will first be shown in the result page, similar to the "Find" function. Then, some or all of them can be selected and deleted by pressing the "Delete" button below the list.

Replace

Similar to the find page, this function asks for an object type, a cell context and object specific conditions. In addition, the function allows specification of replacement values for the parameters. If an entry field is left empty for the replacement value, no replacement is made.

For strings with glob pattern matching, name parts can be reused in the replacement string. For example, if the operator is "~" on a text's string and the match string is "A(*)", the replacement string can be set to "B\1". "\1" means the value of the first bracket in the match string, hence this setup replaces all leading "A"'s by "B".

If the "Replace All" button is pressed, the replacement parameters are set on all selected objects.

If the "Select+Replace" button is pressed, the selected objects will first be shown in the result page, similar to the "Find" function. Then, some or all of them can be selected and the replacement is made on them the "Replace" button is pressed below the list.

Custom queries

The full power of this dialog is unleashed when using that page. Custom queries are not only able to provide the functionality of the three other pages (find, delete and replace), but provide functionality far beyond that simple scenarios.

Custom queries are statements resembling SQL statements with embedded expressions and a rich language to describe shape or cell instantiation details. A in-depth description can be found here: About Custom Layout Queries in the Advanced Editing section).

(this section is also inlined, next)

The custom query dialog page offers an entry field to enter the query and below an "Execute" button to run it. The most recently used queries can be pulled back into the edit field using the drop-down box below the edit field. Custom queries can be saved under a given name and

reused. The list of saved queries can be manipulated with the buttons right to it. See the button's tooltips for a description of the button's functionality.

If the tab is switched to another tab and back, the custom query will be updated reflecting the query corresponding to the current functionality selected on the other tab.

About Custom Layout Queries

Layout queries are an advanced feature of KLayout which provides a very generic method to manipulate or search the geometrical or cell information of a layout. The basic concept of custom queries is borrowed from SQL, the language widely used for accessing databases. Instead of working on linear tables with rows and columns, KLayout's queries work on the layout structure which basically a cell tree, a layer set orthogonal to that and per-cell/per-layer geometrical information which itself is divided into shape classes.

Layout queries have a layered structure, like an onion: the core of the query is a cell query which selects one or many cells with or without their children. The cell query can be wrapped by a shape query which addresses the shapes of the selected cells or instances, optionally confined to specific layers. The last layer is formed by the action: this is the activity that KLayout will perform on the selected objects. The default action is simply to report the results. It is also possible to delete the selected objects or to perform a custom operation on them.

On cell and shape level, conditions can be specified which will restrict the operation on a subset of the selected objects. Reports can be sorted by a arbitrary key derived from the current object.

On cell level three different relationship models are supported:

- Individual cells: no hierarchy is involved
- The cell tree: the cell tree is the parent/child relationship without explicit instantiation information
- The cell instance tree: every individual instance of a cell is considered

Expressions play an important part in layout queries, both as actions (assignment type expressions) as well as conditions or for the derivation of sorting keys. See About Expressions for details about expressions. Within expressions, RBA objects are used to represent shapes or instances (see Class Index for a list of classes available). Depending on the context of the query, a variety of functions is available to access the properties of the current item and context.

The key to layout queries is the "Search and Replace" feature: this dialog uses custom queries to emulate simple search and replace functionality on the first three tabs of the dialog ("Delete" is regarded as a special kind of "replacement" here). However, the true power is revealed on the forth page: here you can enter all kind of custom queries. Clicking on "Execute" will run the query and display the results in the right panel.

If a search or replace action is specified on the first three tabs, the corresponding custom layout query will be shown in the entry box of the forth one. That way it is very easy to create a first query using the standard functions, switch to the custom query page and adjust it to fit the specific requirements.

Building queries: cells

The very core of a query is a cell expression. The most simple form of a cell expression is a simple cell name. This expression will select the cell called "RINGO":

```
RINGO
```

Cell expressions can contain wildcard in the "glob" form made popular by the Unix and Windows command line. "*" is for an arbitrary sequence of zero to many characters, "?" for any single character. "{A,B,C}" is for either the character sequence "A", "B" or "C", "[ABC]" is any of the characters "A", "B" or "C" and "[^ABC]" is for any character not "A", "B" or "C". Round brackets can be used to group parts of the string for later reference. If brackets of any kind are used inside a match string, either single or double quotes should be used around match strings in order to avoid ambiguities with other parts of the query syntax.

This expression will select all cells starting with "T":

```
T*
```

Although it is not necessary to do so, it is recommended to mark a cell query explicitly as such using the optional "cells" or "cell" keyword. This query has the same effect than the previous one but is somewhat more robust if used in nested queries we will learn about later:

```
cells T*
```

A cell expression can already be used by it's own. The report of such a query will simply contain the cells selected by that expression. If combined with an action, such expressions already provide useful manipulation functionality.

The "delete ..." action will delete the given cells:

```
delete cells T*
```

The "with .. do ..." action can be used to manipulate the cell. This example will rename the cell by replacing the "T" prefix with a "S". The part after "do" is an expression which is evaluated for each hit. Note that "$1" is used to refer to the first matching bracket in the last match. "cell.name" is a method call on the object "cell" which is provided by the query in the context of a cell query. "cell" is a "Cell" object (see Cell) and setting the name will basically rename the cell. The expression used for the assignment will put an "S" in front of the rear part of the name, hence replace "T" by "S":

```
with cells "T(*)" do cell.name = "S"+$1
```

Note the quotes around the "T(*)" match expression. They are necessary to make the brackets part of the match expression rather than the cell query. It is usually a good idea to put the match expression inside quotes to avoid ambiguities.

The last example already demonstrates how a combination of two simple concepts - simple cell queries and expressions - form a new and very generic feature. We will soon learn about the power of that concept.

Building queries: cell trees

Cell queries are the most simple form of core queries. The next level is entered by extending the concept to hierarchies. Cell hierarchies come on two flavors: a parent-to-child relationship tree (the cell tree) and the instantiation tree. In the cell tree, each cell is at most present once in the context of a parent cell, independent of the number of times a cell is used inside a parent cell. The cell tree just describes the fact that a cell is a child cell of another, not how the cell is used. The instance tree adds this detailed information as well: how many times a cell is used and what transformations are applied per such instance.

The cell tree can be accessed within cell queries using the "." operator to separate parent and child cell. The following cell query returns all cells which are children of a cell "A":

```
cells A.*
```

Multiple levels may be nested, for example the following query lists all cells which are second-level children of the "A" cell:

```
cells A.*.*
```

Such expressions form a "path" leading from an initial cell to some cell, which is returned by the query. The "." separates the path elements like the slash or backslash does in a file path.

Top cells can be addressed by a leading "." similar to the leading slash of an absolute file path in Unix. The following query will return all top cells:

```
cells .*
```

Brackets can be used to group parts of the path. That has no immediate effect, but it can be usedful in combination with quantifiers and branches as we will see soon. The following queries are equivalent:

```
cells TOP.*.A
cells TOP(.*.A)
cells TOP(.*)(.A)
```

Please note that brackets can only be put between the dot and the previous element. A query like "cells TOP.(*.A)" is invalid since the opening bracket is after the dot.

Alternative paths can be specified by separating them with a comma. Such alternatives must be put inside brackets. For example, this query selects all cells that are children of "TOP" and start with an "A" or which are second-level children and start with an "E":

```
cells TOP(.A*,.*.E*)
```

Path elements can be made optional with a "?" symbol and expanded an variable number of times using quantifiers like "*" (0 to many) and "+" (one to many) or "{n,m}" (n to m times). Note that you'll have to put the expression subject to the quantifier in brackets in order to avoid ambiguities of the star operator. The following expression will return the A cell in every possible child context of "TOP", i.e. as direct child, second-level child and so on:

```
cells TOP(.*)*.A
```

To understand that query, note that the "*" inside the brackets is forming the match string while the outer star is forming the quantifier. That query reads in expanded form "TOP.A", "TOP.*.A", "TOP.*.*.A" etc.

There is a useful abbreviation for the above case. The following query will also produce "A" in every child context of "TOP":

```
cells TOP..A
```

The double dot operator matches an arbitrary part of the instantiation path before and after a cell even without being anchored at one end. Used before a cell name, it will return all contexts a cell is used in (including top cells and all child contexts). Used after a cell, it will return the cell plus all child cells in each possible context. Used before a cell it will deliver all contexts that cell is used in every top cells. The following query will deliver "TOP" plus all it's direct and indirect children:

```
cells TOP..
```

Note that the previous query may deliver the same cell multiple times - once for each context (call path from TOP) it is used in. Hence "TOP.." will basically expand into the cell tree with "TOP" as the root.

In order to get the names of all cells called from a given cell, you can use the "select" action with the cell name and the "sorted by .. unique" output selector to remove duplicates of cell names:

```
select cell_name of cells TOP.. sorted by cell_name unique
```

See below for a description of the "select" action.

Within a path, dynamically computed components can be inserted using the "$(..)" notation which wraps an expression. That expression is evaluated in the context of the previous path component. For example, the following query selects all child cells which are named like their parent with an "A" prefix:

```
cells *.$("A"+cell_name)
```

Building queries: instances

Cell trees can be expanded into instance trees simply by prepending "instances". This will deliver all direct instances of "TOP":

```
instances of TOP.*
```

When asking for instances, more information is available inside the query. For example, the instance's orientation and position is available. With the "instances" specification, array references are expanded into single instances. To keep arrays as such, use "arrays" instead of "instances":

```
arrays of TOP.*
```

Cell or instance queries can be filtered using the "where" clause. After the "where" an expression is expected with a number of predefined variables that reflect the context (see below for the variables available). The following query selects all child cells of "A" where the cell name has a length of 5 characters:

```
cells A.* where len(cell_name)==5
```

Building queries: shapes

So far we have dealt with cells and their instantiations. We enter the next level now by introducing shapes.

Shape queries are built atop of the cell/instance level. A simple example selects all shapes of the cell "TOP":

```
shapes of cell TOP
```

Shape queries can be confined to certain shape types. For example, this confines the query to boxes:

```
boxes from cell TOP
```

Allowed shape type are "boxes", "polygons", "texts" and "paths". In the context of a shape query additional variables are available for expressions. The most important one is "shape" which is a Shape object (see Shape) That objects provides access to the shape addressed in a generic way. Specialization to a specific shape type is possible through the shape specific accessor methods (i.e. "shape.box_width") or the specific objects (i.e. "shape.box").

Multiple shape types can be given with "or" or a comma:

```
boxes or polygons from cell TOP
```

Shape queries can be confined to certain layers. This query will report all shapes from layer 8, datatype 0:

```
shapes on layer 8/0 from cell TOP
```

Intervals can be specified with the hyphen ("-") and multiple layers or intervals can be listed with a comma or semicolon. The following will list the shapes from layer 8, datatype 0 to 10 and layer 9, datatype 0 only (note that "no datatype" is interpreted as datatype 0):

```
shapes on layer 8/0-10, 9 from cell TOP
```

For formats that support named layers only (i.e. DXF), the layer name can be given. The following query lists shapes from layers METAL and POLY (case sensitive!):

```
shapes on layer METAL, POLY from cell TOP
```

Any kind of cell query can be used inside the shape query. If a cell query renders multiple cells, the shape query will be applied to each of the cells returned. If instances are selected by the cell query, the shapes will be reported for each instance. Since the cumulated transformation of a specific instance into the top cell is available through the "path_trans" variable, it is possible to transform each shape into the top cell in the instance case. The following expression combines a "with .. do" action with a shape query to flatten all shapes below "TOP":

```
with shapes on layer 6 from instances of TOP.. do
initial_cell.shapes(<10/0>).insert(shape).transform(path_trans)
```

That expression reads all shapes of cell "TOP" and it's children, inserts them into a new layer 10, datatype 0 and transforms the shape after it has been inserted. This expression makes use of the variables "initial_cell" (a Cell object representing the root cell of the cell query), "shape" (a pointer to the currently selected shape and "path_trans" (a Trans object representing the transformation of the current shape into the root cell of the query). It also employs the angle bracket layer constant notation which specifies a layer in the target notation and can be used in place of the layer index value usually used inside the API. Note that the target layer must exist already, i.e. must have been created in "Edit/Layer/New Layer" for example.

Shape queries can be confined with conditions. A condition is entered with a "where" clause plus an expression that selects the shapes. This condition selects shapes with an area of more than 4 square micron (note that the "um2" unit must be given, since it will cause the value to be converted into the database units used internally):

```
shapes from cell TOP where shape.area < 4 um2
```

Shape conditions can be combined with cell conditions. To avoid ambiguities, the cell query must be put into brackets in that case:

```
shapes from (cells * where len(cell_name)==4) where shape.area < 4 um2
```

Actions

Actions specify operations that are to be performed on the results of a query. The default action is to just list the results. In the "Search and replace" dialog, the results will be listed right to the query entry box as a table. Depending on the context of the query, cell names, cell names plus parent cell, cell instances or shapes are listed.

"select" action

The "select" action will compute one or more results from each item returned by the query and present the computed value in a table. The general form is:

`select` expr1, expr2, ... **`from`** query

"expr1", "expr2" ... are expressions. For example this action computes area and perimeter for all shapes of cell "TOP":

```
select shape.area, shape.perimeter from shapes of cell TOP
```

The "select" action offers sorting with optional reduction to unique values:

`select` expr1, expr2, ... **`from`** query **`sorted by`** sort_key

`select` expr1, expr2, ... **`from`** query **`sorted by`** sort_key **`unique`**

Here "sort_key" is an expression which delivers the value by which the output will be sorted. If "unique" is specified, items with identical sort key are reduced to a single output.

"with" action

The "with" action executes an expression on each item returned by the expression. In that sense it is basically equivalent to the "select" action but the results of the operation are discarded and the intention of the expression is to modify the results. The general form of that action is this:

`with` query **`do`** expr

For example, this action will move all shapes of cell "TOP" from layer 6 to layer 10, datatype 0:

```
with shapes on layer 6 of cell TOP do shape.layer = <10/0>
```

"delete" action

This action will simply delete the objects selected by the query:

`delete` query

For example, this query deletes all shapes from layer 6, datatype 0 on cell TOP:

```
delete shapes on layer 6 of cell TOP
```

Variables available per context

Common variables

The following variables are available in all queries:

Name	Value type	Description
layout	Layout	The layout object that this query runs on.

Cell query context

In the plain cell and cell tree context, the following variables are available:

Name	Value type	Description
path	Array	Array with the indexes of the cells in that path. For a plain cell, this array will have length 1 and contain the index of the selected cell only.
path_names	Array	Array with the names of the cells in that path. For a plain cell, this array will have length 1 and contain the name of the selected cell only.
initial_cell	Cell	Object representing the initial cell (first of path)
initial_cell_index	Integer	Index of initial cell (first of path)
initial_cell_name	String	Name of initial cell (first of path)
cell	Cell	Object representing the current cell (last of path)
cell_index	Integer	Index of current cell (last of path)
cell_name	String	Name of current cell (last of path)
hier_levels	Integer	Number of hierarchy levels in path (length of path - 1)
references	Integer	The number of instances of this cell in the parent cell. Array references count as 1. For plain cells, this value is 0.
weight	Integer	The number of instances of this cell in the parent cell. Array references count as multiple instances. For plain cells, this value is 0.
tot_weight	Integer	The number of instances of this cell in the initial cell along the given path. Array references count as multiple instances. for plain cells, this value is 0.
instances	Integer	Equivalent to "weight", but also available for plain cells. For plain

	cells, the value represents the number of times, the cell is used in all top cells.
bbox	The cell's bounding box.
cell_bbox	Same as "bbox" (disambiguator for shape and instance bounding boxes).

Instance query context

In an instance query context, the properties of the current instance are available as variables in addition to most of the ones provided by the cell query context. These variables are not available in instance context: "weight", "references" and "tot_weight". Apart from that these additional variable are provided:

Name	Value type	Description
path_trans	ICplxTrans	The transformation of that instance into the top cell. For a plain cell that is a unit transformation.
trans	ICplxTrans	The transformation of that instance (first instance if an array).
inst_bbox	Box	The instance bounding box in the initial cell.
inst	Instance	The instance object of the current instance.
array_a	Point	The a vector for an array instance or nil if the instance is not an array.
array_na	Integer	The a axis array dimension or nil if the instance is not an array.
array_b	Point	The b vector for an array instance or nil if the instance is not an array.
array_nb	Integer	The b axis array dimension or nil if the instance is not an array.
array_ia	Integer	The a index when an array is iterated (0 to array_na). Not available with instance queries with "arrays of ...".
array_ib	Integer	he b index when an array is iterated (0 to array_nb). Not available with instance queries with "arrays of ...".

Shape query context

In the context of the shape query, the following variables are available in addition to the variables made available by the inner cell query. The inner cell query is either a instance query or a cell query:

Name	Value type	Description
bbox	Box	The shape's bounding box
shape_bbox	Box	Same as "bbox" (disambiguator for cell or instance bounding boxes)
shape	Shape	The shape object
layer_info	LayerInfo	The layer description of the current shape's layer
layer_index	Integer	The layer index of the current shape

Browsing Instances

All instances of a cell can be browsed by selecting the cell in the cell list (not making it the new top), and choosing "Browse instances" from the "Tools" menu. A simple instance browser comes up that shows all cells that the given cell is instantiated in and how the cell is instantiated.

Another feature that among many other things allows inspection of instances is the "Search and replace" function (see Search and Replace).

The Marker Browser

KLayout offers a generic concept of storing error markers or related information. This concept is called the "Report database" (RDB). An arbitrary number of report databases can be associated with a layout view. Usually, each database refers to a certain layout but that is not a strict requirement.

A report database primarily is a generic collection of "values", which can be strings or other items. Usually, a value is a collection of geometrical objects which somehow flag some position or drawn geometry. Multiple of such values comprise a "marker item". The report database associates these marker items with additional information:

- **Tags:** Flags that indicate certain conditions. The marker browser uses a couple of predefined tags like "important", "waived" and "visited" which can be set or reset by the user indicating whether a marker item is considered important or an error has been waived.
- **Image:** A marker can be assigned a screenshot image which serves for documentation purposes.

Marker items are organized into categories. Each marker item must be associated with a category. Categories themselves can be organized hierarchically, i.e. categories can be split into sub-categories. This offers a way of improving the organisation of such categories.

Marker items are usually associated with a cell, i.e. where a error was detected. By default, a marker item is simply associated with the top cell.

The report database uses a proprietary format based on XML which is capable of storing the annotations provided by the database. It is possible however to import Calibre DRC ASCII format files.

The marker browser is a tool to browse a report database associated with a view. The marker browser can be started using the "Marker Browser" function in the "Tools" menu. The marker browser tracks whether a marker has already been visited similar to the "read" flag in a mail

client. This allows tracking of a review session. The "visited" state is reflected in the database file.

In the marker browser, use the "Open" button to load a XML database file or import files from other formats. Choose "Reload" to reload a file and "Save As" to write a database in XML format.

The marker browser offers three panels:

- **Directory:** This panel lists the categories and cells of the database. Categories or cells with unvisited markers will be shown in bold font. Such with no markers at all are shown in green color. It is possible to suppress these categories or cells by deselecting "Show All" in the directory's context menu. To have the lists sorted by marker count, click at the header of the count column.
  Multiple categories or cells can be selected. In that case, the markers panel will show the markers of all selected cells or categories.
- **Markers:** This panel lists the markers in the selected category and/or cell. A length of the list is limited and can be changed on the configuration page ("Configure" button on the marker browser or in the setup dialog). Various tags are shown in this panel as well. The list can be sorted in various ways by clicking at the respective header.
  When a marker is selected in this list, it will be highlighted in the layout, provided a suitable layout is associated. The way a marker is highlighted and how the view is adjusted can be specified on the configuration page.
- **Info:** This panel summarizes the information for the selected marker. If a screenshot was associated with the marker it is shown here. Click on the thumbnail image to show it in a separate window in the original size.

Similar to the shape and instance browsers, the marker browser offers navigation buttons to select the next marker, category or cell.

The marker browser supports "tagged values": each marker can be associated with a value that has a name. Such values can be imported from RVE files where they are called properties. Tagged values are generated by certain generators and can represent measurement values or similar. When tagged values are present, they will be shown in the markers list. The markers can be sorted by these values by clicking on the appropriate header.

Technology Management

A new feature of KLayout 0.22 is technology management. Technology management summarizes features which require a certain interpretation of a layout. In particular, layout layers are assigned a physical meaning, for example via layers or active area layers in CMOS technologies. Since that interpretation often is depending on the technology the product will be

fabricated with, the ability to provide multiple setups is summarized as "technology management".

The "Technology Manager" in the "Tools" menu is the user interface that allows creating, deleting and editing of technology setups. Read more about the technology manager in About Technology Management

(this section is also inlined, next)

About Technology Management

Technology management summarizes features which require a certain interpretation of a layout. In particular, layout layers are assigned a physical meaning, for example via layers or active area layers in CMOS technologies. Since that interpretation often is depending on the technology the product will be fabricated with, the ability to provide multiple setups is summarized as "technology management".

A technology setup implements the following aspects:

- Layer mapping: when the layout reader loads a file that for a certain technology, it can apply a layer mapping, i.e. apply layer names to GDS layers, filter layer etc.
- Layer properties: depending on the technology, the layer display can be configured by providing a technology specific layer properties file.
- Connectivity: the layer stack and the connections made by layers for the net tracer feature.
- Macros: macros associated with the technology. When the corresponding technology is selected (is the one of the current layout), such macros will show up in the menu if they are bound to a menu entry. Otherwise they will be invisible.
- DRC scripts: in the same way, DRC scripts can be associated with a technology.
- Libraries: if a library is associated with a technology, it is shown in the list of available libraries when an instance is created. Library association cannot be edited. Instead, a library installed in the system comes with a technology association itself.
- File format options: technology specific file reader or writer options can be given. When a layout is saved, it will use the writer options from it's technology. When loading a layout, the reader options from the active technology will be used.

In the future, more aspects may be added to the technology definition.

There is always one "Default" technology that is used when no technology is specified.

Setting up technologies

Technologies can be set up using the "Technology Manager" in the "Tools" menu. There is always a "Default" technology which provides the settings when no technology is selected. New

technologies can be added or technologies can be deleted using the "+" or "x" buttons below the technology tree.

A technology has a name (a short string) and a description. The name is used to identify the technology in various places. The description is the human-readable text that is displayed in the technology selection boxes for example. The short name can be changed by selecting the technology and pressing the "Rename" button or using "Rename" from the technology tree's context menu (right mouse click). The description can be edited on the "General" page.

In the technology manager, below each technology, the components are shown that define the various aspects of a technology. Beside the "General" aspect (names, descriptions) there is a "Layers" component which defines the layer mapping table and layer properties file and the "Connectivity" component which defines the settings for the net tracer.

Using technologies

When more than the default technology is defined, KLayout provides a drop-down menu in the tool bar to select the current technology. The current technology is the technology used when new files are loaded. It is also possible to define the technology to be used on the command line using the "-n" switch (applies to following files and specifies the technology to use by their short name).

The technology of the currently selected layout is shown in the status bar of the main window in the left section. It is possible to switch the technology of a layout already loaded by using the "Layout Properties" dialog from the "File" menu. After switching the technology, the layer properties defined in the technology can be applied and the associated macros or DRC scripts are shown in the menu if they are associated bound to a menu entry or the key binding becomes active if a shortcut is defined for that macro.

Technologies and macros or DRC scripts

Macros or DRC scripts are stored in sub-folders relative to the technology's base path. When no base path is specified or the base path is invalid, macros or DRC scripts cannot be associated. KLayout will look search for macros, if a directory called "macros" is present in the base path. If it finds files with a valid macro suffix there it will associate them with this technology. The same way, KLayout will look for DRC scripts, if a directory called "drc" is present in the base path.

Macros and DRC scripts associated with a technology are shown in the technology manager. To edit or debug scripts of macros, use the macro development environment ("Macros/Macro Development"). If a technology has a macros or DRC scripts folder, the macro or DRC scripts tree in the development environment will show a corresponding top-level branch for that technology.

Multiple technologies can share the same base path - hence it is possible to share macros or DRC scripts between technologies.

Managing technologies

Technologies can be imported and exported to technology files (suffix ".lyt"). This is mainly useful to exchange technology settings between users or technologies.

Except for the default technology, technologies are kept in technology folders in KLayout's application path. They are read from subfolders from the "tech" directories. The technology definition itself is held in a file with extension ".lyt". The technology folder may have subfolders to hold library files, macros, DRC runsets, LEF files and other pieces of the technology package.

Technologies can be managed using packages. Packages are a convenient way to share add-ons between users. Packages can be installed from a common repository and allow easy addition and removal of components. Technologies are one aspect of packages, so it's possible to create packages that contribute one or more technologies. See About Packages for more details.

(this section is also inlined, next)

About Packages

"Salt" is KLayout's package manager which allows selecting and installing packages from a global repository. Packages make KLayout more tasty. Packages (the "grains") may cover a variety of features:

- Ruby or Python macros
- DRC runsets
- Technologies
- Fonts for the Basic.TEXT PCell
- Static layout libraries
- PCell libraries
- Code libraries for Ruby and Python
- Binary extensions

Packages can depend on other packages - these are installed automatically if a package requires them and they are not installed yet.

Packages are identified by name. A package name needs to be unique in the package universe. You can use a prefixed name like `www.mydomain.org/nameofpackage` to create a non-ambiguous name. Use a slash to separate the prefix from the actual package name. The choice of the prefix is entirely up to you as long as it contains letters, digits, underscores, hypthens or dots. You can use a domain name that is owned by yourself for example. You can use multiple prefixes to further differentiate the packages inside your namespace.

Packages also come with version information, so KLayout can check for updates and install them if required. KLayout will assume strict upward compatibility. This specifically applies to packages that other packages are depending on (such as code libraries). If you need to change them in a non-backward compatible way, you'd need to provide a new package with a different name.

Packages come with some meta data such as authoring information, an optional icon and screen shot image, license information and more. The more information you provide, the more useful a package will become.

The key component for public package deployment is the "Salt.Mine" package repository service. This is a web service that maintains a package index. It does not host the packages, but stores links to the actual hosting site. In order to author a package, you need to upload the package to one of the supported host sites and register your package on the Salt.Mine page. Registration is a simple process and the only information required is the link to your host site and a mail account for confirmation.

Installing Packages

To install external packages, open the package manager with "Tools/Manage Packages". On the "Install New Packages" page, a list of available packages is shown. Select the desired packages and mark them using the check mark button. Marked packages will be downloaded and installed with the "Apply" button.

A filter above the package list allows selecting packages by name. The right panel shows details about the package currently selected.

Updating Packages

To check for updates, use the "Update Packages" tab of the package manager. In the list, those packages for which updates are available are shown. Mark packages for update using the check mark button. Click "Apply" to apply the selected updates.

Uninstalling Packages

To uninstall packages, open the package manager using "Tools/Manage Packages". Go to the "Current Packages" tab. Select a package and use the "Remove Package" button to uninstall the package.

Creating Packages

For package development you can utilize KLayout to initialize and edit the files inside the package folder or populate the folder manually.

KLayout offers initialization of new packages from templates. You can modify that package according to your requirements afterwards. To create a package from a template, open the package manager using "Tools/Manage Packages", go to the "Current Packages" tab and push the "Create (Edit) Package" button. Chose a template from the list that opens and enter a package name (with prefix, if desired). Select "Ok" to let KLayout create a new package based on the template you selected.

The package details can be edited with the "pen" button at the top right of the right details panel. Please specify at least some author information, a license model and a version. If the package needs other packages, the dependencies can be listed in the "Depends on" table. Those packages will be automatically installed together with the new package. The showcase image can be a screen shot that gives some idea what the package will do. The package details are kept in a file called "grain.xml" inside the package folder. You can also edit this file manually. The "grain.xml" is the basic description file for the package.

If the package is a macro or static library package, the macro editor can be used to edit the package files. If the package is a tech package, the technology manager can be used to edit the technology inside the package. To populate the package folder with other files use your favorite editor of KLayout itself for layout files.

Deployment inside your organisation

Once a package is finished, it needs to be deployed to make it available to other users. Deployment basically means to put it on some public place where others can download the package. For local deployment inside an organisation, this can be a web server or a folder on a file server. KLayout talks WebDAV, so the web server needs to offer WebDAV access. A subversion (SVN) server provides WebDAV by default, so this is a good choice. Git can be used too, but you'll need to mirror the Git repository to a file system or WebDAV share.

After a package has been made available for download, it needs to be entered in the package index. For local deployment, the index can be a file hosted on a web server or on the file system. The package index location needs to be specified by the `KLAYOUT_SALT_MINE` environment variable which contains the download URL of the package index file.

For public deployment, the Salt.Mine service (`http://sami.klayout.org`) is used to register new packages in the package index. By default, KLayout loads the package index from that service, so once your package is registered there, everyone using KLayout will see it.

The Package Index

Public Packages are published on the Salt.Mine server. This is a web service that delivers a packages index with some meta data such as current version, the icon and a brief description. KLayout uses this list to inform users of packages available for installation and available updates. For local deployment, the package index can be served by other ways too. The only requirement is to be accessible by a http, https or file URL.

The basic format of the index is XML with this structure:

```
<salt-mine>
  <salt-grain>
    <name>name</name>
    <version>Version</version>
    <title>Title of the package</title>
    <doc>A brief description</doc>
    <doc-url>Documentation URL</doc-url>
    <url>Download URL</url>
    <license>License model</license>
    <icon>Icon image: base64-encoded, 64x64 max, PNG preferred</icon>
  </salt-grain>
  ...
  <include>URL to include other index files into this one</include>
  ...
</salt-mine>
```

You can include other repositories - specifically the default one - into a custom XML file. This allows extending the public index with local packages.

When the package manager is opened, KLayout will download the index from `http://sami.klayout.org/repository.xml`. You can set the `KLAYOUT_SALT_MINE` environment variable to a different URL which makes KLayout use another dictionary service, i.e. one inside your own organisation. This service can be any HTTP server that delivers a package list in the same format than the Salt.Mine package service. The URL can also be a "file:" scheme URL. In this case, KLayout will download the list from the given file location.

When installing a package, KLayout will simply download the files from the URL given in the package list. KLayout employs the WebDAV protocol to download the files. This protocol is spoken by Subversion and GitHub with the subversion bridge. The latter requires a simple translation of the original Git URL's to obtain the subversion equivalent.

Selecting Rulers, Shapes Or Instances

Rulers, images, shapes or instances can be selected by either clicking on the shape, instance, ruler or image in "Select" mode or by dragging a selection rectangle with the left mouse button pressed. In this case, all objects inside the selection rectangle will be selected.

Pressing the Shift key in addition to selecting shapes, instances or other objects will extend the current selection. Pressing Ctrl key will remove the given objects from the selection.

For layout objects (shapes and instances), it is possible to select the objects through the hierarchy or on the level of the current cell only (top level objects only). By default, objects are selected through the hierarchy. That means, objects in child cells can be selected. By extending the

selection over multiple instances of a cell, the same object can be selected multiple times. While editing objects inside a child cell may be a desirable feature, this can lead to confusing effects in the end. Hence, it may to more intuitive to only work on top level, i.e. on the current cell. To enable this mode, check the "Select Top Level Objects Only" menu item in the "View" menu or the same in the edit mode options dialog (F3).

In top-level only mode, instances of cells are selected only if the cells have a visual appearance. This is the case, if they contain at least one shape on a visible layer for example. If they are empty with respect to the visible layers, they cannot be selected. If the cell's box is shown, because the hierarchy levels shown are confined a sufficiently small interval, the cell instances will always be selected.

Images and rulers are simple objects not embedded in a hierarchy and can be selected by clicking at them or enclosing them in the selection box.

Object properties can be inspected or edited (if allowed) by opening the properties dialog. To edit or inspect the properties of selected objects, choose "Properties" from the "Edit" menu. A dialog will open which shows the properties for the first selected object. To proceed to the next object push the "Next" button, to go back to the previous object push the "Previous" button.

Depending on the application mode, the properties of the selected objects may be edited too. To apply changes to the current object, choose "Apply". To apply the changes and close the dialog, choose "Ok". To apply the changes to the current object and all other selected objects of the same kind, choose "Apply To All" (if applicable). To close the dialog without applying any changes, choose "Cancel".

"Apply To All" will try to apply the changes in some smart way to other objects. For example, for boxes, and change of the box dimensions will be applied in the same way (same shift in the four directions) than for the current object. If the width of a path has been changed, the same width will be applied to all other paths and so forth.

For shapes and instances, the "User properties" can be edited too. "User properties" are arbitrary properties attached to shapes and instances. They consist of a key (preferably a number for GDS2 compatibility) and a value (preferably a string in GDS2). Multiple properties of that kind can be attached to one object. User properties can be edited by pushing the "User Properties" button which will open a dialog that allows editing of the properties. If this dialog is closed with "Ok", the user properties will be kept, but applied to the layout object only after pressing "Apply"/"Ok" or "Apply To All". The latter will apply the new properties to all other shapes of the same kind.

Finally, for shapes and instances, the "Instantiation" button will show the instantiation path of the shapes or instances.

The layer of a shape cannot be changed through the properties dialog. To move a shape to a different layer, use "Change Layer" from the "Selection" sub-menu in "Edit".

More Configuration Options

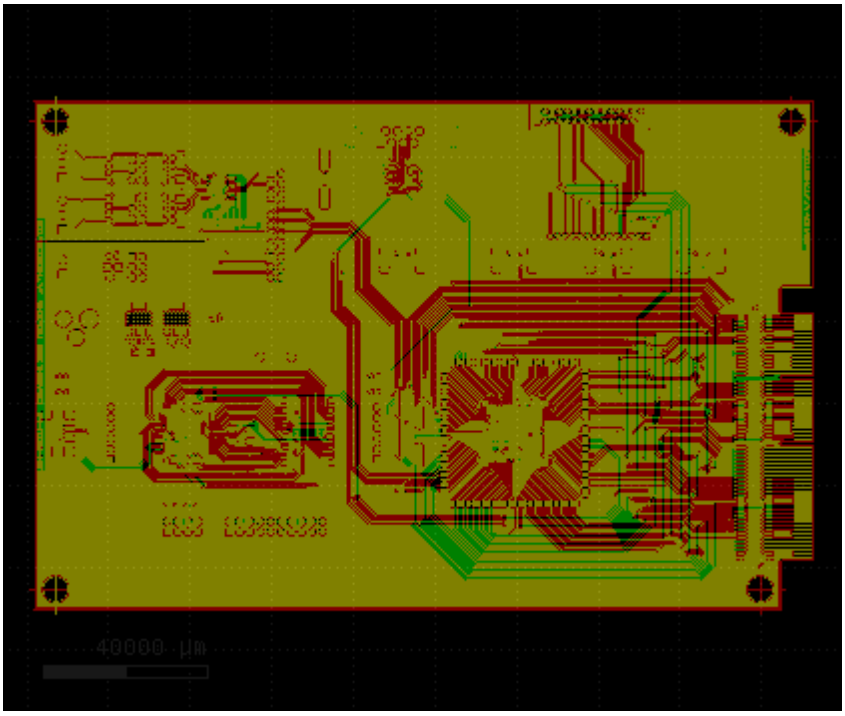
The option dialog available with the "Setup" function from the "File" menu offers numerous configuration options from background to rulers configuration.

In this dialog for example, the color palette can be edited, so that different colors are available or the stipple palette can be configured. In addition, it is possible to define the order how these colors or stipples are assigned to layers initially and which colors are not used for layer coloring.

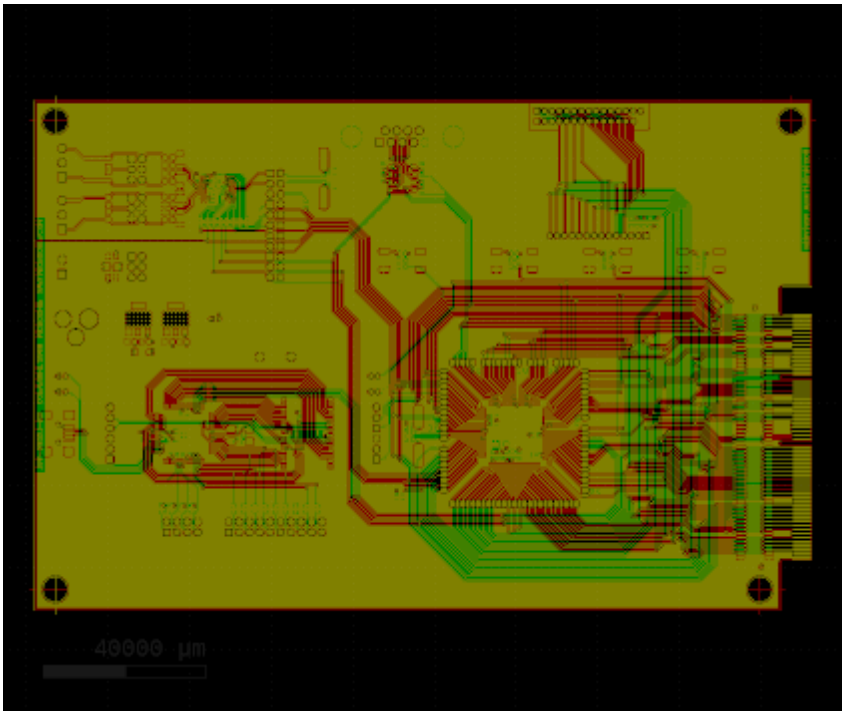
A particular useful feature is the oversampling scheme. Oversampling is provided as an option to enhance the image quality. The image is rendered at a higher resolution and then downsampled to the screen resolution. In effect, lines appear thinner and more details can be resolved. As a negative side effect currently the stipple pattern becomes finer and the crosses in marker mode are smaller. On the other hand the resolution effect can be quite impressive.

Oversampling can be enabled on the "Display/General" page in the setup dialog. 2x and 3x oversampling is provided. The following screenshots illustrate the effect of oversampling:

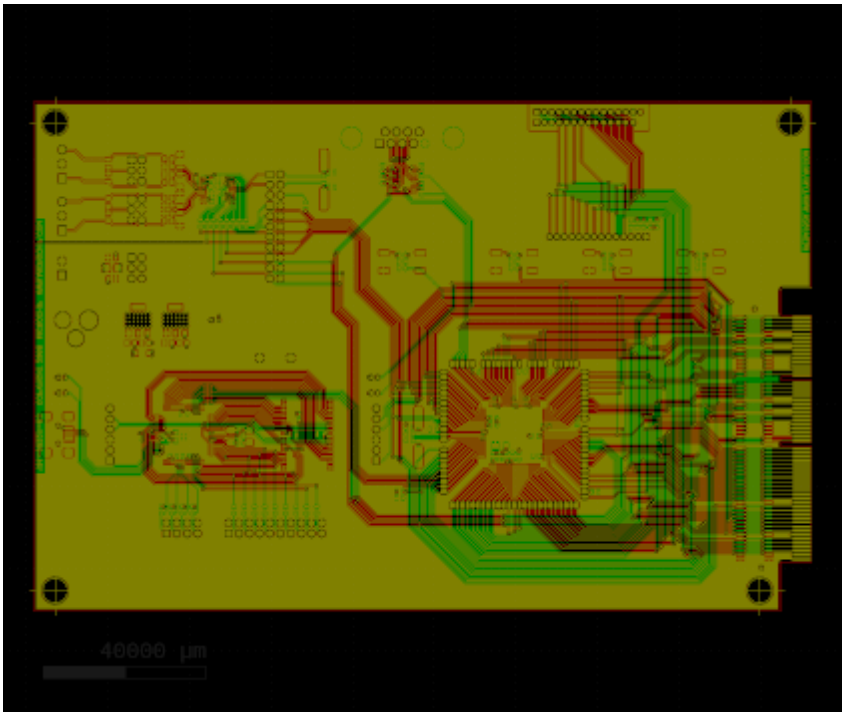
Normal (1x)



2x oversampling



3x oversampling



Undo And Redo

Most operations such as changing of layer colors can be undone using the "Undo" function from the "Edit" menu. Analogous, the operations can be redone again using the "Redo" function from the "Edit" menu.

Saving A Layout Or Parts Of It

A layout or a subcell of it can be saved to either GDS2 or OASIS. To save a layout, choose "Save As" from the "File" menu. To save just a cell, select the cell in the cell tree (it does not need to be the currently shown one) and select "Save Current Cell As" from the context menu (right mouse button) of the cell tree.

A file dialog will pop up to select the file name to which to write the cell or layout. After a file has been selected, an option dialog will be shown to specify further options. In this dialog, it is possible to constrain saving to a subset of layers, i.e. just visible ones. Also, the database unit can be changed or the layout can be scaled by a given factor.

For OASIS, a compression level can be specified. At a level of 0, no particular attempt is made to compress shapes. At higher levels, shapes are classified and array compression is tried. The higher the level, the more attempts are made to compress shapes into arrays. In particular for flat layouts, compression of shapes requires some memory and slows down OASIS writing considerably.

Saving And Restoring A Session

A session can be saved and restored later. A "session" involves the files loaded, bookmarks, annotations, layer settings, hierarchy settings, images and application setup. Sessions are stored as XML files with the suffix ".lys".

To save a session, choose "Save Session" from the "File" menu. To restore a session, choose "Restore Session". KLayout can be started with a certain session using the "-u" option from the command line followed by the session file. On Windows installations, session files are registered as being opened automatically by KLayout.

Further View Options

The "View" menu offers some more options to tune the display:

- **Show Grid**: This is a fast access method for the respective configuration option. Unchecking this menu entry hides the grid lines and the scale bar.
- **Grids**: this submenu allows fast access to the current grid setting. You can choose from one of the default grids that are configured on the "Default Grids" page in the "Application" section of the setup dialog. Grids are mainly used for editing geometry but also for snapping rulers if they are configured to do so.
- **Show Texts**: when this option is unchecked, texts are not drawn. That applies to text objects and user properties. Drawing is somewhat faster when texts are not drawn.
- **Show Cell Frames**: when this option is unchecked, cell frames from hidden cells or cell below the hierarchy levels selected are not drawn.
- **Show Layers Without Fill**: when this option is checked, the shapes are drawn without fill. Hence the layout is shown in a kind of wire frame. This option is intended to temporarily disabling the fill pattern. Keeping this option checked may be confusing because setting a fill pattern may not have any effect then.
- **Synchronized Views**: with this option, all views (tabs) in the main window are synchronized, i.e. they show the same region of the layout. This option can be useful if two layouts are loaded in different views for clarity, but identical regions need to be inspected.
- **Select Top Level Objects**: this menu item gives a quick access to the respective editor option. If this option is checked, only objects from the top level are selected. This is in particular important for editing because editing a subcell can cause effects in other places there that cell is instantiated but no edit is intended.
- **Toolbar**, **Navigator**, ...: disable or enable the respective tool windows.
- **Highlight Object Under Mouse**: with this option checked, the object under the mouse is highlighted temporarily if the mouse hovers over that object. Since that can be confusing in some (rare) cases, this menu item provides quick access to that configuration option.

End of Klayout Basics Section

Start of Klayout Editing Section

Editing Functions

Welcome to KLayout's user manual. This is the manual chapter covering the editing features of KLayout. The editor features are available only if KLayout is started in editor mode. The following subtopics are available:

Edit Mode

Editor functions can only be used if KLayout runs in edit mode. KLayout can be put into editing mode by simply supplying the "-e" option on the command line:

`klayout -e` [<input file>] [`-l` <layer properties file>]

Accordingly, with the command line option "-ne", non-editable mode can be enforced.

On Windows, there are start menu entries for editor and viewer mode. Klayout can be configured to use editing mode as default when started. To enable editing by default, check the "Edit mode" check box on the "Application" tab in the setup dialog ("File/Setup").

In editing mode, some optimizations are disabled. This results in somewhat longer loading times and a somewhat higher memory consumption. The actual increase strongly depends on the nature of the input file: for example, OASIS shape arrays are not kept as such in editing mode and resolved into individual shapes.

Basic Principles Of Editor Mode

This section covers the basic working principles of editor mode.

Pick And Drop Principle

Most drawing programs employ the click-and-drag paradigm: left-click on an element and drag it to the destination keeping the mouse button pressed. Although being pretty intuitive, this principle has one disadvantage: it is hard to do something other than dragging, while you keep the mouse button pressed. In particular this means: no zooming (or would you like to press the right mouse button as well, draw the zoom box and then release just the right mouse button ...?). In order to allow zoom and potentially other operations, KLayout employs the pick-and-drop-principle.

In pick-and-drop, you pick an element by clicking at it with the left mouse button, move it (without any mouse button pressed) and drop it (by left-clicking at the target position). Since the mouse button is not pressed, the mouse is free for other operations: just the dragged item is "sticking" to the mouse cursor.

In addition, while dragging the object, Shift and Ctrl keys can be used to force certain direction constraints or override the ones specified in the options (i.e. "move" or "edit" options): The Shift key forces KLayout into orthogonal mode: movements are restricted to horizontal or vertical unless not applicable. The Ctrl key forces KLayout into diagonal mode: movements are restricted to horizontal, vertical or the diagonal axes. Ctrl plus Shift will release all directional constraints - movements will be allowed in any direction.

Basic Editor Options

Most tools being using in editing mode have certain options, i.e. when drawing a path, the width and extension mode has to be specified. There exists a general setup dialog for editing options. It

can be opened using "Editor Options" from the "Edit" menu or using the F3 shortcut (unless overridden).

In the dialog there is always a generic settings tab and - depending on the tool chosen - a tool specific tab. On the generic tab, these settings can be changed:

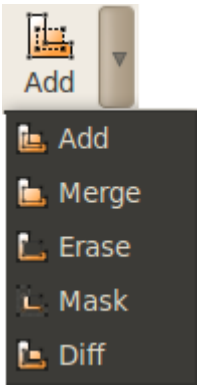
- **Editor grid:** Every editing operation is confined to that grid. It can be either disabled, aligned with the global grid (used i.e. for rulers and display) or specified explicitly. It can even be anisotropic - i.e. there can be a different grid in y than in x direction.
- **Connection angle constraint:** When a connection is drawing, i.e. a segment of a path or an edge of a polygon, this mode determines, if the segment or edge is confined to certain directions. In "Any Angle" mode, there is no such confinement. In "Diagonal" mode, the edge or segment can be vertical, horizontal or in one of the two diagonal directions. In "Manhattan" mode, only horizontal and vertical edges or segments are allowed.
- **Movement direction constraint:** When something a dragged (i.e. moved), this mode determines if the movement is confined to certain directions. In "Any Direction" mode, there is no such confinement. In "Diagonal" mode, the movement can be vertical, horizontal or in one of the two diagonal directions. In "Manhattan" mode, only horizontal and vertical movements are allowed.
- **Top level selection mode:** In top level selection mode (when the check box is checked), only elements on the level of the currently shown cell are selected ("top level" refers to the top level of the currently shown cell here). That means, that if shapes from a subcell are selected, the whole instance of this subcell is selected.

  In hierarchical selection mode (when the check box is not checked), elements are selected from subcells are well. This mode allows in-place editing of subcells which is a powerful feature but also creates strange side effects if other instances of this cell changes as well.

Whenever you change something in the settings dialog, use "Apply" or "Ok" to apply your changes.

Background combination modes

KLayout offers several ways to combine the shapes drawn with shapes that are already there. The mode can be selected with the "Select background combination mode" tool button in the tool bar:



These are the modes available:

- **Add**: this is the default mode: the drawn shape will simply be added to the existing shapes.
- **Merge**: in this mode, the drawn shape will be merged with the existing shapes. This operation will always render a polygon that is the drawn shape merged with any polygons that touch or overlap the drawn shape. Paths will be converted to polygons always. This mode is equivalent to a boolean "OR" operation.
- **Erase**: in this mode, the drawn shape will be subtracted from the existing shapes. This mode can be used to create notches or slits in shapes touching or overlapping the drawn shape. The drawn shape will vanish. This mode is equivalent to a boolean "NOT" operation.
- **Mask**: in this mode, the drawn shape will act as a mask for existing shapes. Only the parts overlapping the drawn shape will remain. This mode is equivalent to a boolean "AND" operation.
- **Diff**: finally, in this mode, the drawn shape will invert existing shape. This mode is equivalent to a boolean "XOR" operation.

Selection

The basic entity that some operations work with is the "selection". This is basically a set of shapes of instances on which an operation should be applied. A selection can be established by either clicking on a element in "Select" mode or by dragging a selection rectangle. When the mouse is released, all elements inside the selection rectangle are selected.

The selection set can be modified by adding elements (press the Shift button in addition to selecting elements), by removing elements (press Ctrl in addition) or by toggling the selecting (press Shift and Ctrl in addition: remove already selected ones and add new ones).

Partial Mode

"Partial editing" is a powerful feature that allows modification of shapes. Edges or segments of polygons resp. paths can be moved, vertices, edges or segments from polygons or paths can be deleted and new points can be inserted into polygons and paths. "Partial editing" can be applied to a complex partial selection: Multiple edges or vertices can be selected and deleted or moved.

The normal selection works "full element". In this mode, the whole shape is being moved or deleted. Only in full element mode, shapes or instances can be sent to the clipboard.

Basic Editing Operations

This section covers some basic operations when editing layout.

Creating A Layout From Scratch

To start with a fresh, empty layout, choose "New" from the "File" menu. A form is opened that requires you to specify some basic parameters. These are:

- **Top cell:** this is the name of the first (and only) cell that will be present in the layout.
- **Database unit:** this is the database unit (the conversion factor between integer coordinates and micron units. This is basically the "resolution" of the layout.

- **Initial window size:** this is the size of the initial window shown, when the top cell is opened the first time. Since the initial view is empty, there is no geometrical guidance. By specifying an initial size, at least the "canvas" dimensions are known.

If a default layer properties file is specified in the setup dialog ("Application" tab), this is loaded into the layer view list automatically. Without such a file, the layer list is empty at the beginning and layers must be created with "Layer/New" from the "Edit" menu, before any shapes can be drawn.

Creating A New Layer

You can create new layers using the "Layer/New Layer" function from the "Edit" menu. You are prompted to enter GDS layer and datatype numbers and optionally an OASIS layer name. On "Ok", the layer will be created and will be inserted into the layer panel.

Creating A New Cell

You can create new cells using the "New Cell" function from the hierarchy panel's context menu (right mouse click on the cell hierarchy panel). You are prompted to enter the new cell's name (a cell with that name must not exists yet) and a window size. The window size is the initial dimension of the view when the new cell is shown. Apart from that it does not have any meaning.

Creating A Polygon

Select "Polygon" mode from the toolbar. Choose a layer from the layer panel in which to create a new polygon. Left-click at the first vertex of the polygon. Move the mouse to the next vertex and place a new one with a left mouse button click. Move to the next vertex. Depending on the connection mode, the edges created are confined to certain directions. See Basic Editor Options for a detailed description of the modes. Use the "editor options" dialog (F3 shortcut) to change the mode, even during editing.

(Basic Editor Options is also inlined, next)

Double-click at the final point to finish the polygon. Press the ESC key to cancel the operation.

A polygon will never be "open": there are always edges connecting the current vertex with the initial one. Depending on the mode, this final connection is either a straight line or a combination of edges. In "diagonal mode", there are manifold possibilities to create a final connection in a more or less smart way. The program uses some heuristics to determine one feasible combination. Although this heuristics is not infinite smart, it should be easy to lead the algorithm to the desired solution, by pointing the mouse into the desired direction.

Basic Editor Options

Most tools being using in editing mode have certain options, i.e. when drawing a path, the width and extension mode has to be specified. There exists a general setup dialog for editing options. It can be opened using "Editor Options" from the "Edit" menu or using the F3 shortcut (unless overridden).

In the dialog there is always a generic settings tab and - depending on the tool chosen - a tool specific tab. On the generic tab, these settings can be changed:

- **Editor grid:** Every editing operation is confined to that grid. It can be either disabled, aligned with the global grid (used i.e. for rulers and display) or specified explicitly. It can even be anisotropic - i.e. there can be a different grid in y than in x direction.
- **Connection angle constraint:** When a connection is drawing, i.e. a segment of a path or an edge of a polygon, this mode determines, if the segment or edge is confined to certain directions. In "Any Angle" mode, there is no such confinement. In "Diagonal" mode, the edge or segment can be vertical, horizontal or in one of the two diagonal directions. In "Manhattan" mode, only horizontal and vertical edges or segments are allowed.
- **Movement direction constraint:** When something a dragged (i.e. moved), this mode determines if the movement is confined to certain directions. In "Any Direction" mode, there is no such confinement. In "Diagonal" mode, the movement can be vertical, horizontal or in one of the two diagonal directions. In "Manhattan" mode, only horizontal and vertical movements are allowed.
- **Top level selection mode:** In top level selection mode (when the check box is checked), only elements on the level of the currently shown cell are selected ("top level" refers to the top level of the currently shown cell here). That means, that if shapes from a subcell are selected, the whole instance of this subcell is selected.

  In hierarchical selection mode (when the check box is not checked), elements are selected from subcells are well. This mode allows in-place editing of subcells which is a powerful feature but also creates strange side effects if other instances of this cell changes as well.

Whenever you change something in the settings dialog, use "Apply" or "Ok" to apply your changes.

Creating A Box

Select "Box" mode from the toolbar. Choose a layer from the layer panel in which to create a new box. Left click at the first point, move the mouse to the second point and finish the box by left-clicking at the second point. Press the ESC key to cancel the operation.

Hint: A box, once created, will remain a box. For example, it is not possible to delete one vertex of it, thus forming a triangle. This is only possible for polygons.

Creating A Text Object

Select "Text" mode from the toolbar. The editor options dialog will open that additionally prompts for the text string. Don't forget to click "Apply" to take over the current string. If the dialog has been closed unintentionally, it can be reopened with the F3 shortcut.

To actually draw the text, move the mouse to the desired location and left-click to place it.

A text can be given a size which is stored in a GDS2 file (OASIS files do not provide this feature). The size of the text is only shown in the layout if a scalable text font is selected (the "Default" font is not scalable) and text scaling is enabled. In order to do so, choose a scalable font from the "Text font" selection box in the "Display" tab of the setup dialog and check the "Apply text scaling and rotation" box in the same tab.

The text can also be rotated, which is shown as well only if text scaling and rotation is enabled. To rotate a text while placing it, click the right mouse button. This will rotate the text by 90 degree counterclockwise.

Creating A Cell Instance

Select "Instance" mode from the toolbar. The editor options dialog will open that additionally prompts for some instance parameters. The most important one of course is the cell that shall be placed. Geometrically, the rotation angle can be specified, the mirror option can be set and the instance may be specified as a regular array. As an array, the instance represents multiple placements of the cell, arranged in regular grid which is specified by the two axis vectors and instance counts in each direction. Don't forget to click "Apply" to take over the current settings. If the dialog has been closed unintentionally, it can be reopened with the F3 shortcut.

To place the instance, move the mouse to the desired location and left-click to place it. While moving, the right mouse button can be used to rotate the instance by 90 degree counterclockwise. Press the ESC key to cancel the operation.

Starting with version 0.22, KLayout supports libraries. Libraries provide cells from the outside of the layout. These cells are imported into the layout and a copy is stored inside the layout. Still, KLayout maintains the reference to the original layout, so if the library changes, the cell will be replaced when the layout is loaded again. Cells imported from a library appear as "Library.Cell" in the cell tree and the layout. Here, "Library" is the library name and "Cell" is the cell name.

The library is selected from the pull-down box to the right of the cell name. You can use the "..." button to browse the cells available in the selected library.

Libraries are a convenient way to provide common layout building blocks. In addition, KLayout now also supports PCells (parametrized cells). Such cells do not have a static content, but instead they are created dynamically from a piece of code using a set of parameters specific for the PCell type. For example, a circle has two parameters: the layer where the circle should appear and the radius of the circle.

PCells are provided by libraries. If a PCell is selected from a library, the instance properties page also offers a panel to edit the PCell parameters. What parameters are available depends on the type of PCell.

KLayout comes with a standard library "Basic" which offers some basic curved shape types and a text generator.

KLayout offers a unique feature for the PCell implementation: a PCell can employ "guiding shapes". "Guiding shapes" are shapes that do not appear as layout themselves but are used by the PCell to derive it's geometry from. For example the "rounded path" PCell of the "Basic" library uses a path as a guiding shape. This path is manipulated to obtain the final shape.

Guiding shapes are drawn on the cell box layer and can be manipulated with the normal shape manipulation functions (in particular move and partial edit). Also, the shape properties can be edited via the "Edit/Properties" function.

To learn more about libraries, read About Libraries. Read About PCell's for details about PCell's.

(These "About" sections are also inlined, next)

About Libraries

Starting with version 0.22, KLayout offers a library concept. Libraries are a way to import cell layout into a layout from the outside and thus a convenient way to provide standard building

blocks for the layout. Using a cell from a library is easy: when asked for a cell, select the library where to take the cell from and choose a cell from that library.

Libraries are basically just foreign layouts that are virtually linked to the current layout. When a cell is imported from a library, it is copied into the current layout, so that the current layout by itself is a valid entity.

When a layout containing library references is saved, KLayout stores some meta information in that file which allows it to restore the library links and related information. For GDS, that meta information is stored in a separate top cell. For OASIS, the meta information is stored in special per-cell properties. For other formats, the meta information is not stored currently.

Libraries can be provided in several ways:

- **As ordinary layout files:** Such libraries are simple layout files (GDS, OASIS or other support format). KLayout looks up those libraries in the "libraries" subfolders of the search path and gathers all layout files it finds there into the library repository.

  The search path usually includes the installation site (where the KLayout executable resides) and the application folder (i.e. "~/.klayout" on Linux). Hence libraries can be installed locally (i.e. in "~/.klayout/libraries") or globally (at the installation site).

  For GDS files, the library name will be the LIBNAME of the GDS file. Otherwise it will be the name of the library file minus the extension.

- **Coded libraries:** Such libraries are provided by code, either through shared objects/DLL's or through Ruby code. Basically such code has to provide a layout object containing the library cells. A coded library can also provide PCell's (parametrized cells) as library components. Sell About PCell's for details about parametrized cells.

Starting with version 0.25, libraries can be provides through packages. This means, they can be downloaded from some repository and can be managed within the package manager. Library installation is very simple this way. Library deinstallation too. See About Packages for details about packages.

("About Packages" section appears previously in the Basics section)

About PCell's

Starting with version 0.22, KLayout offers parametrized cells (PCell's). PCells are a feature found in other tools to simplify layout by providing generators for common layout building

blocks. Parametrized cells do not contain static layout but are created dynamically by code using a given set of parameters.

For example, a circle PCell requires two parameters: the layer where the circle should be produced and the radius of the circle to produce. The code is responsible for creating the circle from these parameters.

Using a PCell is easy: choose the library and the PCell from that library when asked for the cell in the instance options dialog. For PCell's, KLayout offers an additional parameters page where it asks for the parameter required by the PCell. The placement of the PCell is done as for simple instances. PCell's offer the same instantiation options that normal cells.

KLayout provides a simple library called "Basic" with some useful basic PCell's. See About The Basic Library for more details about that library.

Unlike other tools, KLayout offers the unique feature of "guiding shapes". A guiding shape is some kind of "ghost shape" that is not produced as real layout but is present as a part of the PCell instance. It is drawn in the style of the cell frame but can be edited as a normal shape. In particular, a guiding shape can be manipulated with the properties dialog and the partial edit mode. A special case is a point-like shape which can act as a handle of the PCell. In move mode, these shapes can be moved to change the parameter related to that handle.

Another use case for guiding shapes is the rounded path. This PCell uses a path as the input shape and applies rounding to the path's spine corners to compute a new path which smoothly bends around the corners. The radius of the bends is a numerical PCell parameter while the input shape controlling the geometry and the width of the path is the guiding shape.

A PCell implementation consists of at least three parts: a description text, a parameter declaration and a production callback. In addition, a PCell can provide a method that "fixes" parameters according to the PCell's consistency rules (coerce parameters). Technically, a PCell is a class implementing a certain interface with these methods.

PCell's are usually packed in libraries. PCell libraries can be provided as shared objects/DLL's (in C++) or as Ruby scripts. Because PCell code is only executed if required, performance usually is not the main objective. A Ruby implementation will therefore be sufficient in most cases and is a much easier to maintain. The Ruby approach also benefits from KLayout's integrated development environment.

For an introduction into PCell programming with Ruby, see Coding PCell's In Ruby.

(The "About The Basic Library" section is also inlined, next; Ruby references are after that)

About The Basic Library

The "Basic" library

The "Basic" Library provides some useful generic PCell's. One use model is to draw a shape and convert the shape to one of the provided PCell's. This use model is suitable for creating Circles, Ellipses, Donuts, Texts and rounded and stroked polygons or rounded paths.

To use that feature, draw the shape and choose "Convert To PCell's" from the "Edit"/"Selection" menu. A dialog will be shown where the target PCell can be selected. Only those PCell's supporting that shape type will be shown.

The "Basic" library provides the following PCell's:

- **TEXT**: A text generator
- **CIRCLE**: A circle
- **DONUT**: A donut (circle with hole)
- **ELLIPSE**: An ellipse
- **PIE**: A pie (a segment of a circle)
- **ARC**: An arc (a segment of a donut)
- **ROUND_PATH**: A rounded path (a path bending around the corners with a given radius)
- **ROUND_POLYGON**: A rounded polygon (a polygon with rounded corners)
- **STROKED_BOX**: A stroked box (the "rim" of a box, optionally with smooth corners)
- **STROKED_POLYGON**: A stroked polygon (the "rim" of a polygon, optionally with smooth corners)

TEXT

The text generator can produce texts in various forms. The following sample shows inverse text, normal text and text with bias and enlarged character spacing:

It's even possible to install custom fonts. Fonts are basically GDS files with the following features:

- One cell per character. Cells must be either named like the character "A", "B", "0" etc. or like the ASCII code in 3-digit decimal notation (i.e. "036" for the dollar character).
- The characters must be drawn in the character cells on layer 1/0. A box defining the extension of the characters (including spacing) must be drawn on layer 2/0. Optionally a background rectangle for the "inverse font" feature can be drawn on layer 3/0.
- One cell called "COMMENT" with text objects defining the basic text properties through their strings, in particular:
    - **design_grid=x**: specifies the basic grid the characters are designed on. "x" is the grid in database units.
    - **line_width=x**: specifies the line width in database units.
    - A comment string which is displayed in the font selection box on the PCell parameters page.

Custom fonts are installed by copying the font file to a folder named "fonts" in one of the places in KLayout's path. The standard font can be found in "src/std_font.gds" in the source package.

CIRCLE and ELLIPSE

These PCell's define a circle and an ellipse. In both cases, the number of interpolation points (per full circle) can be specified. The default is 64 points. A circle features a handle to define the diameter. An ellipse features two handles defining the diameters in x and y direction.

When a shape is converted to a circle or ellipse PCell, the shape's bounding box will be used to define the enclosing box of the circle or ellipse.

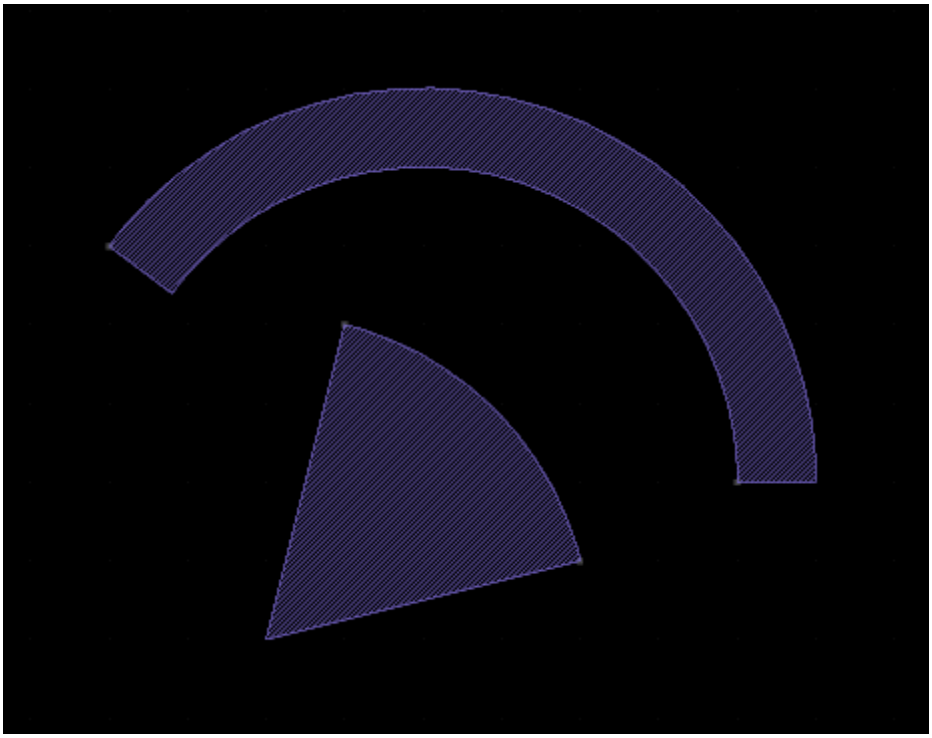
DONUT

The donut PCell creates a circle with a hole. This PCell features the same parameters than the circle but an additional parameter defining the hole radius. For that, it provides two handles - one for the outer and one for the inner radius.

When a shape is converted to a donut, the shape's bounding box will be used to define the enclosing box of the donut and the hole's diameter will be chosen to be half of the outer diameter.

PIE and ARC

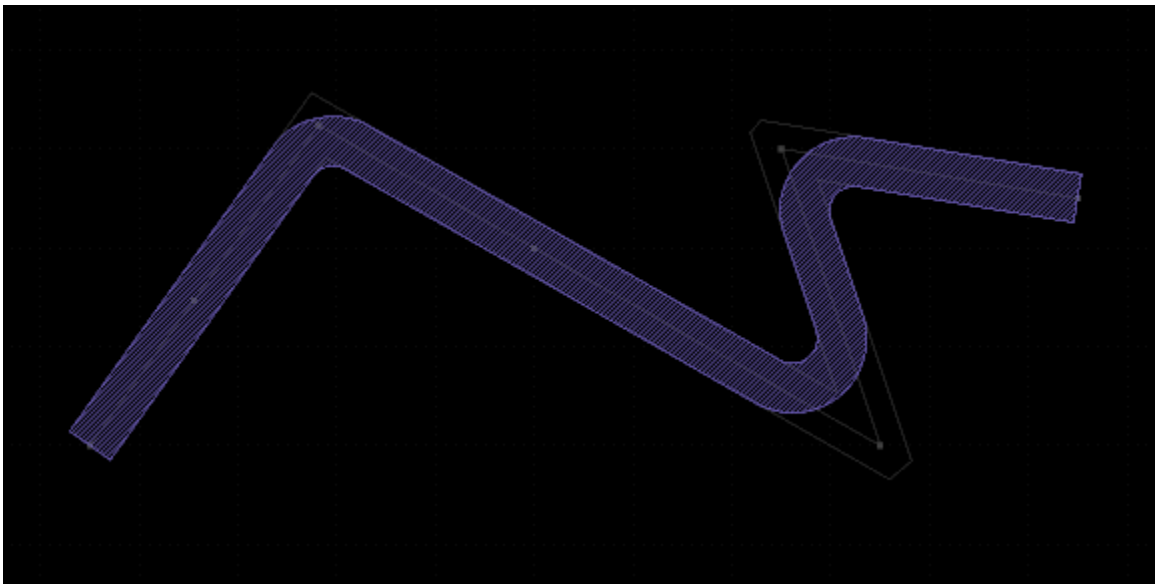
Both of these PCell's are segments of circles or donuts. The PIE PCell features two handles to define the radius and start and end angle. The ARC PCell also features two handles to define outer and inner radius as well. The following image shows PIE and ARC in action:



Both PCell's do not support conversion of shapes.

ROUND_PATH

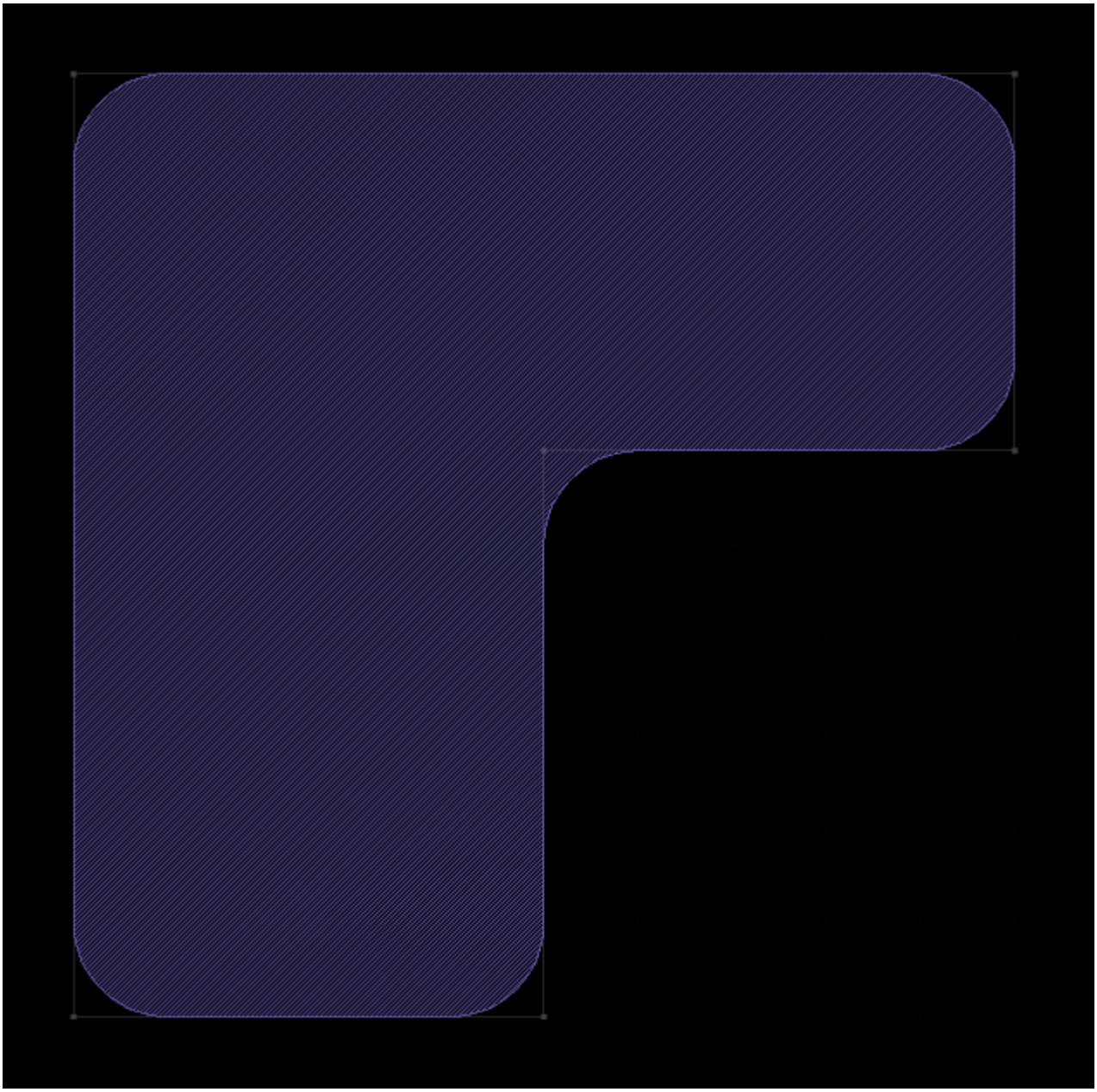
The round path is a PCell that is based on a path object but is capable of smoothing the path's corners by applying a radius. The following image gives an example:



The PCell features a parameter that defines the radius. The path itself can be manipulated as usual, in particular with partial edit mode. Path objects can be converted to ROUND_PATH pcells. In that case, the initial radius will be chosen to be roughly 10 percent of the minimum bounding box dimensions of the original path.

ROUND_POLYGON

The round polygon is a PCell that is based on a polygon object but is capable of smoothing the polygon's corners by applying a radius. The following image gives an example:



The PCell features a parameter that defines the radius. The polygon itself can be manipulated as usual, in particular with partial edit mode. Polygon, box or path objects can be converted to ROUND_POLYGON pcells. In that case, the initial radius will be chosen to be roughly 10 percent of the minimum bounding box dimensions of the original polygon.

STROKED_POLYGON or STROKED_BOX

The stroked polygon or box is a PCell that is based on a polygon object but will produce the "rim" of this polygon. In addition, it can apply corner rounding with a given radius.

The PCell features two parameters that define the radius and width of the "rim" line. The polygon or box itself can be manipulated as usual. Polygon, box or path objects can be converted to STROKED_POLYGON or STROKED_BOX pcells. In that case, the initial radius will be zero. The width of the rim line will be chosen to be roughly 10 percent of the minimum bounding box dimensions of the original polygon. For STROKED_BOX, the bounding box of the original shape will be used as the basic shape.

Coding PCell's In Ruby

A good starting point for Ruby PCell's is the PCell sample. Create a macro in the macro development IDE (use the "+" button) and choose "PCell sample" from the templates.

The Sample

We'll do a code walk through that sample here and explain the concepts while doing so. Here is the complete sample:

```
# Sample PCell
#
# This sample PCell implements a library called "MyLib" with a single PCell
that
# draws a circle. It demonstrates the basic implementation techniques for a
PCell
# and how to use the "guiding shape" feature to implement a handle for the
circle
# radius.
#
# NOTE: after changing the code, the macro needs to be rerun to install the
new
# implementation. The macro is also set to "auto run" to install the PCell
# when KLayout is run.

module MyLib

  include RBA

  # Remove any definition of our classes (this helps when
  # reexecuting this code after a change has been applied)
  MyLib.constants.member?(:Circle) && remove_const(:Circle)
  MyLib.constants.member?(:MyLib) && remove_const(:MyLib)

  # The PCell declaration for the circle
  class Circle < PCellDeclarationHelper

    include RBA

    def initialize

      # Important: initialize the super class
      super

      # declare the parameters
      param(:l, TypeLayer, "Layer", :default => LayerInfo::new(1, 0))
      param(:s, TypeShape, "", :default => DPoint::new(0, 0))
      param(:r, TypeDouble, "Radius", :default => 0.1)
      param(:n, TypeInt, "Number of points", :default => 64)
      # this hidden parameter is used to determine whether the radius has
changed
      # or the "s" handle has been moved
      param(:ru, TypeDouble, "Radius", :default => 0.0, :hidden => true)

    end

    def display_text_impl
      # Provide a descriptive text for the cell
      "Circle(L=#{l.to_s},R=#{'%.3f' % r.to_f})"
    end
```

```
    def coerce_parameters_impl

      # We employ coerce_parameters_impl to decide whether the handle or the
      # numeric parameter has changed (by comparing against the effective
      # radius ru) and set ru to the effective radius. We also update the
      # numerical value or the shape, depending on which on has not changed.
      rs = nil
      if s.is_a?(DPoint)
        # compute distance in micron
        rs = s.distance(DPoint::new(0, 0))
      end
      if rs && (r-ru).abs < 1e-6
        set_ru rs
        set_r rs
      else
        set_ru r
        set_s DPoint::new(-r, 0)
      end

      # n must be larger or equal than 4
      n > 4 || (set_n 4)

    end

    def can_create_from_shape_impl
      # Implement the "Create PCell from shape" protocol: we can use any
shape which
      # has a finite bounding box
      shape.is_box? || shape.is_polygon? || shape.is_path?
    end

    def parameters_from_shape_impl
      # Implement the "Create PCell from shape" protocol: we set r and l from
the shape's
      # bounding box width and layer
      set_r shape.bbox.width * layout.dbu / 2
      set_l layout.get_info(layer)
    end

    def transformation_from_shape_impl
      # Implement the "Create PCell from shape" protocol: we use the center
of the shape's
      # bounding box to determine the transformation
      Trans.new(shape.bbox.center)
    end

    def produce_impl

      # This is the main part of the implementation: create the layout

      # fetch the parameters
      ru_dbu = ru / layout.dbu

      # compute the circle
      pts = []
      da = Math::PI * 2 / n
```

```
      n.times do |i|
        pts.push(Point.from_dpoint(DPoint.new(ru_dbu * Math::cos(i * da),
ru_dbu * Math::sin(i * da))))
      end

      # create the shape
      cell.shapes(l_layer).insert(Polygon.new(pts))

    end

  end

  # The library where we will put the PCell into
  class MyLib < Library

    def initialize

      # Set the description
      self.description = "My First Library"

      # Create the PCell declarations
      layout.register_pcell("Circle", Circle::new)
      # That would be the place to put in more PCells ...

      # Register us with the name "MyLib".
      # If a library with that name already existed, it will be replaced
then.
      register("MyLib")

    end

  end

  # Instantiate and register the library
  MyLib::new

end
```

Preamble

The first important concepts are PCell class and library. A PCell is provided by implementing a certain class and providing the functionality of the PCell through various methods. In fact there are only three methods which must be implemented. In the sample we use PCellDeclarationHelper as the base class for our PCell. This is a convenience wrapper around the basic interface, PCellDeclaration. Since that interface is too much "C++"-like and is somewhat tedious to use, the PCellDeclarationHelper is the recommended starting point.

Using the same concept, a library is an object derived from the Library class. It is basically a container for PCell's and static layout cells. A library has to be initialized (most conveniently in the constructor), registered and initialized once. That makes the library available to the system and it can be used in layouts.

Please note, that the sample PCell is configured for auto-run. This way, the library is installed when KLayout starts and before any layouts are loaded. That way, the library is available for layouts read from the command-line for example.

Let's now start with our code walk:

```
module MyLib

  include RBA

  # Remove any definition of our classes (this helps when
  # reexecuting this code after a change has been applied)
  MyLib.constants.member?(:Circle) && remove_const(:Circle)
  MyLib.constants.member?(:MyLib) && remove_const(:MyLib)
```

It is recommended to put the library code into a separate module. That allows mixing in other modules (in that case RBA) without affecting the main module. The second recommendation is to remove classes which are already defined with the names we are going to create. While developing a PCell is it necessary to frequently rerun the script to register the new version of the library and PCell. If we do not remove the existing class, Ruby will refuse to reopen a class for example if we change the super class or methods we have deleted will still remain. That is avoided by removing the classes before the create them again. In Ruby, a class can be removed by removing the constant with the class name. Note the way, the script checks whether a class is defined by using "member?" on the list of constants. This method should be preferred over "const_defined?" which behaves differently on Ruby 1.8 and Ruby 1.9.

The PCell Class

First we define a PCell class derived from PCellDeclarationHelper. This is the most convenient way to declare a PCell:

```
  # The PCell declaration for the circle
  class Circle < PCellDeclarationHelper

    include RBA
```

Again we include RBA which allows us to use RBA classes inside the PCell without having to write "RBA::" in front of the class names.

The initialization of the object is already a very important step. First, it must initialize the super class. Then it has to declare the PCell parameters. Each PCell has a set of parameters that define the appearance of the PCell. Parameters have a symbolic name, a type, a description and optionally a default value and further attributes. The name is important because it identifies the parameter throughout the system and in layout files as well. It should not be changed. The description is an arbitrary string and can be changed or localized.

Parameters are declared using the "param" method of PCellDeclarationHelper:

```
    def initialize

      # Important: initialize the super class
      super

      # declare the parameters
      param(:l, TypeLayer, "Layer", :default => LayerInfo::new(1, 0))
      param(:s, TypeShape, "", :default => DPoint::new(0, 0))
      param(:r, TypeDouble, "Radius", :default => 0.1)
      param(:n, TypeInt, "Number of points", :default => 64)
      # this hidden parameter is used to determine whether the radius has
changed
      # or the "s" handle has been moved
      param(:ru, TypeDouble, "Radius", :default => 0.0, :hidden => true)

    end
```

In that sample we declared a PCell parameter "l" with type "TypeLayer" which indicates that this is a layer in the layout. "s" is a parameter shape represents the handle and is a shape. A shape is either of type DBox, DText, DPath, DPolygon or DPoint. Shape parameters implement the "guiding shape feature" that KLayout offers to manipulate that parameter graphically. "r" and "n" are simple numerical parameters. All parameters have default values which are set with the "default" symbolic parameter. As a layer, "l" must have a LayerInfo value. "s" is a DPoint which reflects the handle. Since default values not only preset the parameters to a reasonable value but also define the subtype of a parameter (here the DPoint shape), providing a default is strongly recommended. As shapes need to be independent from the database unit for portability, they are expressed in micron units. Hence the use of the "D" forms (DPoint etc.).

"ru" is a special parameter. Because we have two ways to modify the radius (the handle and the numerical value), it is used as a shadow parameter do determine which one of these two values has changed. Depending on that information, either the handle or the radius is updated. Because this parameter should not be shown in the parameter page, it is marked "hidden".

There are some more options for parameters. See the documentation of PCellDeclarationHelper for more details about the further attributes.

The parameter declaration will create accessor methods for each parameter. These accessor methods can be used to get and set the current value of the parameter inside the production method and other methods. For that, it will use the symbolic name of the parameter. The setter is called "set_x" (where x is the parameter name). Although Ruby would allow using "x=" to mimic an assignment, this option leads to some confusion with definition of local variables and was not considered here. The following methods are created in the sample:

- `l, set_l, l_layer`: getter and setter for the current value of "l". l_layer is the layer index in the context of the PCell production method. The layer index can be used to access the layer in the layout or cell.
- `s, set_s`: getter and setter for the current value of "s".
- `r, set_r, n, set_n, ru, set_ru`: same for "r", "n" and "ru".

After the PCell initialization is finished, we can start with the production code. These are the methods that KLayout will call on certain opportunities. The first method that a PCell must implement is the display text callback:

```
def display_text_impl
  # Provide a descriptive text for the cell
  "Circle(L=#{l.to_s},R=#{'%.3f' % r.to_f})"
end
```

KLayout will call this method to fetch a formatted string that represents the content of a PCell. This text is used for the cell tree and cell box labels. To avoid confusion, it should start with the name of the PCell. The bracket notation is not mandatory, but it's always a good idea to follow some common style. The information delivered by this method should be short but contain enough information so that a PCell variant can be distinguished from it's sibling.

The next method is called whenever something on the parameters has changed. This method allows to adjust the parameters so that they obey certain limitations. It can also raise exceptions for invalid parameter combinations. In our case we use this method to adjust the handle or the numeric radius to the effective value. We also enforce a minimum number of vertex counts for the resulting polygon. Implementing this method in general is optional. By default, no modification of the parameters is done:

```
def coerce_parameters_impl

  # We employ coerce_parameters_impl to decide whether the handle or the
  # numeric parameter has changed (by comparing against the effective
  # radius ru) and set ru to the effective radius. We also update the
  # numerical value or the shape, depending on which on has not changed.
  rs = nil
  if s.is_a?(DPoint)
    # compute distance in micron
    rs = s.distance(DPoint::new(0, 0))
  end
  if rs && (r-ru).abs < 1e-6
    set_ru rs
    set_r rs
  else
    set_ru r
    set_s DPoint::new(-r, 0)
  end

  # n must be larger or equal than 4
  n > 4 || (set_n 4)

end
```

The implementation of the following three methods is optional: they are used to implement the "PCell from shape" protocol. If "Create PCell from shape" is selected in KLayout's Edit menu, it will call "can_create_from_shape_impl" for each known PCell. This method will be given the shape, layout and layer. If this method responds with "true", KLayout offers this PCell as a conversion target in the list. When this PCell has been selected, KLayout calls

"parameters_from_shape_impl" and "transformation_from_shape_impl" to obtain the initial parameters and the initial transformation for the new PCell created from that shape. "parameter_from_shape_impl" will use the default values for all parameters unless they are set with the respective setters in the implementation body.

```
    def can_create_from_shape_impl
      # Implement the "Create PCell from shape" protocol: we can use any
shape which
      # has a finite bounding box
      shape.is_box? || shape.is_polygon? || shape.is_path?
    end

    def parameters_from_shape_impl
      # Implement the "Create PCell from shape" protocol: we set r and l from
the shape's
      # bounding box width and layer
      set_r shape.bbox.width * layout.dbu / 2
      set_l layout.get_info(layer)
    end

    def transformation_from_shape_impl
      # Implement the "Create PCell from shape" protocol: we use the center
of the shape's
      # bounding box to determine the transformation
      Trans.new(shape.bbox.center)
    end
```

The most important method is "produce_impl" which actually creates the layout. For that, it can use all methods of Layout and Cell and most other RBA classes. It can even create instances. Although that is possible, it is not recommended to create cells in the production code. This would pretty much degrade performance and lead to a confusing variety of cells. It is possible to use boolean operations by using the methods of EdgeProcessor for example. Some care must be taken to avoid interaction with the user interface, in particular calling methods of LayoutView and MainWindow should be avoided.

The actual layout of the PCell is cached and the production code is called only when the PCell parameters have changed. However, to reduce the risk of performance degradation, the method should run quickly and not spend too much time in long loops or huge data sets.

```
    def produce_impl

      # This is the main part of the implementation: create the layout

      # fetch the parameters
      ru_dbu = ru / layout.dbu

      # compute the circle
      pts = []
      da = Math::PI * 2 / n
      n.times do |i|
        pts.push(Point.from_dpoint(DPoint.new(ru_dbu * Math::cos(i * da),
ru_dbu * Math::sin(i * da))))
      end
```

```
      # create the shape
      cell.shapes(l_layer).insert(Polygon.new(pts))

    end

  end
```

Of course, more than one PCell class can be declared. Each PCell type must have an own implementation class which we will use later to create the PCell's from.

The Library

The library is the container for the PCell's. All important code is packed into the constructor of the library.

```
  # The library where we will put the PCell into
  class MyLib < Library

    def initialize

      # Set the description
      self.description = "My First Library"

      # Create the PCell declarations
      layout.register_pcell("Circle", Circle::new)
      # That would be the place to put in more PCells ...

      # Register us with the name "MyLib".
      # If a library with that name already existed, it will be replaced
then.
      register("MyLib")

    end

  end
```

First, a library needs a description that we set with the description setter. Then, we instantiate all PCell classes once and register that instance in the library space.

The library is basically an ordinary Layout object that we can access through the "layout" method. The library can consist of more that PCells - all cells that we put into the layout will become available as library components (more precisely: all top cells). We could use RBA::Layout::read for example to feed the layout with cells from a file.

At the end of the constructor we register our instance inside the system with the given name. To avoid confusion, it is recommended to use the same name for the class and the library.

Finally we only need to instantiate the library:

```
  # Instantiate and register the library
```

```
MyLib::new
```

This line of code will instantiate the library and, through the constructor, instantiate the PCell's and register the library. We are done now and can use the library and our PCell.

Debugging The Code

When you have modified the code, you need to rerun the script. That will create the classes again and re-register the PCell's and the Library with the new implementation. PCell's already living in the layout will be migrated to the new implementation by mapping their parameters by their symbolic names.

The PCell code can be debugged with KLayout's built-in Ruby debugger. If the macro development IDE window is open, just load the PCell code and set a breakpoint. When KLayout calls the PCell implementation, the breakpoint will be triggered. Local variables can be inspected and modified in the console for example. Single-stepping is supported as well. If the execution is stopped, KLayout will finish the operation with some error message.

It is also possible to print output to the console if the macro development IDE is open. Just use the methods of stdout that Ruby offers or simply "puts".

Please note that while the macro development IDE is opened, macro execution is considerably slower than usually, because the IDE will plug itself into the Ruby interpreter and trace the execution. When the IDE window is closed, Ruby runs at full speed. While in a breakpoint, KLayout's main window is only half alive. Only the IDE is active and the main window will not even repaint correctly. This prevents possible interactions with the executed code.

Moving The Selection

The whole selection can be moved in "Move" mode. If some elements are already selected, choose "Move" mode and select a reference point by left-clicking at the position. The reference point will be used as the "dragging handle" - each element is moved relative to this position. In no elements are selected when entering move mode, simply click at the element to move and place it somewhere else with a left mouse click.

While moving, the whole selection can be rotated by 90 degree counterclockwise with a right mouse button click. The ESC key will cancel the operation.

For movements, the movement direction constraint apply. See Basic Editor Options for details about the modes available. For example, in manhattan mode, only horizontal and vertical movements are allowed. The global movement constraint can be overridden by pressing Shift (orthogonal), Ctrl (diagonal) or both Shift and Ctrl (any angle) while moving the mouse.

If a move distance and direction is known numerically, "Move By" from the "Edit/Selection" menu can be used. A dialog will open that allows specification of the horizontal and vertical move distance in micrometers. Positive values move to the top or right and negative ones to the bottom or left. This dialog also applies to partial mode, so that edges or parts of a layout can be moved precisely by a certain distance in a certain direction.

In the same way, "Move To" allows one to reposition the selection to a certain point. The point which is positioned can be chosen relative to the bounding box of the selection. If first example, the lower-left corner of the selection is picked as the reference point and a certain position is given in the dialog, the selection is moved such that the lower left position of it's bounding box will match the given coordinate.

Note: A new "Infix Move" function has been created as of 0.26 but the documentation of it has not been added to the HTML-pile as yet?

Other Transformations Of The Selection

The selection can be flipped at x- or y-axis, rotated as a whole or moved by a certain distance using the functions available in the "Selection" submenu of the "Edit" menu. For example, "Flip Vertically" flips the selection at the x-axis. A selection can be rotated by an arbitrary angle using the "Rotation By Angle" function from the "Selection" submenu.

Partial Editing

When objects have to be modified after the have been created, partial editing comes into play. "Partial" refers to the fact that just parts of a polygon or path are edited. For example, just one vertex or an edge of a polygon can be moved. Partial editing mode also allows deleting single vertices or edges or to insert new ones. In partial editing mode, multiple edges or vertices can be selected, even a whole shape can be selected and can then be moved or deleted.

When moving the selected parts, the movement direction constraint applies. See Basic Editor Options for details about the modes available. For example, in manhattan mode, only horizontal and vertical movements of parts are allowed. Again, the global movement constraint can be overridden by pressing Shift (orthogonal), Ctrl (diagonal) or both Shift and Ctrl (any angle) while moving the mouse.

To enter partial mode, click on the "Partial" button in the toolbar. Parts (edges or vertices) can then be selected either by simply clicking at them or by dragging a selection rectangle. As in normal selection mode, the modifier buttons Shift and Ctrl can be used to add a selection to the existing one or to remove elements from the existing selection. Partial selection is subject to the

"top level only" constraint (see Basic Editor Options for a description of the top level selection mode).

Simply clicking at an item immediately enters "move" mode. In this mode, you can position the element at the desired target location and place it there by left-clicking at the position. Press "ESC" to cancel the operation. When a complex selection is made, move mode is entered by clicking at one of the selected items (the edges or vertices, not the shape to which they belong).

When moving parts, certain constraints apply, i.e. single edges can only be moved perpendicular to their current position. In addition, the movement is confined to the editing grid.

The selected items can be deleted by using the "Delete" function from the "Edit" menu or pressing the "Delete" key. If not enough vertices remain to form a valid object, the object is deleted (i.e. a polygon with less then 3 points).

By double-clicking at an edge or path segment, an additional point is created on this edge at the cursor's position. You can create a "bend" on a path by placing two new vertices on that segment and moving the connecting segment between these vertices away from the former center line. This basically requires two double-clicks on the path's centerline, a single click on the newly formed segment and a single click to drop it at the new position.

Moving Shapes To A Different Layer

The selected shapes can be moved to a different layer as a whole. For this, choose "Change Layer" from the "Selection" submenu of the "Edit" menu. All selected shapes are moved to the layer that is the current one (marked with a rectangle) in the layer list. The shapes will not be moved across the hierarchy but just inside their cell.

All layers (source and target) must be located in the same layout. To move shapes to a different layout, use copy & paste.

Other Layer Operations

The layer specification can be edited using the "Edit Layer Specification" method from the "Layer" submenu inside the "Edit" menu. A dialog is shown in which the layer, datatype and (OASIS) name of the layer currently selected in the layer panel can be edited. On save, the shapes are then mapped to the new layer.

A layer can be cleared (either cellwise, on a cell's hierarchy or for all cells) using the "Clear Layer" method from the "Layer" submenu inside the "Edit" menu.

Layers can be copied (duplicated) using "Copy Layer" from the "Layer" submenu inside the "Edit" menu and deleted completely using "Delete Layer".

Copy And Paste Of The Selection

Of course, copy and paste is supported as usual. Shapes can be copied between layouts: by opening two layouts, shapes can be moved from one layout to another. The shapes are mapped to the same layer than they have been on in the source layout. If a layer does not exist yet in the target layout, it is created.

Shapes in the selection are simply copied to the clipboard in the way they appear in the current cell. This means, if the shapes are pasted into a different layout they are put on the same position, but flat into the current cell. This provides a way to flatten a hierarchy: choose "hierarchical selection mode" in the editor options dialog (deselect "top level only"), select the shapes to flatten and copy everything to a different cell.

In non-hierarchical selection mode ("top level only" selection mode) or by clicking on a cell frame when the hierarchy levels are limited, instances can be selected as well. When copying instances to the clipboard, two possible methods exist:

- **Shallow copy:** In this mode, just the instance is copied. When it is pasted into any target layout, the target cell of the instance is looked up and instantiated.
- **Deep copy:** Not only the instance but the instantiated cell is copied as well. When pasting that into a different layout, the target cell will be created as well. If a cell with that name already exists, a variant is created and instantiated.

Delete A Cell

To delete a whole cell, select the cell in the hierarchy browser and choose "Delete Cell" from the context menu (right mouse button). This time, three possible modes are offered:

- **Shallow delete:** Just the cell (it's shapes and instances) are deleted, not any cells referenced by this cell. Since cells might no longer be referenced after that, they may appear as new top cells in the layout.
- **Deep delete:** The cell and all it's subcells are deleted, unless the subcells are referenced otherwise (by cells that are not deleted). In this delete mode a complete hierarchy of cells can be removed without side effects.
- **Complete delete:** The cell and all it's subcells are deleted, even if other cells would reference these subcells.

Rename A Cell

To rename a cell, select the cell in the hierarchy browser and choose "Rename Cell" from the context menu (right mouse button). You are prompted for a new name which must not exist yet.

Copy And Paste Of Cells

Whole cells can be copied to the clipboard as well. To copy a whole cell, select the cell in the hierarchy browser (make sure the focus is in that window) and choose "Copy" or "Cut" from the "Edit" menu. To paste such a cell into a target layout, choose "Paste" from the "Edit" menu.

Two copy modes are provided: deep and shallow copy. When "copy" or "cut" is chosen and the cell instantiates other cells, a dialog will been shown in which the mode can be selected:

- **Shallow copy**: In shallow mode, only the cell itself will be copied. No copies of the child cells will be created. If a cell copy is created, the cell will call the same cells than the original cell. If a cell is copied to another layout (in a different tab), the child cells will not be carried along and "ghost" cells will be created or existing cells with the same name will be used as child cells.
- **Deep copy**: In deep copy mode, the cell plus it's child cells are copied. All cells will be carried along and when pasting the cell, copies of all children will be created as well.

When a cell is pasted into another layout and there is a "ghost cell" with that name, the pasted cell will replace the ghost cell. If there is a normal cell with that name, a new cell variant will be created and the name of the pasted cell will be changed by adding a suffix to create a unique name.

Copying a cell in deep copy mode from one layout to another provides a way to merge two layouts into one: simply copy the top cell of the first layout into the second one and instantiate both in a new top cell for example.

End of Basic Editing Operations
Start of Advanced Editing Operations

Advanced Editing Operations

This section covers the more advanced features of editor mode.

Layout Transformations

Some functions are available to transform a whole layout. Whole-layout transformations are applied to all cells in a way that every cell will be modified in the same way. This feature is specifically useful for scaling layouts. It is worth noting that scaling a layout this way does not produce instances with magnifications.

The layout transformation functions are available in "Edit/Layout" (flip, rotate, scale, move).

Search and Replace

KLayout offers a "search and replace" function which provides a basic search and search+replace, but also a very generic and powerful extended feature called "custom queries" (see below for that). The search and replace dialog can be found in the "Edit" menu under "Search and replace".

The dialog provides four tabs in the left panel: "Find", "Delete", "Replace" and "Custom". The functionality of these four tabs is explained below.

The left side of the dialog will hold results for operations which deliver a result, for example the "Find" operation. The result is a list which displays various items, depending on the nature and parameters of the operation. If the item represents a layout object, for example a shape or a cell instance, the items selected in the list are highlighted in the layout to indicate their position. The length of the list is limited to avoid performance degradation for very long lists. The number of items shown can be configured on the configuration page.

The configuration page which allows configuration of the search and replace dialog's behavior is shown when the "Configure" button at the left bottom corner is pressed. It can be found as well in the setup dialog under "Browsers", "Search Result Browser".

Find

The functionality of the "Find" tab is simple: Various conditions can be specified and all objects matching that condition are listed when the "Find" button is pressed.

The parameters of that function involve: Object type, cell scope and object specific conditions. The object type is either "Instances" or a shape object. For shapes, the shape type can be confined to "Box", "Polygon", "Path" or "Text" which enables specific features.

The cell context can be one of:

- **Current cell**: look in the current cell and none else. Child cells are ignored.
- **Current cell and below**: look into the current cell and all child cells, where all instances of the children are considered.
- **All cells**: look into every cell individually, but don't consider the way the cells are instantiated.

Depending on the object type various parameters are available to be included in the condition. Each condition applies to one specific parameter and is usually composed of an operator (less, less than, equal ...) and a value against which the value of the parameter is checked. Length and area values are given in micron or square micron units.

String values can be matched against glob pattern using the tilde ("~") match and non-match ("!~") operators. Glob pattern are the ones used for file names on the command line and use "*" for an arbitrary sequence of characters and "?" for a single arbitrary character. Here are some examples for glob pattern:

A*	The string must start with a capital "A"
A	The string must contain a capital "A" somewhere
ABC?	"ABC" followed by any character
N{AND,OR}	"NAND" or "NOR"
A[0-9]	"A" followed by a digit
A[^0-9]	"A" followed by a non-digit

If the value field is left empty, no check is made on that parameter. All conditions which are checked must be fulfilled to make the object listed on the results page.

Delete

Similar to the find page, this function asks for an object type, a cell context and object specific conditions.

If the "Delete All" button is pressed, all selected objects are deleted.

If the "Select+Delete" button is pressed, the selected objects will first be shown in the result page, similar to the "Find" function. Then, some or all of them can be selected and deleted by pressing the "Delete" button below the list.

Replace

Similar to the find page, this function asks for an object type, a cell context and object specific conditions. In addition, the function allows specification of replacement values for the parameters. If an entry field is left empty for the replacement value, no replacement is made.

For strings with glob pattern matching, name parts can be reused in the replacement string. For example, if the operator is "~" on a text's string and the match string is "A(*)", the replacement string can be set to "B\1". "\1" means the value of the first bracket in the match string, hence this setup replaces all leading "A"'s by "B".

If the "Replace All" button is pressed, the replacement parameters are set on all selected objects.

If the "Select+Replace" button is pressed, the selected objects will first be shown in the result page, similar to the "Find" function. Then, some or all of them can be selected and the replacement is made on them the "Replace" button is pressed below the list.

Custom queries

The full power of this dialog is unleashed when using that page. Custom queries are not only able to provide the functionality of the three other pages (find, delete and replace), but provide functionality far beyond that simple scenarios.

Custom queries are statements resembling SQL statements with embedded expressions and a rich language to describe shape or cell instantiation details. A in-depth description can be found here: About Custom Layout Queries.

The custom query dialog page offers an entry field to enter the query and below an "Execute" button to run it. The most recently used queries can be pulled back into the edit field using the drop-down box below the edit field. Custom queries can be saved under a given name and reused. The list of saved queries can be manipulated with the buttons right to it. See the button's tooltips for a description of the button's functionality.

If the tab is switched to another tab and back, the custom query will be updated reflecting the query corresponding to the current functionality selected on the other tab.

Hierarchical Operations: Flatten Instances, Make Cell From Selection, Move Up In Hierarchy

Klayout provides several operations that move shapes or instances up and down in hierarchy. All these operations are accessible through the "Edit" menu in the "Selection" sub-menu.

- **Flatten instances**: Replace the selected instances by the contents of the instantiated cell. KLayout will ask, if all levels or just the first level of the cell should be expanded. If all levels are expanded, the cell will be resolved into a set of shapes in the current cell's hierarchy.
- **Move up in hierarchy**: Applies only to selections inside child cells of the current cell (thus does not make sense if 'top level only' selection mode is active). The selected shapes and instances are brought up to the current cell's level and removed from the original cell.
  A non-destructive way of moving a shape up in the hierarchy is to copy and paste the shape. This does an explicit flattening of the shapes selected when inserting them.
  Hint: the current implementation removes the selected object from it's original cell. Since it only creates new copies for the selected instances, the object ist lost for all other instances of the cell. This may create undesired side effects and it is likely that this behaviour will change in future implementations.
- **Make cell from selection**: Removes the currently selected objects and places them into a new cell whose name can be specified.

Creating Clips

KLayout provides a utility to create rectangular clips from a given cell. One or more rectangles can be specified. The current cell is cut along the edges of these rectangles. For each rectangle, a new cell is created containing the clipped content for the rectangle. Finally, if more than one rectangle is specified, all the clips are combined into a master top cell which appears as a new top cell in the cell hierarchy.

The clips can be either specified by coordinates, taken from another layer (which must contain boxes which then are copied into the output as well) or taken from the rulers. In the latter case, the rulers' start and end points are taken as the corners of the clip rectangles. It is convenient therefore to create a new ruler type with a box appearance for this purpose.

Clips are done hierarchically: child cells are clipped as well, potentially creating variants (which may be shared by several clips). This way, large clips can be created from large layouts in an efficient way. **Hint:** Clipping will not work exactly if the layout contains cell instances with arbitrary rotation angles such as 45 degree.

Creating Clips

KLayout provides a utility to create rectangular clips from a given cell. One or more rectangles can be specified. The current cell is cut along the edges of these rectangles. For each rectangle, a new cell is created containing the clipped content for the rectangle. Finally, if more than one rectangle is specified, all the clips are combined into a master top cell which appears as a new top cell in the cell hierarchy.

The clips can be either specified by coordinates, taken from another layer (which must contain boxes which then are copied into the output as well) or taken from the rulers. In the latter case, the rulers' start and end points are taken as the corners of the clip rectangles. It is convenient therefore to create a new ruler type with a box appearance for this purpose.

Clips are done hierarchically: child cells are clipped as well, potentially creating variants (which may be shared by several clips). This way, large clips can be created from large layouts in an efficient way. **Hint:** Clipping will not work exactly if the layout contains cell instances with arbitrary rotation angles such as 45 degree.

Resolving Arrays

Instance arrays are handy to produce large regular arrangements of cells. Sometimes it is necessary to resolve these arrays - for example to remove or modify a single instance from the array. One way to achieve that is to create cell variants (see Create Cell Variants, but this feature will also create a copy of the instantiated cell.

(The "Create Cell Variants" section appears at the end of this subsection)

The "Resolve Arrays" function available in "Edit/Selection" allows resolving of an array into individual instances which then can be edited individually.

PCell Operations

PCell's can be created from shapes if the PCell is derived from a "guiding shape". Specifically the Basic library PCell's support derivation from guiding shapes with a few exceptions. So for example, a round-cornered polygon can be created from a normal polygon by selecting the polygons, choosing "Edit/Selection/Convert To PCell" and selecting the "Basic.ROUND_POLYGON" for the PCell.

PCell's can be converted to normal cells by choosing "Edit/Cell/Convert Cell To Static" or "Edit/Layout/Convert All Cells To Static". Normal (static) cells can be edited individually but do no longer offer parameters to control the look of the cell.

Layer Boolean Operations

KLayout now comes with a set of boolean operations. These operations are available in the "Layers" submenu of the "Edit" menu ("Boolean Operations" and "Merge" functions). A dialog will open that allows specification of mode, input layer(s), output layer and certain other options.

- **AND**: intersection. The output layer will contain all areas where shapes from layer A and layer B overlap.
- **A NOT B**: difference. The output layer will contain all areas where shapes from layer A are not overlapping with shapes from layer B.
- **B NOT A**: difference. The output layer will contain all areas where shapes from layer B are not overlapping with shapes from layer A.
- **XOR**: symmetric difference. The output layer will contain all areas where shapes from layer A are not overlapping with shapes from layer B and vice versa.

In addition, a **MERGE** operation is provided, which is a single-layer operation that joins (merges) all shapes on the layer. As a special feature, this operation allows selecting a minimum overlap count: 0 means that output is produced when at least one shape is present. 1 means that two shapes have to overlap to produce an output and so on. This does not apply for single polygons: self-overlaps of polygons are not detected in this mode.

All operations can be performed in three hierarchical modes:

- **Flat**: Both layers are flattened and the results are put into the current top cell.
- **Top cell**: perform the operation on shapes in the top cell only.
- **Cellwise**: perform the operation on shapes of all cells below the current top cell individually. This mode is allowed only if the layouts of both inputs and output are the same.

For the first two modes, the source and target layout can be different, provided that all layouts are loaded into the same view. This allows combining layers of different layouts. For example to compare them using a XOR function.

As a special feature, KLayout's boolean implementation allows choosing how "kissing corner" situations are resolved. KLayout allows two modes:

- **Maximum coherence**: the output will contain as few, coherent polygons as possible. These polygons may contain points multiple times, since the contour may return to the same point without closing the contour.
- **Minimum coherence**: the output will contain as much, potentially touching polygons as possible.

The following screenshots illustrate the maximum coherence (left) and minimum coherence (right) modes for a XOR operation between two rectangles.

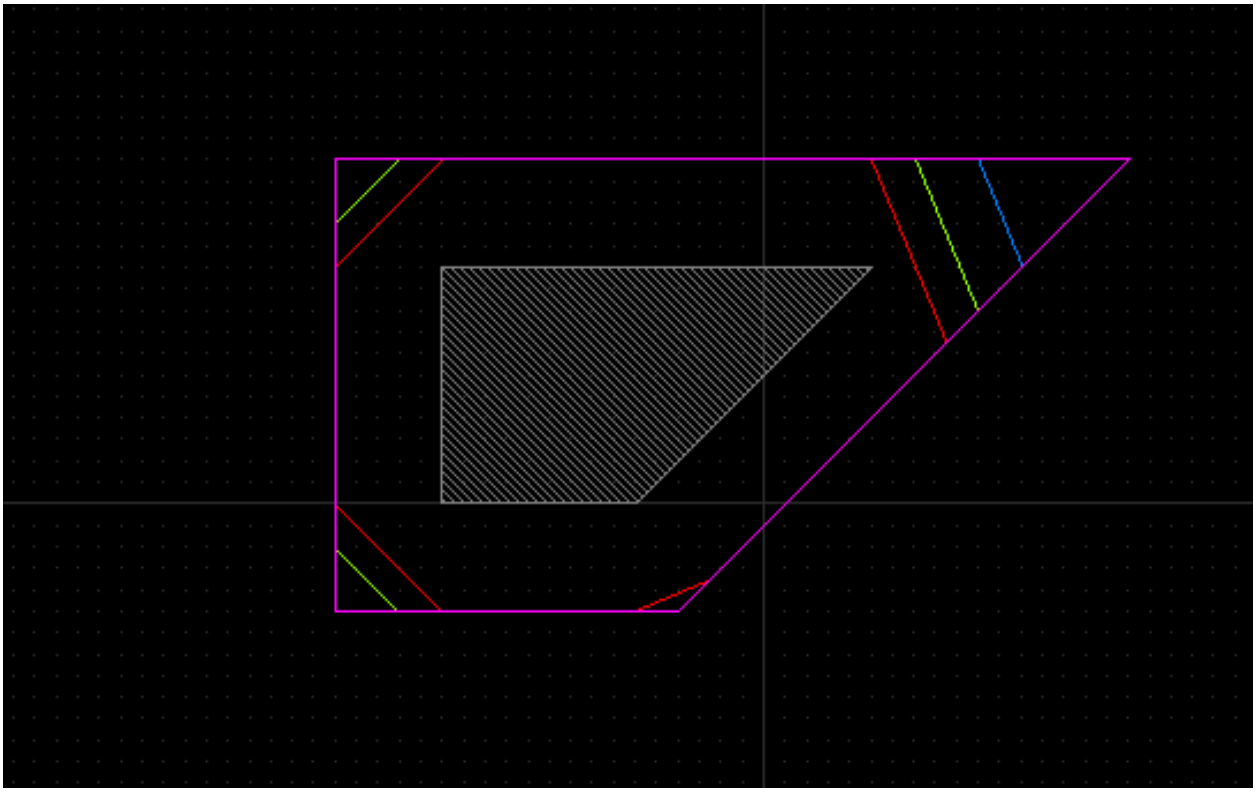
Layer Sizing

A sizing operation allows growing or shrinking of the shapes of a layer by a given offset, which is applied per edge. Positive values will enlarge the shapes while negative values will shrink the shapes. The offset can be given separately for horizontal and vertical direction. However, the sign of both values must be identical (i.e. "0.5,0" or "1.0,0.2", but not "0.2,-0.2").

The sizing function can be found in the "Layers" submenu of the "Edit" menu. A dialog will open that allows specification of input and output layers, sizing value in micron: a single value for same sizing in x and y direction or comma-separated list of two values (i.e. "0.2,0.1").

As for the boolean operations, hierarchical mode and kissing corner resolution can be specified (see Layer Boolean Operations for a description of these modes). In addition, the cutoff strategy

for sharp edges can be chosen from strict to virtually unlimited. The following screenshot demonstrates the effect for "strict" (red) to "weak" (purple) cutoff modes.



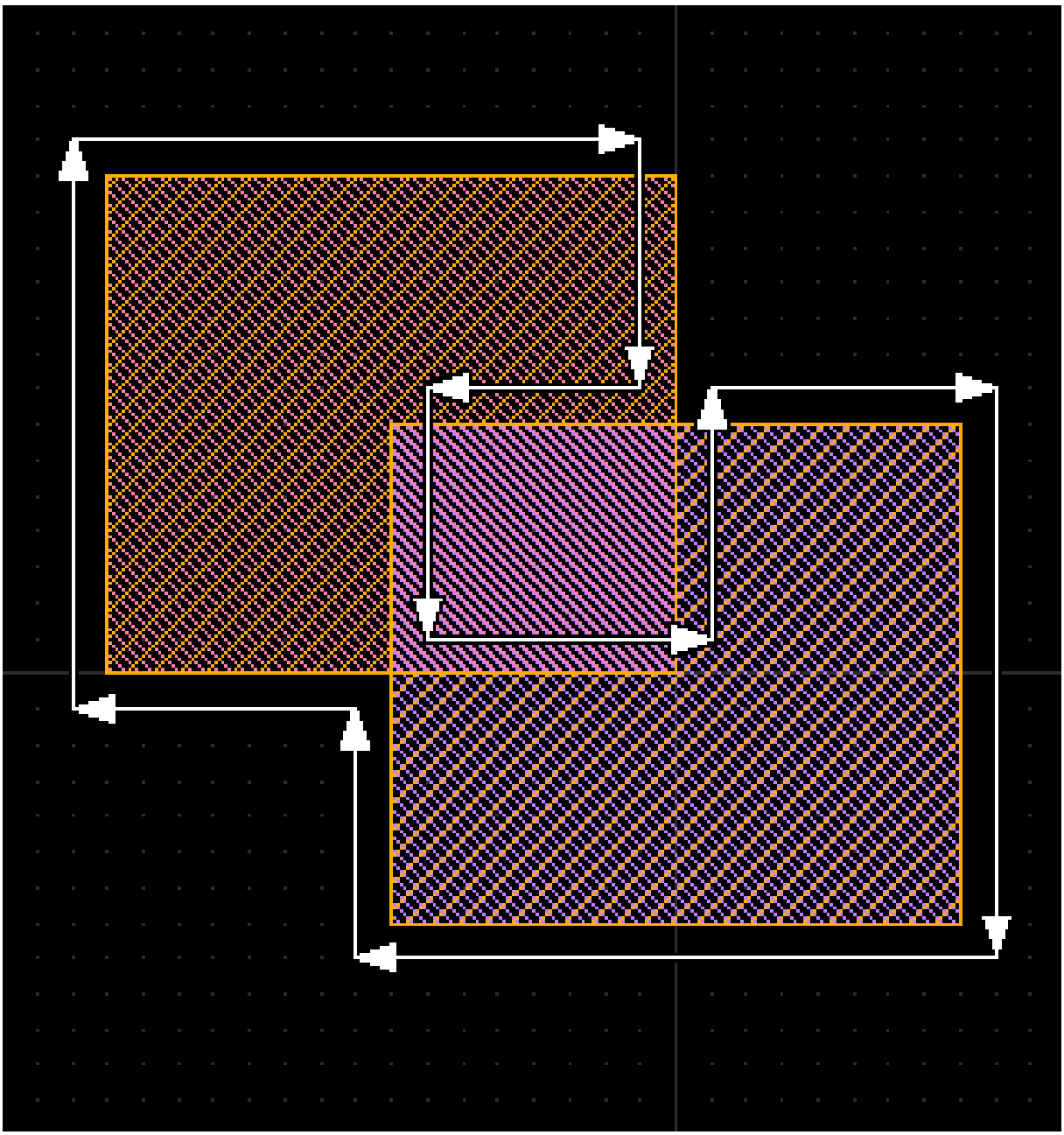
Shapewise Boolean Operations

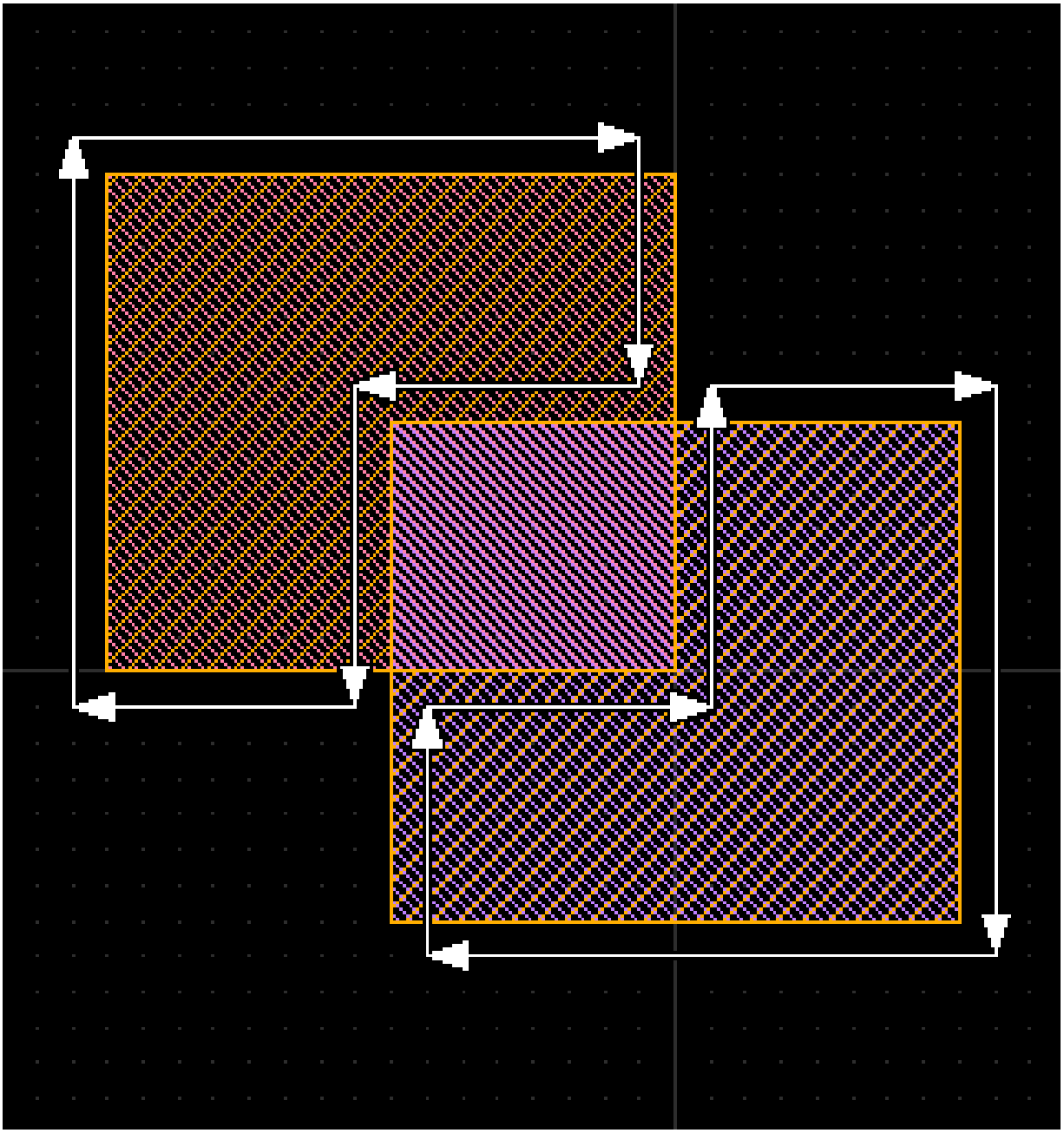
Boolean operations are also available on selected shape sets. These operations use the concept of "primary" and "secondary" selection. The primary selection contains all shapes that are selected in the first step. The secondary selection contains all shapes that are selected in additional steps using the "Shift" modifier key.

The following operations are available in the "Selection" submenu of the "Edit" menu:

- **Merge**: merge all shapes in the primary and secondary selection and write the results to the layer of the primary selection.
- **Intersection**: Compute the intersection (AND) of primary and secondary selection and write the results to the layer of the primary selection.
- **Subtraction**: Compute the difference (A NOT B) of primary (A) and secondary (B) selection and write the results to the layer of the primary selection.

Note: traditional "cut" or "chop" functions may have to be done using the Subtraction function and a "cutter" object on the same layer as the to-be-cut object





The boolean operations are currently implemented flat and based on a full-level edge representation. This means, that the complete layer is flattened (if "flat" mode is requested) and converted into a set of edges which the processor runs on. This will lead to huge resource requirements for very large layouts and is not recommended for such applications currently.

The boolean processor is based on an iterative approach to cover grid snap effects which makes it highly accurate but somewhat slower than a single-pass scanline implementation. Performance penalty is about 2x (two times slower) compared to an efficiently implemented single-pass algorithm.

Shapewise Sizing

The selected shapes can be sized with a given enlargement and shrink distance, similar to the layer operation but with less options. The sizing function can be found in the "Selection" submenu of the "Edit" menu. A dialog will open that prompts for the sizing value (one value for same sizing in x and y direction in micron or two comma-separated values for different sizing in x and y direction).

Object Alignment

This operation use the concept of "primary" and "secondary" selection. The primary selection contains all shapes that are selected in the first step. The secondary selection contains all shapes that are selected in additional steps using the "Shift" modifier key.

The object alignment function allows aligning of all objects in the secondary selection to the objects in the primary selection (i.e. objects in the primary selection define the reference points but are not moved). An "object" can be a shape or an instance of a cell. Cell instances are referred to by their bounding box which can be either computed from the visible layers alone or from all layers.

Alignment can be specified differently in horizontal and vertical direction. Horizontal alignment can be "none" (no change), "left" (align left sides), "center" (align centers) or "right" (align right sides). Vertical alignment can be "none", "bottom", "center" or "top".

The alignment function can be found in the "Selection" submenu of the "Edit" menu. A dialog will open which allows specification of the alignment mode and bounding box computation mode for cell instances.

Corner Rounding

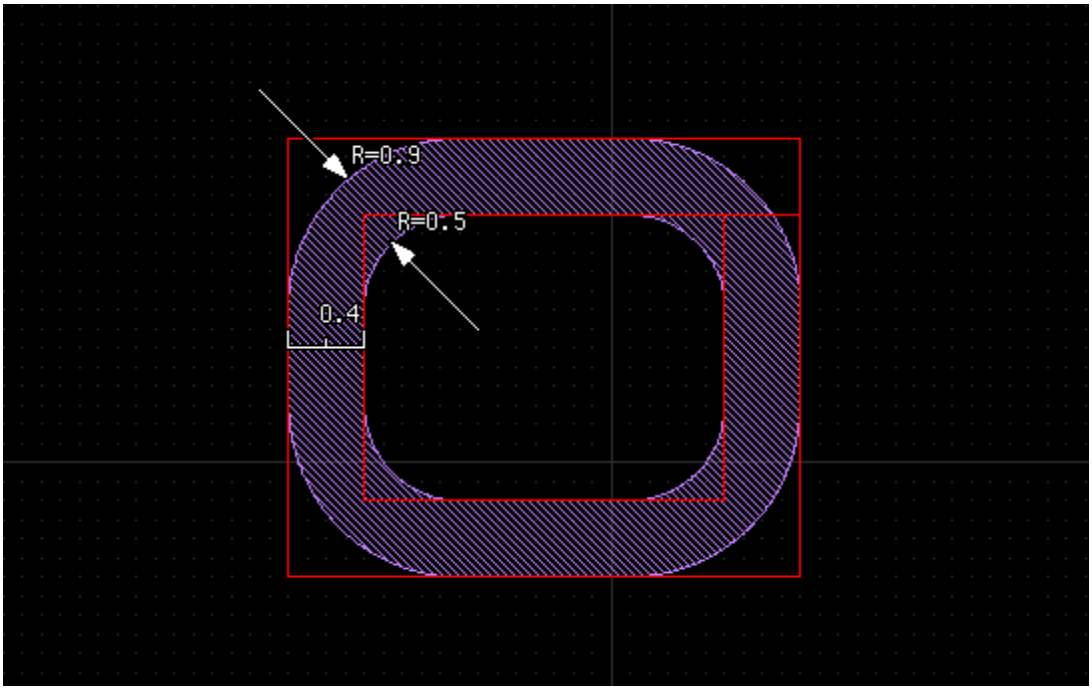
In some applications, i.e. power devices, it is desirable to have round corners instead of sharp corners to limit the electrical field. Klayout now offers a convenient way to create such structures. The basic idea is to draw the structures with sharp, 90 degree corners and then "soften" the corners by rounding them to a given radius. The resulting polygons can then be

written to GDS files, even though GDS does not have the concept of "soft" (or circular) geometries.

The interesting part is: **the corner rounding function can be re-applied on such geometries on a polygon basic**. That means, that even if such a modified polygons are saved to GDS or otherwise modified, the original geometry can be reconstructed and the corner radius can be changed. No special geometrical objects or special GDS annotation is required to achieve this. This requirement imposes some (probably weak) limitations:

- The number of points per corner must not be too small (currently at least 32 on the full circle)
- The original geometry must not exhibit sharp corners and the original segments must be at least twice the corner radius in length.
- The corner segments must be perceivable as such, i.e the angle between adjacent edges must be "nearly" 180 degree. This imposes some restrictions on the minimum length of such a segment and on the accuracy by which they can be expressed in database units. This boils down to a certain length limit in terms of database units.

The following screenshot illustrates the round corners function. As can be seen in this example, it is necessary to allow a different radius specification for "inner" and "outer" corners.



The corner rounding function operates on selected shapes. It can be found in the "Selection" submenu of the "Edit" menu. A dialog will open which allows specification of the radius values and the desired resolution. If the selected polygon already has rounded corners, the corner rounding will be removed and the original polygon reconstructed before the new corner rounding is applied. By specifying "0" for the radius, the original sharp corners will be recovered.

Cell Origin Adjustment

The cell origin is important for a cell because this point is the instantiation anchor for cell instances. The cell origin adjustment function allows shifting the origin to a certain place relative to a cell's bounding box. This can be either the center, a corner or the middle of an edge of the bounding box. The bounding box can either be computed from all or just from the visible layers.

If the "Adjust instances in parents" check box is clicked, the instances of the cell will be adjusted in the opposite direction. Hence, the cell effectively does not change, but locally the origin of the layout will be shifted.

The cell origin adjustment function can be found in the "Cell" submenu of the "Edit" menu.

Create Cell Variants

KLayout offers a feature that is very useful when you want to edit a single instance of a cell rather the cell itself. Editing a cell means that the changes applied to the cell appear at all places where this cell is placed. That may not be desired - if you want to modify a layout in one particular place you may not want to have side effects at other places.

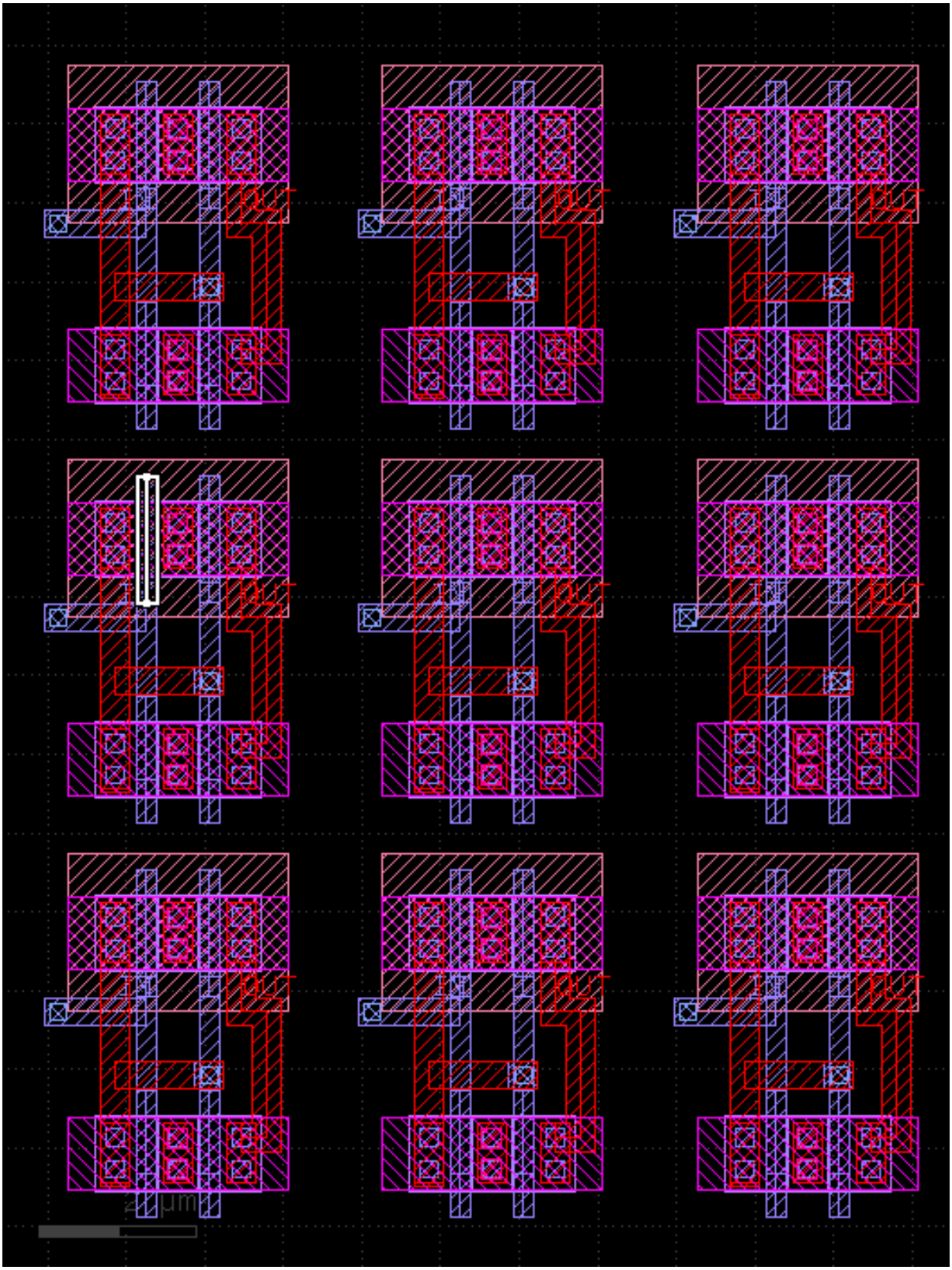
The usual way of achieving that is to copy a cell, replace a particular instance by the new cell and edit the new cell. This can be tedious, in particular if there is an array instance where just one instance must be modified. In that case, the array has to be split so the single instance is isolated before it can be replaced with a new cell.

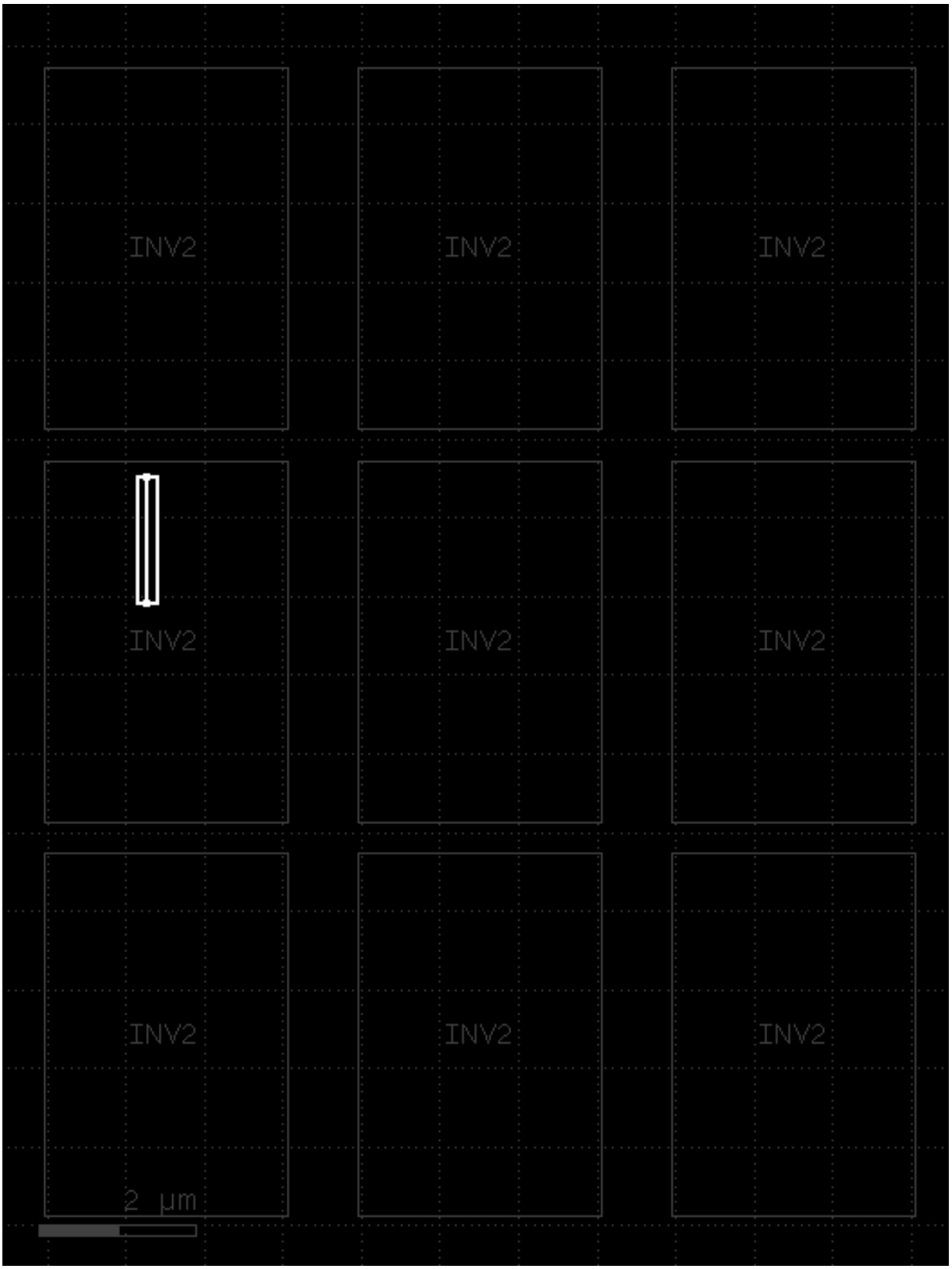
KLayout offers a feature that automates this task. It is found in the "Edit/Selection/Make Cell Variants" menu. It basically works like this: For all selected objects it will follow the hierarchy up to the current cell. It will create new cell copies for all cells found along that path and use these new cells instead of the original ones.

The effect is, that after using "Make Cell Variants", the selection can be modified (i.e. deleted) without having any undesired side effects.

This feature can also transform array instances and isolate certain instances of the array. The following screenshots demonstrate that feature.

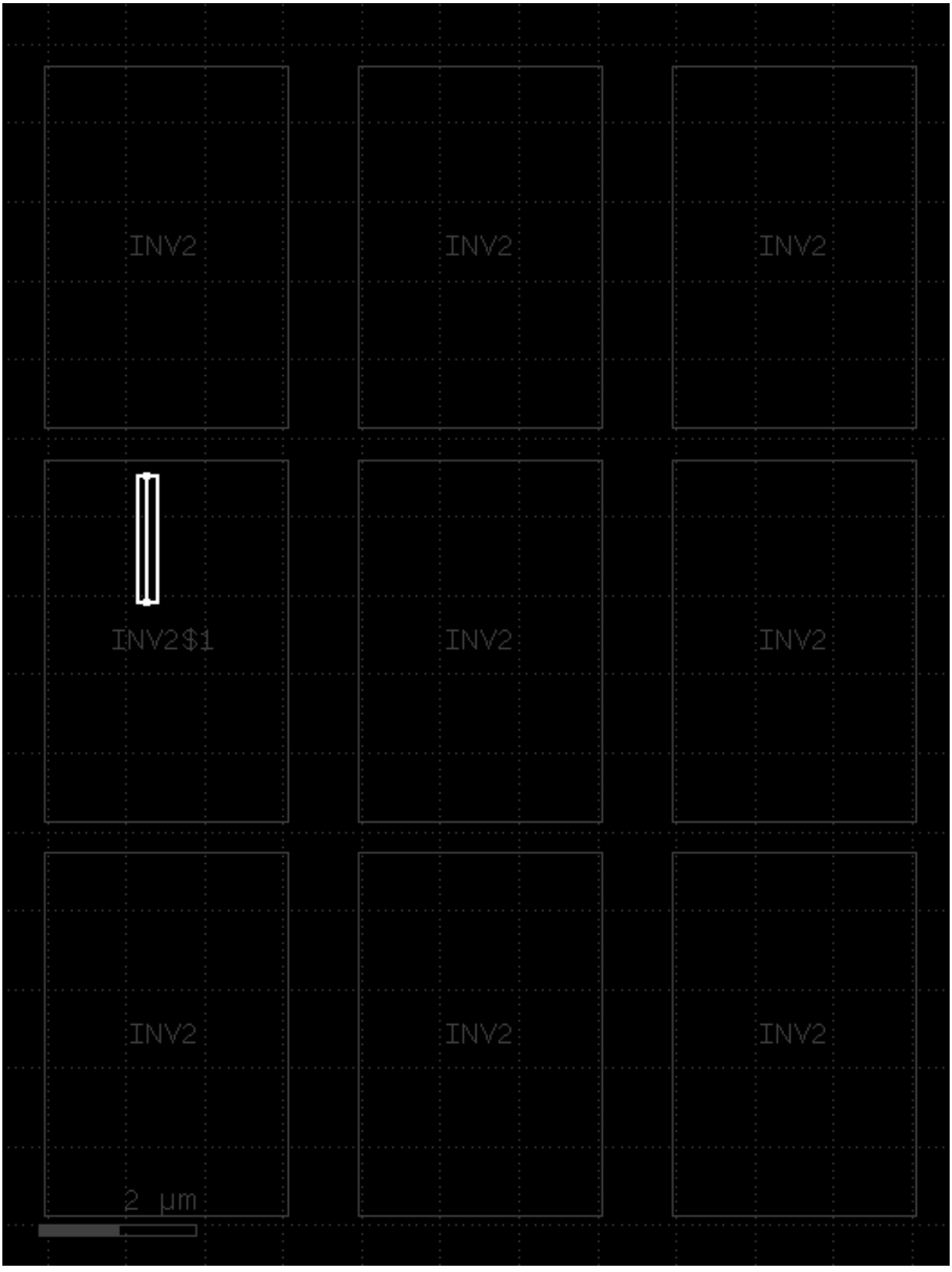
This is the initial situation: a cell is instantiated 9 times in a 3x3 array and one shape inside one of these instances is selected. The two following screenshots show the full-level hierarchy view and just the top level hierarchy to demonstrate that the cell is placed 9 times.


OUT
OUT
OUT
OUT
OUT
OUT
OUT
OUT
OUT
2 µm

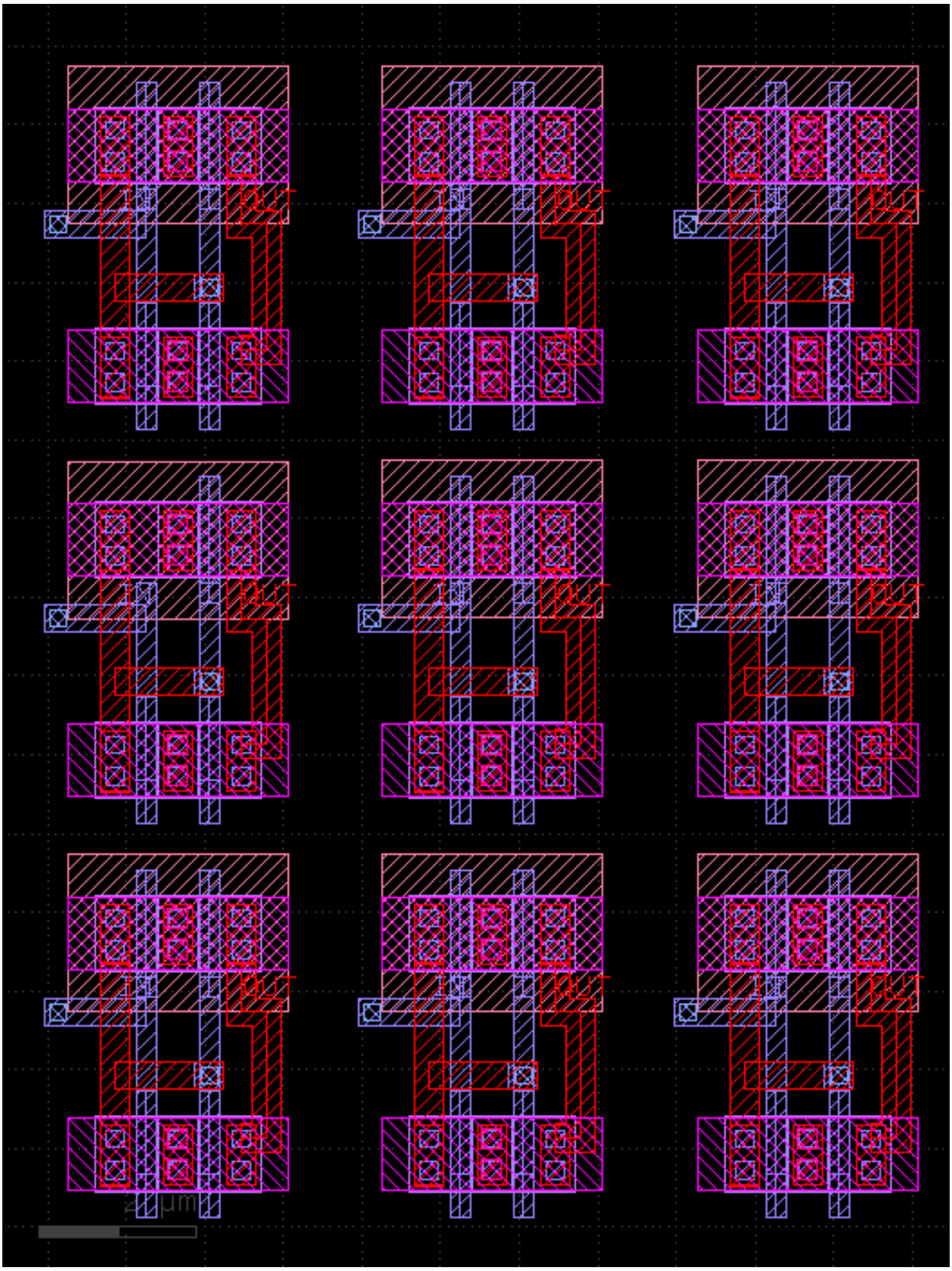


If that shape would be deleted, it would disappear in all instances. Imagine that is not intended. You can now use the "Make Cell Variants" feature to create a cell variant for the one instance

that we have selected the shape in. After we did so, one instance is replaced by a copy of the cell which is called "INV2$1". The picture looks like this:



We can now delete the single shape without any side effect on the other instances:



End of Editing Functions section

Start of Advanced Topics section

Advanced Topics

This section describes some more advanced features of KLayout:

The XOR Tool

The XOR tool performs a geometrical XOR (also ANOTB and BNOTA for asymmetric differences) on two layouts by performing the respective boolean operations layer by layer. The XOR tool is started using "XOR Tool" from the "Tools" menu). Currently, the tool compares all or just the visible layers. Currently, it compares layers from one layout vs. the identical layers from the other layout.

The current implementation employs a flat XOR processor. This limits the application somewhat to small and medium sized layouts and does not make use of hierarchy, which basically excludes applications for very hierarchical layouts (i.e. memory arrays). The memory footprint associated with the flat approach can be mitigated by using the tiling feature which performs the operation on a tile with limited size. This does not reduce the run times but the memory requirements.

The XOR tool allows specification of tolerances. Basically a tolerance is an undersizing step following the boolean operation. This way, small markers can be suppressed. This is particular useful to remove markers resulting from tiny differences between the layouts being compared. Multiple tolerances can be specified. In that case, multiple undersize steps are performed to create sets of layers with different tolerances each. For example, a tolerance specification of "0,0.001,0.005,0.010" will create four sets (marker categories) containing all difference markers and others for markers indicating differences larger than 1nm, 5nm and 10nm.

Tiling can be enabled by entering a tile size into the entry box. For semi-flat layouts such as standard cell blocks, a tile size of 1000 micron is a good starting point. The choice of the tile size mainly determines memory requirements.

The XOR tool allows sending the output either to a marker database or to another or one of the input layouts. The mode can be selected with the "Output" drop-down box. If output is sent to one of the original inputs, it is mandatory to specify a layer offset which maps the original layer to a new layer. An offset of "1000/0" for example means, that differences between shapes on layer "16/0" will be sent to "1016/0" for the first tolerance category and "2016/0" for the second.

Finally, the XOR can be confined to a region. This saves time if differences in parts of the layout are of interest. To select a region use the drop-down box next to "From region":

- **All:** Do an XOR on the whole area (default)
- **Visible region:** Confine the XOR to the visible part of the layouts
- **Clipped to ruler:** Draw one or many rulers to specify the region to which the XOR shall be confined. Each ruler specifies a region given be the extension of the ruler. If you use a box-type ruler, the ruler gives a better visualization of the region

The Diff Tool

As the XOR tool, the Diff tool performs a comparison of two layouts. In contrast to the XOR tool, it does a cell-by-cell and object-by-object comparison and reports differing cells, instances and geometrical objects. In effect, the comparison is more strict and not purely geometry-related. It does not verify the identity of the layouts on mask level but rather the exact identity of the objects that comprise the layout file. On the other hand, the Diff tool usually detects the actual changes rather than their effect on geometry.

Usually, that kind of comparison is very sensitive to "cosmetic" changes, i.e. cell renaming. KLayout's Diff tool tries to mitigate this effect with these features:

- Before it does the cell-by-cell comparison it tries to detect cells which have been renamed by comparing their instantiation. That way, it can compare the right cells even though their names may be different. The basis of that functionality is a cell matching algorithm. This algorithm compares cells by taking into account their bounding boxes, shape counts per layer, number of instances and other parameters. The algorithm will choose a partner cell which matches closest with respect to these parameters. If that scheme fails, it is possible to revert to name matching by unchecking the option "Don't use names to match cells".
- It allows some level of control over the strictness of the compare. For example, cell arrays can be expanded before the individual instances are compared. By default, some second-order information like users properties or certain text properties is not compared.
- The diff tool can also work in "XOR" more. In that mode, the differences found are used to provide input for a subsequent, polygon-only XOR step. The result is a fair approximation of a true, as-if-flat XOR which delivers a superset of the true XOR's results. It may report some locations as being different which if fact are not, but it will not fail to report differences where there are some. Compared with the XOR tool's

functionality, some options are missing (i.e. tolerance), but the performance is much better.

The Diff tool is found in the "Tools" menu. In this dialog:

- Select layout A and B in the "Input" section.
- Uncheck "Don't use names to match cells" to revert to pure name matching. Cells which have been renamed will not be compared against then.
- Check "Run XOR on differences" to select the "XOR mode".
- Check "Summarize missing layers" to have missing layers reported as one difference instead of one per shape.
- Check "Detailed information" to receive detailed information about every difference. Without that option, only the number of differing shapes or instances is reported.
- Check "Expand cell arrays" to compare individual instances of array instances.
- Check "Exact compare" to include second-order information (i.e. user properties, text orientation) in the compare.

The Diff tool will create a marker database and show the results in the marker database browser.

The Fill (Tiling) Utility

The fill utility creates a regular pattern of fill unit cell instances in certain areas of a layout. This feature is usually referred to as "tiling" or "fill". It is based on a rectangular unit cell which is repeated in x- and y-direction to fill the available space. In most cases, the intention is to fill empty areas in the layout to enhance the layout uniformity for a better process performance.

Before the fill utility can be used, a fill cell must be prepared in the layout that is filled. The dimension of the cell are defined by a box drawn on an arbitrary layer. This box must represent the "footprint" of the cell. This is the space that one instance will cover in the region to be filled.

The fill utility is found in the "Utilities" sub-menu of the "edit" menu and is available in edit mode only. To use this utility, open the dialog and

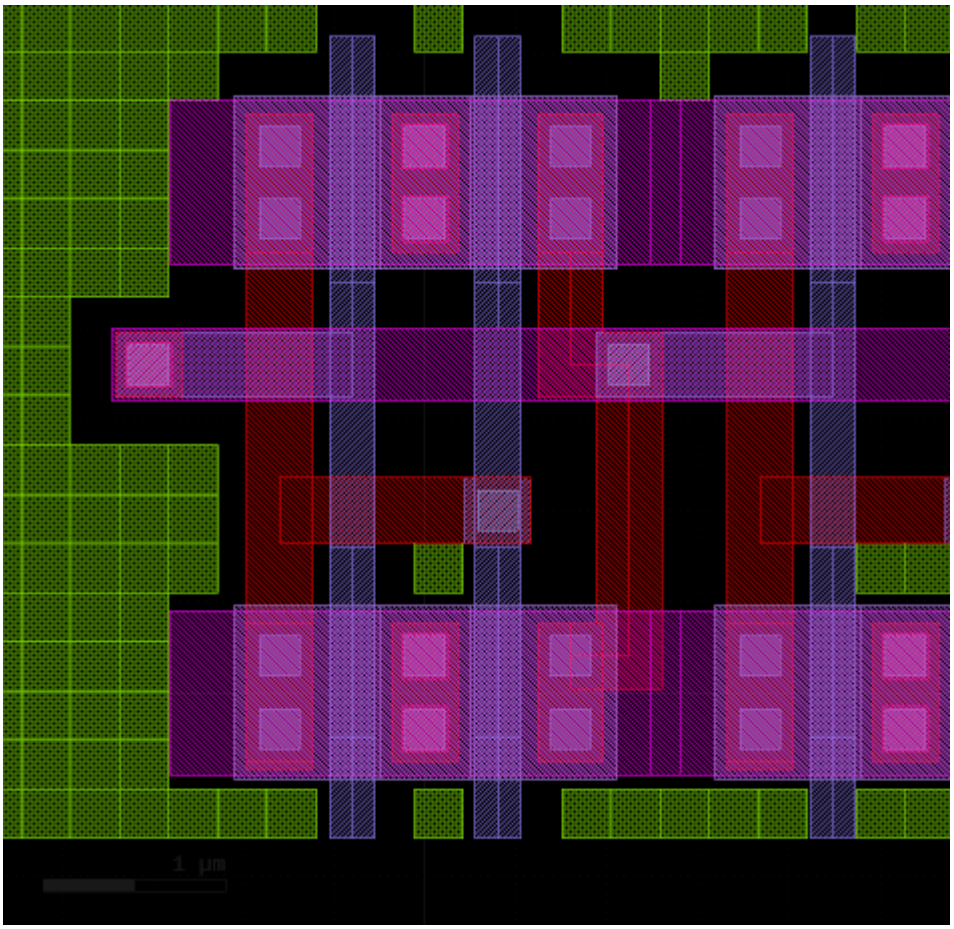
- Select the outer boundary of the fill region ("what to fill"). Available choices are: Full cell, the interior or the polygons on a given layer, the interior of all selected polygons, a single box or an area defined by a ruler.
- Specify if the fill area should keep a certain minimum distance to the border of the fill region.
- Specify the regions within the fill region which must not be filled. Available choices are: All layers (don't create fill over any polygon drawn), all visible layers (don't create fill over any polygon visible), all selected layers or don't exclude anything.

- If the fill tiles must keep a certain minimum distance from the exclude regions, specify that distance in the "Spacing around exclude areas" entry field.
- Specify the fill cell and the boundary layer which defines the cell's footprint in the "Fill Cell" group.

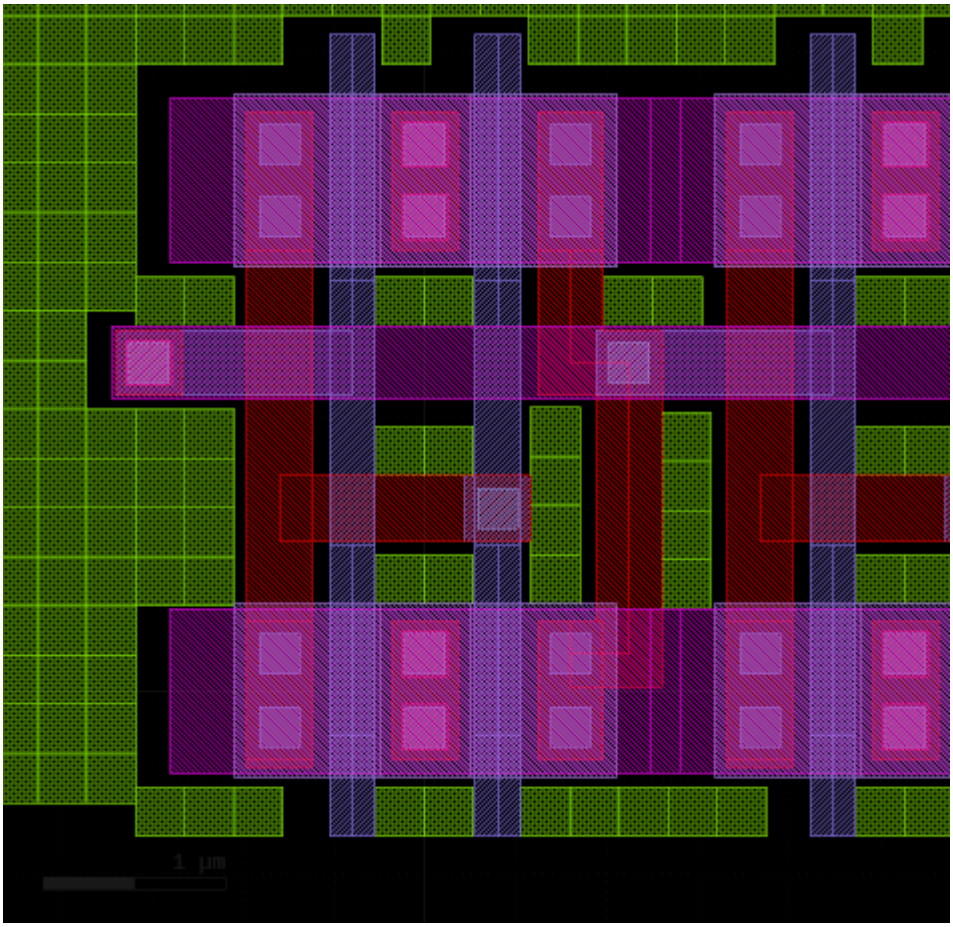
By default, the fill utility operates on a fixed raster. This can lead to a poor fill efficiency in some cases. The fill utility offers a "Enhanced fill" option, where it tries to find a cell arrangement which is not necessarily on a common raster but provides a better fill performance. In addition, second-order fill is supported. In that case, a second - usually smaller - fill cell can be specified which is used to fill the remaining areas of the layout.

The following screenshots show the effect of the different fill modes for some artificial fill problem.

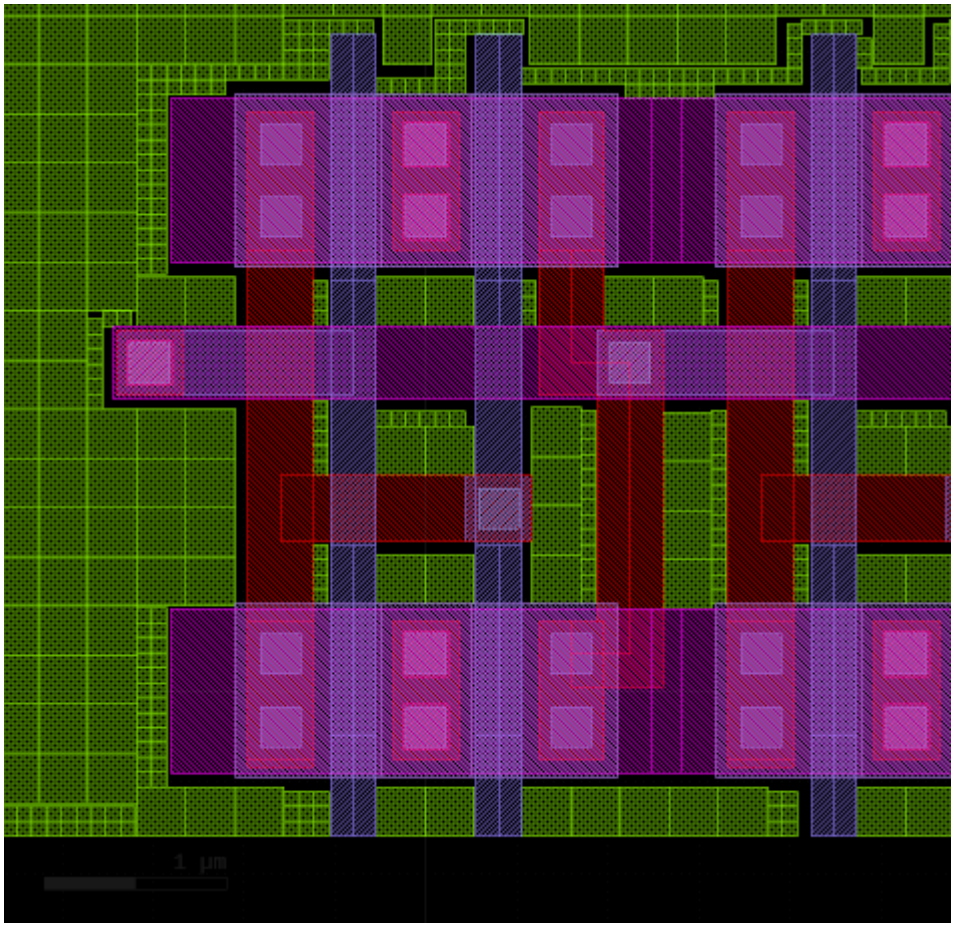
Default:



Enhanced:



Enhanced plus second order:



Import Gerber PCB Files

- The import dialog
- The layer stack flow
- The free layer mapping flow
- General options

Gerber PCB import allows creating GDS layout data from Gerber PCB files or to add Gerber files to GDS files as new layers. The import function supports a majority of the RS274X features for artwork files and a couple of different formats for the drill files. The importer will take a set of files and convert them to layout geometry. The importer offers some functionality to adjust the data appropriately, i.e. to define output layers and apply geometrical transformations. Another basic capability is to merge the geometry of a layer to remove overlaps and join paths into larger polygons.

Because of the manifold options, the import specification can become pretty complex. Therefore, it can be saved into a file (suggested suffix is ".pcb") in XML format which contains the importer specifications. Once such a file is created, KLayout can read this file like usual stream files (i.e. it can be specified on the command line) and use it as a recipe to import the associated Gerber files.

The PCB import function is available in the "File" menu ("Import/Gerber PCB"). Different entry points are provided that start a new project, open an existing project or to continue with the last project.

The basic workflow to import PCB data is:

- Specify the directory where the PCB data files are located (the "base" directory).
- Specify the import mode (the destination of the layout data).
- Decide about the layer mapping mode: free layer mapping or metal stack mapping. Free layer mapping allows an arbitrary mapping between PCB layers and GDS layers. This specification is the most flexible one but is tedious to enter. Metal stack mapping is easier to specify but confined to mapping a set of PCB files to a metal/via/metal stack scheme.
- Specify the files, GDS layers and PCB to GDS layer mapping.
- Specify a transformation if desired, either by specifying mapping points or a transformation directly.
- Decide about further options (i.e. merging, database unit, top cell name etc.).

The basic decision is how to specify the layer mapping. In free mode, the specification requires these steps:

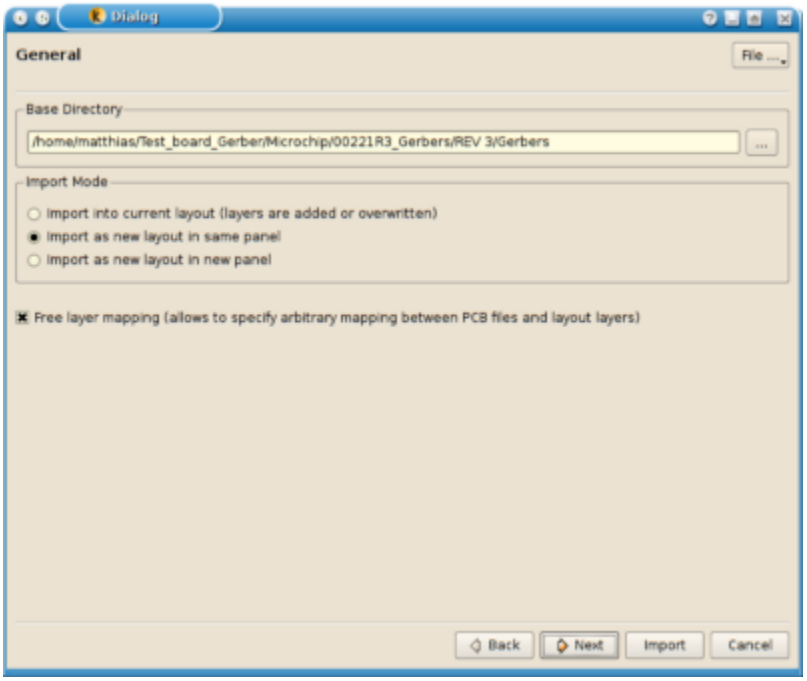
- Specify the files to load (in the dialog on the "Files" page).
- Specify a list of output layers.
- Fill the input to output mapping matrix which assigns one or many output layers to each input file.

In layer stack mode, the specification workflow consists of these steps:

- Specify the GDS layer stack (the complete stack available for mapping PCB data into). The idea is basically to put another set of series of metal/via/metal layers on top of the GDS layer stack. The PCB layer closest to the die surface is placed into the first metal layer which is supposed to be the first above the on-chip layers.
- When the GDS layers are set, specify how many metal and drill hole files the PCB file set contains and whether the chip will be mounted on the top or bottom of the PCB. The latter decides in which order the PCB layers are assigned to GDS layers (remember, the first GDS layer will be the PCB layer closest to the die surface).
- Enter file names for the artwork files corresponding to metal layers.
- Specify file names for the drill files and what metal layers are connected by the (plated) drill holes. Since a drill hole can connect multiple layers in the stack, a connection information is always of the type "from metal, to metal" with the drill holes connecting all metal layers between "from" and "to".

The import dialog

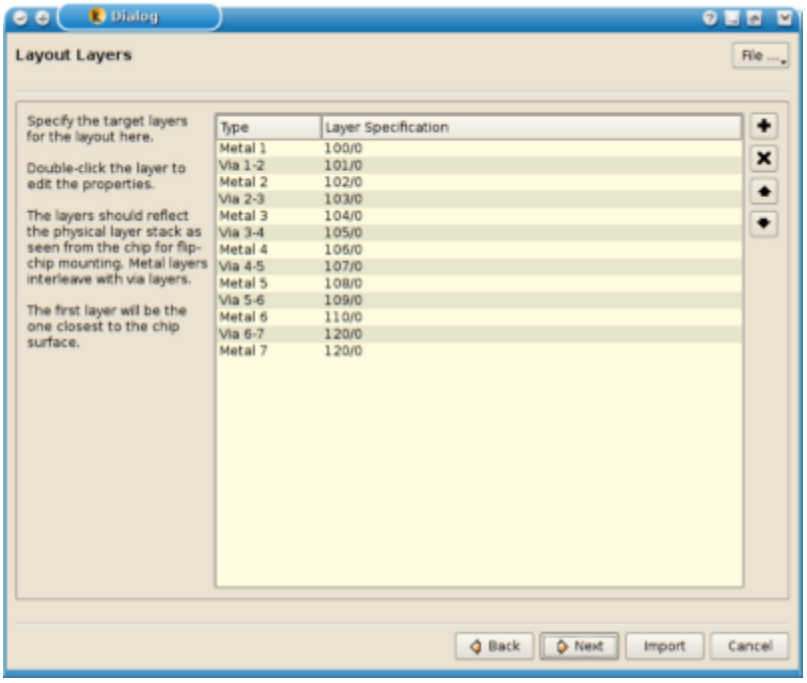
The import dialog is organized in multiple pages that reflect the workflow for the import specification. On every page, the "File" menu button allows saving the current settings as a PCB import project, to open an existing project or to create a new project and to restart from scratch.



The first page offers some basic options:

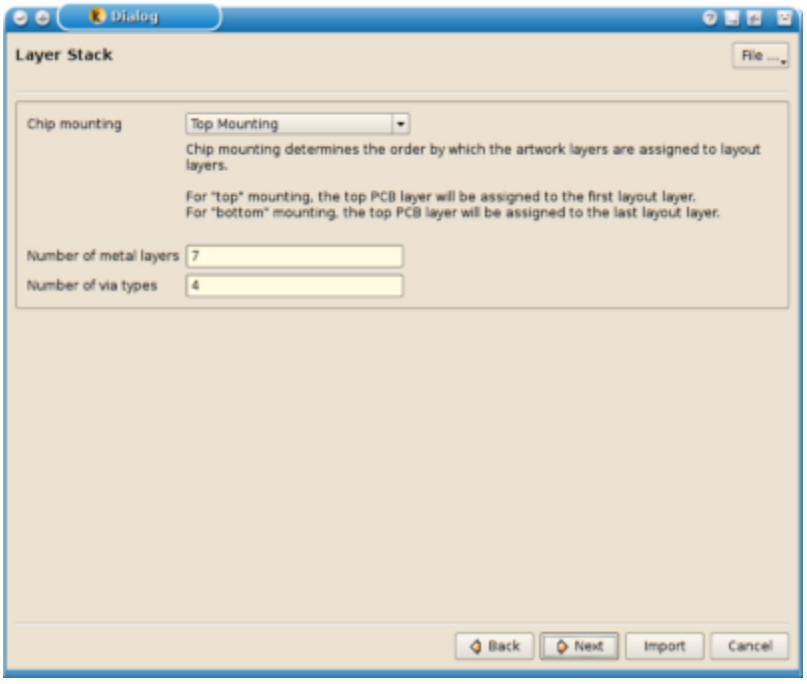
- **Base directory:** this is the directory where all the PCB files are found. Not necessarily all files must be located there but are looked for relative to this directory. If all files are moved, just the base directory must change. The base directory is not stored in a project file. Instead, the base directory is the directory where the project file is stored. Basically this implies, that all data files will be referred to relative to the project file.
- **Import mode:** PCB data can be imported into the current layout (into the current cell). Usually, it this case, layers will be added to the current layout. Alternatively, a layout can be created which will be either placed into a new panel or added to the current panel.
- **Layer mapping mode:** Specify here whether to use free or layer stack mode. Check the box to use free layer mapping mode.

The layer stack flow

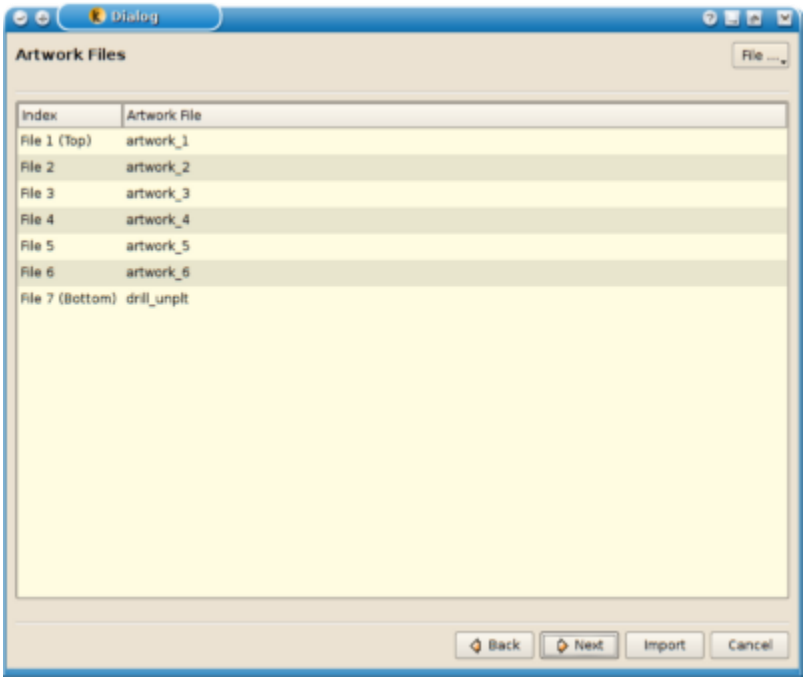


In the layer stack flow, on the first page, a sequence of metal and via layers must be specified. The assignment of metal and via layers is done automatically. The sequence is always a metal layer followed by a via layer. The number of layers must be odd so the last layer is a metal layer again. Via layers will connect the adjacent metal layers only.

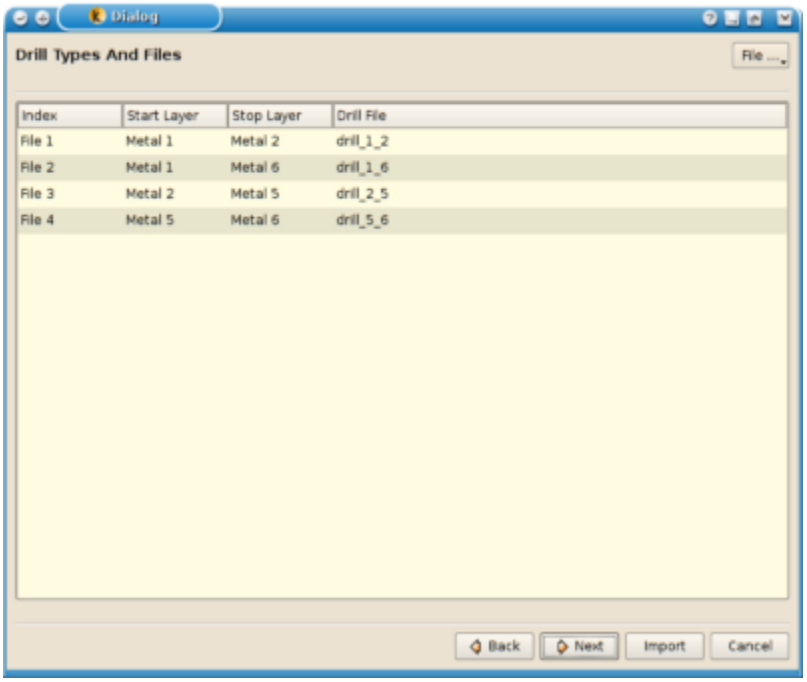
Use the "+" button to add new layers. Move layers by selecting them and moving them up or down with the arrow buttons. Use the "X" button to remove all selected layers.



On the next page, the number of artwork and drill files needs to be specified. Later, the actual files need to be entered and assigned to metal or via layers. In addition the chip mounting position needs to be specified. In "top mounting" mode, it is assumed that the chip is placed surface down on the top (first) PCB layer. Thus the first metal above the chip stack will be the top PCB layer. In "bottom mounting" mode, the last PCB metal layer will be the first metal layer above the chip stack.

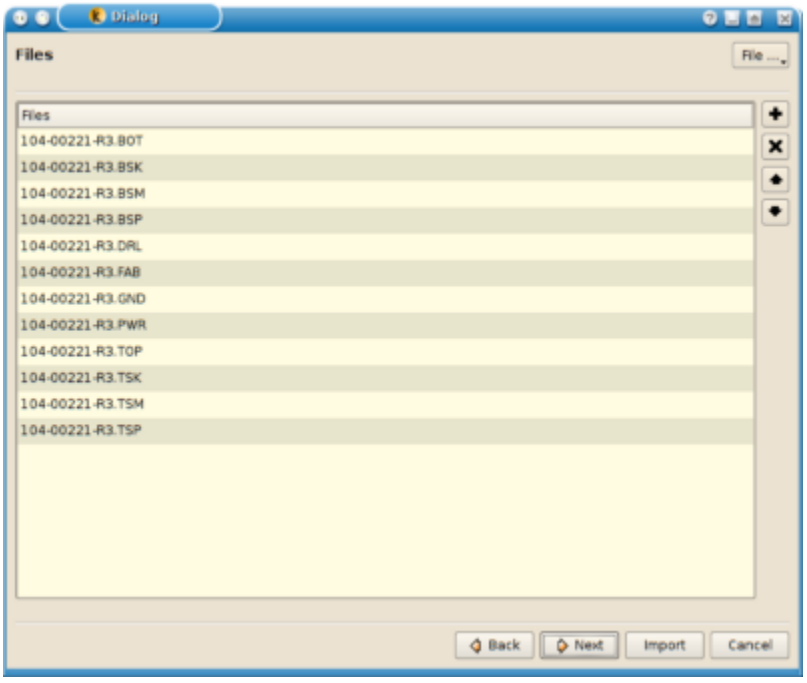


On the "Artwork Files" page, the artfile file names must be entered. They are automatically assigned to the respective metal layers. The assignment order depends on the mounting mode.

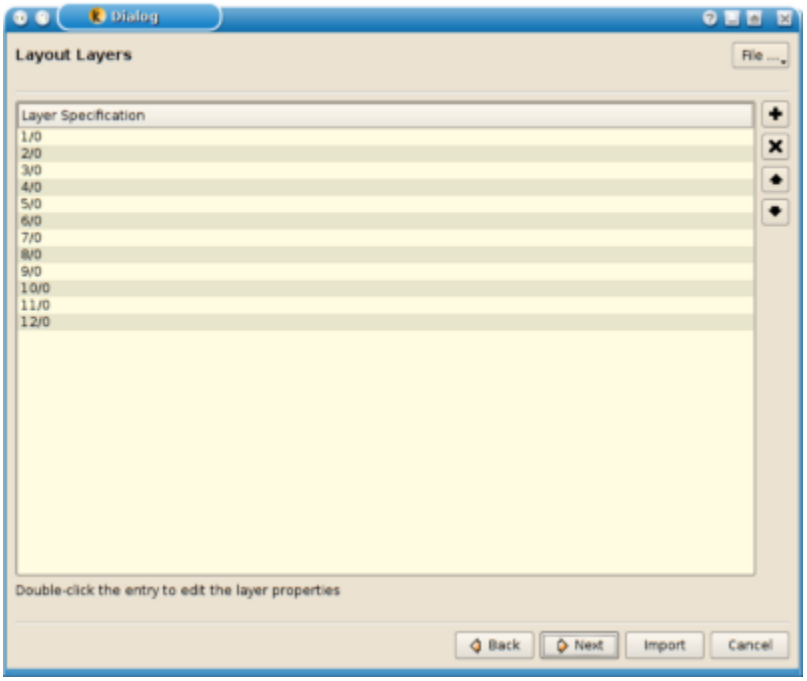


On the "Drill Files" page, the drill file names must be entered. Each drill file describes a certain drill step, which can connect multiple metal layers. On this page, this specification must be made. The first and last metal layer connected by the plated hole must be specified. The corresponding via layers will then be used to create via shapes.

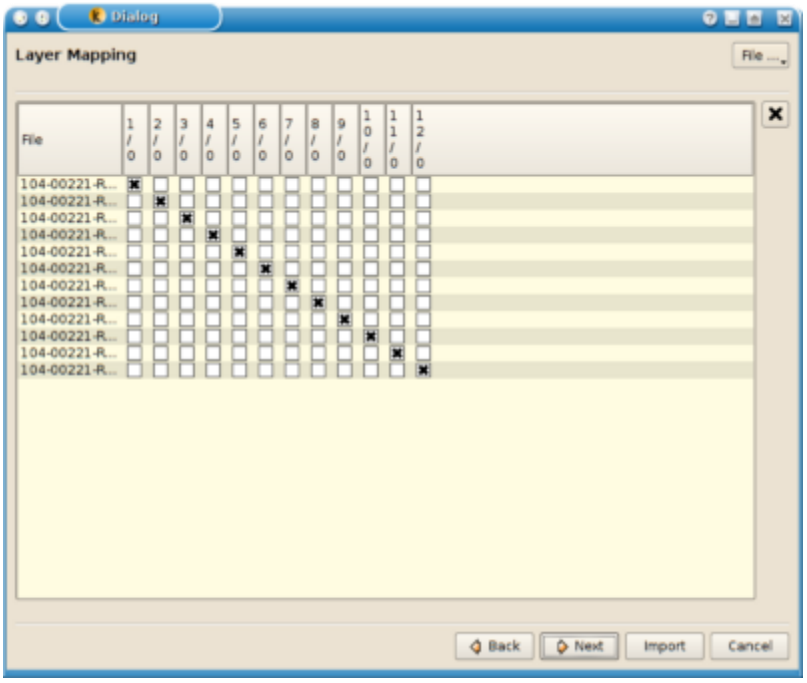
The free layer mapping flow



On the "Files" page, all PCB data files must be specified. This includes artwork and drill files. The order is not important but it is recommended to follow the physical stacking. This simplifies the assignment to GDS layers later. Use the arrow buttons to move the selected entries up or down. Use the "X" button to delete files from the list and use the "+" button to add new files.

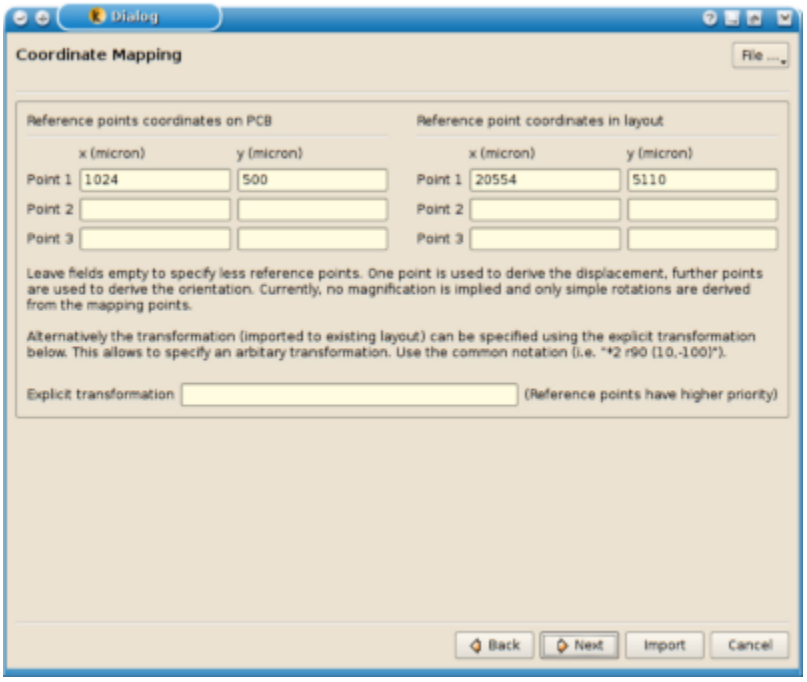


On the "Layout Layers" page all target layers must be specified. Provide a list with all layers that are used as target layers for the import. Again, the order is not important but maintaining a technological order will simplify the assignment in the next step. As on the previous page use the arrow buttons to move selected entries and the "+" and "X" button to add new entries and deleted the selected ones.



On the "Layer Mapping" page, each file can be assigned to one or may GDS layers. The assignment is described in form of a matrix where an "x" means that the file or layer given by the row is imported into the layer given by the column. A file can be imported into multiple layers which basically will duplicate the shapes. Click at the boxes to set or reset the mark. Use the "X" button on the left to reset all marks for the rows selected.

General options

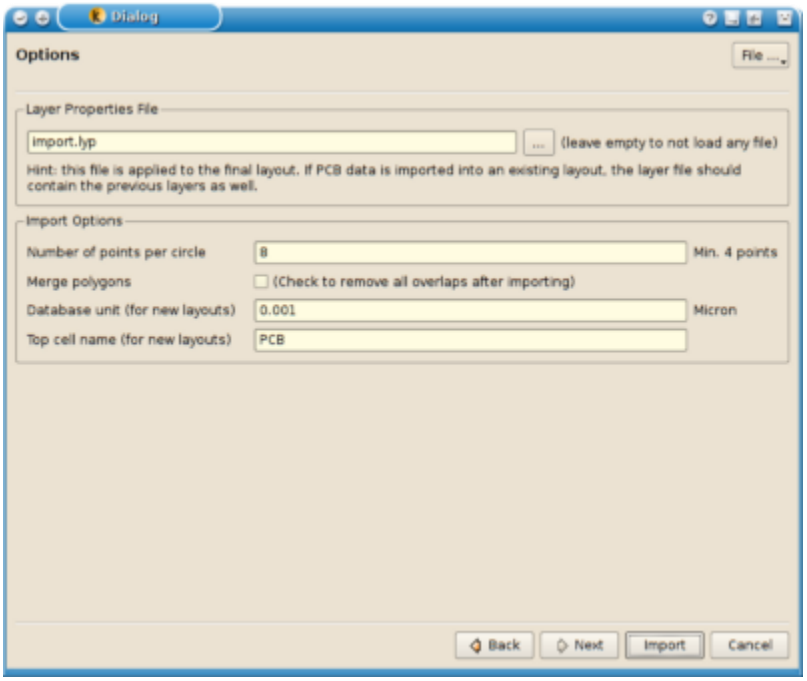


The "Coordinate Mapping" page allows specification of the transformation of the PCB data into the GDS space. Since PCB and GDS rarely share the origin, a transformation can be specified which is applied to the shapes when importing them.

A specification can be made in two ways:

- **By specifying matching points:** The transformation will be computed such that the given PCB coordinates are mapped to the given GDS coordinates. Up to three coordinate pairs can be given. If one coordinate pair is given, a displacement is derived. If two coordinate pairs are given, the rotation is computed as well (only multiples of 90 degree are supported currently). If three coordinate pairs are give, the algorithm can derive mirroring as well.
- **By explicitly specifying the transformation:** The transformation can be specified explicitly in the lower entry field. The format is "x,y" for a simple translation (x, y are given in micron units), "rx" or "mx" for a rotation by the angle "x" or mirroring at the line with angle "x" and "*x" for a magnification of "x". All specifications can be combined, i.e. "r90 170,-5100" specifies a rotation by 90 degree and displacement by 170 micron in horizontal and -5.1 mm in vertical direction.
  For a comprehensive description of that string, see Transformations in KLayout.

Hint: Both specifications can be combined, i.e. one coordinate pair can be given to define the displacement and the rotation can be specified explicitly.



Finally, on the "Options" page, various options can be set:

- **Layer properties file:** If specified, this layer properties file will be loaded after the layers have been imported. The file is specified relative to the base directory.

- **Number of points per circle:** KLayout resolves the circular apertures commonly used in PCB layout into polygons to perform geometrical operations. This options allows choosing how many points will be used for the approximation of a full circle. Less points will mean less accurate representation but smaller polygons hence better performance on boolean operations used to compute clear areas for example.
- **Merge polygons:** If this option is set, all polygons will be joined if they overlap or touch. Note, that merging also happens implicitly if clear layers are used because the boolean operations used to cut out clear regions will implicitly merge the previous layout. This implicit merging cannot be disabled.
- **Database unit and top cell name:** This option allows choosing the database unit and top cell name for new layouts. This applies only, if the import mode implies a new layout.

Import Other Layout Files

This function can merge other layouts into the layout loaded. Merging means that the hierarchy of the specified layout is inserted into the given layout. Different modes are available that control the way how the hierarchy is merged. This function is available in the "File" menu as "Import/Other File Into Current".

The workflow for importing a different layout is this:

- Specify the file to input. At least the file name is required. Additionally, a cell can be specified. In that case, only the cells referred to by the given cell (directly or indirectly) are imported. Reader options can be specified separately for the import. Reader options are applied the same way than the reader options are used for the standard load function.
- Specify the import mode. The modes are described below.
- Specify the layer mapping. Either the shapes are imported on their original layer or an offset can be used that will be added to the layer to form the target layer of the import. An offset of "1000/0" for example specifies to add 1000 to the layer and use the original datatype.
- Specify an optional transformation. The imported layout will be transformed accordingly. The transformation can be specified explicitly or with up to three points which are mapped onto each other.

Four import modes are available that control how the hierarchy of the imported layout is inserted into the existing layout:

- **Merge**: in this mode, the contents of the imported cell will be put into the current cell and the child hierarchy is added below the current cell.
- **Extra**: in this mode, new top level cells containing the hierarchy tree of the imported cell or cells will be created. In this mode, multiple cells can be imported if the imported layout contains multiple top cells. Leave the cell specification empty for this.

- **Instantiate**: the imported cell will be instantiated into the current cell as a separate hierarchy.
- **Merge hierarchy**: The fourth mode is a little bit more complex. Basically it works like "Merge", but identifies corresponding cells and merges the contents for the corresponding imported cells into the original cells. The algorithm identifies corresponding cells by requiring that the flat instances of the imported child cell exactly equal the flat instances of the corresponding original cell (where flat refers to the instances of a cell in the context of the current cell). This is done by selectively thinning out the candidate list and finally employing a name similarity measure to resolve ambiguities.

The import function will create new cell names using the "$x" suffix to avoid name ambiguities.

The Net Tracing Feature

The net tracing function allows tracing of a net by detecting touching shapes that together form a conductive region. It features specification of a stack of metal (or in general "conductive") layers optionally connected through via shapes. The net tracing algorithm will follow connections over the via shapes to form connections to other metal layers. The material footprint can be derived from single layout layers or a boolean combination of several layers. For example, this allows selecting source and drain regions of transistors by subtracting the poly region from the active area region.

Single net tracing

This algorithm is intended for extracting single nets and employs an incremental extraction approach. Therefore extraction of a single small net is comparatively fast while extraction of large nets such as power nets is considerably slower compared to hierarchical LVS tools currently.

The net tracing function can be found in the "Tools" menu. The user interface allows tracing of multiple nets which are stored in a list of nets extracted. If labels are found on the nets, these are used to derive a net name. Beside that, the cells which are traversed in the net extraction are listed, so the cells being connected by this net can be identified.

Before nets can be extracted, a layer stack must be specified. Press "Layer Stack" on the user interface to open the layer stack dialog. Layers must be specified in the "layer/datatype" notation. The via specification is optional. If no via layer is specified, both metal layer shapes are required to touch in order to form a connection. If a via layer is specified, a via shape must be present to form the connection.

KLayout allows specification of symbolic layers and to use boolean expressions. That way it is possible to assign meaningful names to layers (i.e. "POLY" or "VIA1") and to use derived layers (i.e. "ACTIVE-POLY" for the source and drain regions of a CMOS transistor). Read more about these features in Connectivity and Symbolic Connectivity Layers.

(These sections are inlined, next; also appear in LVS documentation)

If a layer stack has been defined, a net can be traced by pressing the "Trace Net" button and clicking on a point in the layout. Starting from shapes found under this point, the net is extracted and listed in the net list on the left side of the net tracing dialog. If "Lock" is checked, another net can be traced by clicking at another point without having to press the "Trace Net" button again.

The "Trace Path" function works similar but allows specification of two points and let the algorithm find the shortest connection (in terms of shape count, not geometrical length) between those points. If the points are not connected, a message is given which indicates that no path leads from one point to the other.

The display of the nets can be configured in many ways. The configuration dialog is opened when "Configure" is pressed in the trace net dialog. Beside the color and style of the markers used to display the net it can be specified if and how the window is changed to fit the net.

Tracing all nets

This algorithm is borrowed from the LVS feature, where the scenario is extended by device recognition and netlist formation. In the context of the net tracer, nets consist of the connected shapes but don't attach to devices. As LVS extracts all nets in one sweep, using this feature in the net tracer will deliver all nets at once. Although this is a richer information output, the tracing of all nets is typically faster then tracing a single, big net such as power nets. The LVS net extractor also supports hierarchical processing which gives a considerable performance improvement and more compact net representations.

To extract all nets, use "Trace All Nets" from the "Tools" menu. It will start extracting the nets immediately. It will take the connectivity definition from the standard, single-net net tracer. You can edit this layer stack either from the single net tracer UI, from the technology manager or "Edit Layer Stack" from the "Tools" menu.

After the net tracer has finished, the netlist browser dialog opens. In this dialog you can:

- Browser the circuit hierarchy (taken from the cell hierarchy) in the left half of the central view.
- Browse the nets of the circuits in the right half of the view. Clicking on a net will highlight the net.
- Configure the net highlighting behavior. Use the "Configure" button.
- Export all or selected nets to layout, save the netlist (with shapes) to a file, load it back from a file and manage the netlist database. Use the "File" menu button in the right upper corner.

- Search for net names (if labeled) and circuits using the search edit box.
- Navigate through the history using the "back" and "forward" buttons at the top left.

Extracted nets are written and read in a KLayout-specific format called "L2N" ("layout to netlist"). This format contains both the nets and the shapes that make up a net. This way, the traced nets can be saved and retrieved later.

Connectivity

Use the connectivity page to specify the conductor layers and their connections. On a conductor layer, all touching or overlapping shapes are connected. A connection is made between conductor layer when the shapes of the two conductor layers overlap. Optionally a via layer can be specified which must be present to make a connection between the two conductor layer.

To specify a conductor layer

- Use "layer/datatype" notation to specify explicit GDS layers and datatypes.
- Enter the layer name to specify either a named layer or a symbolic layer. Symbolic layers must be defined in the symbol table (see Symbolic Connectivity Layers) and can be computed from boolean expressions.
- Instead of using a symbolic layer, enter a expression directly without defining a symbol (see Symbolic Connectivity Layers). Inside the expressions
    - Use "layer/datatype" notation to specify an original layer with explicit GDS layers and datatypes.
    - Use the name to specify a named original layer or to refer to a different symbolic layer defined in a symbol entry.
    - Use the operators '+', '*', '-' and '^' to specify logical OR, AND, NOT and XOR respectively. The precedence of evaluation is '^' and '*' before '+' and '-'.
    - Use round brackets to group expressions.

By creating conductor layers with boolean expressions, it is possible for example to separate an active area layer of a CMOS transistor into source and drain regions by subtracting the gate poly. Symbolic layers are useful to use "speaking" names for layers instead of the numeric layer/datatype specification. Please note, that the net tracer is considerably slower when using boolean expressions.

Symbolic Connectivity Layers

Use the symbol table to specify derived layers and to assign names to layer/datatype combinations. A symbolic layer must have a name which can be used in the connectivity table

instead of the original layers. In addition, an expression must be specified that defines the contents of the layer.

Inside the expressions

- Use "layer/datatype" notation to specify an original layer with explicit GDS layers and datatypes.
- Use the name to specify a named original layer or to refer to a symbolic layer defined in another entry.
- Use the operators '+', '*', '-' and '^' to specify logical OR, AND, NOT and XOR respectively. The precedence of evaluation is '^' and '*' before '+' and '-'.
- Use round brackets to group expressions.

Examples:

17	Abbreviation for GDS layer 17, datatype 0
2/0*17/0	Logical AND between layer 2 and 17, both datatype 0
2*(5+7)	Logical AND between layer 2 and the logical OR combination of 5 and 7
ACTIVE-POLY	Logical NOT between the symbolic layer ACTIVE (defined in another entry) and POLY (also defined in another entry).

End of Advanced Topics section

DRC and LVS topics are put in separate documents

About Expressions

Beside a ruby programming API, KLayout provides support for simple expressions in some places. In particular this feature is employed to generate dynamic strings, for example when deriving the label text for a ruler.

String interpolation

The feature of inserting dynamic content into a string is called interpolation. The Syntax KLayout uses for string interpolation is a dollar character followed by the expression which is evaluated. Simple expressions can be put directly after the dollar character. Others must be put into brackets.

Every dollar expression is evaluated and the expression is substituted by the result string. For example:

String	Evaluates to
`An irrational number: $sqrt(2).`	`An irrational number: 1.4142136`
`1+2 is $(1+2).`	`1+2 is 3.`

String interpolation plays a role where expressions are used to generate dynamic strings. When expressions are used as standalone features (i.e. as parts of a custom layout query - see About Custom Layout Queries), string interpolation is not supported inside string constants, but strings can be built dynamically using the "+" operator.

Basic data types

Expressions use different data types to represent strings or numeric values. The following data types are supported currently:

Type	Examples
Numeric	`1.2` `-0.5e-6`
String	`"abc"` `'x'`
Boolean	`true` `false`
Array	`[1,5,4]`
Undefined (no value)	`nil`

Apart from that, all RBA objects are supported with their methods (see Class Index). For example, that is a valid expression which gives a value of 100:

```
Box.new(-10, 0, 90, 60).width
```

In a boolean context (i.e. the conditional evaluation "condition ? expr1 : expr2"), "nil" and the boolean "false" will render false, while all other values render true. This follows the Ruby convention and in effect, unlike some other languages, a numeric value if 0 is not treated as "false" but as "true"!

Constants

The following constants are defined currently:

Constant	Description
`M_PI`	The mathematical constant 'pi'
`M_E`	The mathematical constant 'e'
`false`	'false' boolean value
`true`	'true' boolean value
`nil`	The 'undefined' value

Operators and precedence

KLayout's expressions support the following operators with the given precedence:

Prec.	Operator	Data types	Result type	Description
1	`(...)`	Any		Grouping of sub-expressions
2	`[...,...]`	Any	Array	Array formation
3	`!...`	Boolean	Boolean	Logical 'not'
3	`~...`	Numeric	Numeric	Bitwise 'not' (evaluated as 32 bit integers)
3	`-...`	Numeric	Numeric	Negation
4	`...^...`	Numeric	Numeric	Bitwise 'xor' (evaluated as 32 bit integers)
4	`...&...`	Numeric	Numeric	Bitwise 'and' (evaluated as 32 bit integers)
4	`...|...`	Numeric	Numeric	Bitwise 'or' (evaluated as 32 bit integers)
5	`...%...`	Numeric	Numeric	Modulo
5	`.../...`	Numeric	Numeric	Division
5	`...*...`	Numeric	Numeric	Product
		Numeric*String	String	String multiplication (n times the same string)
6	`...-...`	Numeric	Numeric	Subtraction
6	`...+...`	Numeric	Numeric	Addition
		String	string	Concatenation
7	`...<<...`	Numeric	Numeric	Bit shift to left
7	`...>>...`	Numeric	Numeric	Bit shift to right
8	`...==...`	Any	Boolean	Equality

8	...!=...	Any	Boolean	Inequality
8	...<=...	Any	Boolean	Less or equal
8	...<...	Any	Boolean	Less
8	...>=...	Any	Boolean	Greater or equal
8	...>...	Any	Boolean	Greater
8	...~...	String	Boolean	Match with a glob expression
8	...!~...	String	Boolean	Non-match with a glob expression
9	...&&...	Boolean	Boolean	Logical 'and'
9	...||...	Boolean	Boolean	Logical 'or'
10	...?...:...	Boolean?Any:Any	Any	Conditional evaluation

The match operators work on strings. They use the glob pattern notation (as used in the shell for example) and support substring matching with the usual bracket notation. Substrings can be referred to by "$n" later, where n is the nth bracket. For example:

Expression	Result
`"foo" ~ "f*"`	true
`"foo" ~ "bar"`	false
`"foo" !~ "bar"`	true
`"foo" ~ "f(*)"; $1`	"oo"

Method calls

Expressions support all the objects provided by KLayout for the Ruby API. Objects are values inside expressions like integers or strings are. Sometimes, objects can be manipulated with the operators as well (like "box1 + box2"). The most important way to work with objects however are methods.

The dot calls a method on an object. Before the dot an expression must be given which results in an object, or a class name must be given. In the latter case, static methods will be called. After the dot, a valid method name is expected.

Important note: the method names used inside expressions usually is equivalent to the names mentioned in the class documentation. Setter methods like "box_with=" can be used as targets in assignments, i.e.

```
shape.box_width = 20
```

Boolean predicates (like "is_box?") are used **without** the question mark because that is reserved for the decision operator (".. ? .. : .."):

```
shape.is_box
```

Concatenation of expressions

The semicolon separates two expressions. The value of that compound expression is the value of the last one.

Variables

Depending on the context, some variables may be already defined. For example, when used for generating ruler dimension labels, "D" is a predefined variable that is the length of the ruler. See the specific documentation on these variables.

Inside expressions, variables can be defined to store intermediate results for example. To define a variable use the "var" keyword followed by the variable name and an optional initialisation. Assignment of values can be done with the "=" operator. For example, the following expression gives the result 4:

```
var x = 3; x = x + 1; x
```

Special variables

Special variables start with a dollar character. Currently the only special variables available are "$1..9" which is the 1 to 9nth substring match of the last match expression.

Special constants

In the context of a layout, various additional constant expressions are supported:

Distance and area units

A value with a unit is automatically converted to database units. For example, "`0.15 um`" will give 150 if the database unit of the layout is 1 nm. See below for a list of units available. Supported units are:

Unit	Description
`nm`	Nanometers
`um`, `mic`, `micron`	Micrometers
`mm`	Millimeters
`m`	Meters
`nm2`	Square nanometers
`um2`, `mic2`, `micron2`	Square micrometers
`mm2`	Square millimeters
`m2`	Square meters (for very big chips)

Layer index constants

A layer given in the common notation and enclosed in angle brackets is converted to a layer index. For example: "<16/0>" will be converted to the layer index of the layer with layer number 16 and datatype 0.

Cell index constants

A cell name enclosed in double angle brackets will be converted to the index of that cell, for example "<<TOP>>".

Functions

KLayout's expressions support the following functions:

Function	Data types	Result type	Description
absolute_file_path(x)	String	String	Convert a relative file path to an absolute one
absolute_path(x)	String	String	Returns the absolute path component of a file specification
abs(x)	Numeric	Numeric	Returns the absolute value of a number
acos(x)	Numeric	Numeric	Inverse cosine function
asin(x)	Numeric	Numeric	Inverse sine function
atan2(x,y)	Numeric	Numeric	Inverse tangent of x/y
atan(x)	Numeric	Numeric	Inverse tangent function
basename(x)	String	String	Returns the basename component of a file specification
ceil(x)	Numeric	Numeric	Round up
combine(x,y)	String	String	Combines the path components x and y using the system specific separator
cosh(x)	Numeric	Numeric	Hyperbolic cosine function
cos(x)	Numeric	Numeric	Cosine function
env(x)	String	String	Access an environment variable
error(x)	String		Raise an error
exp(x)	Numeric	Numeric	Exponential function
extension(x)	String	String	Returns the extension component of a file specification
file_exists(x)	String	Boolean	Returns true if the given file exists
find(s,t)	String	Numeric	Finds the first occurrence of a t in s and returns the position (where 0 is the first character)
floor(x)	Numeric	Numeric	Round down

`gsub(s,x,y)`	String	String	Substitute all occurrences of a x in s by y
`is_array(x)`	Any	Boolean	True if the argument is an array
`is_dir(x)`	String	Boolean	Returns true if the given path is a directory
`is_nil(x)`	Any	Boolean	True if the argument is undefined
`is_numeric(x)`	Any	Boolean	True if the argument is numeric
`is_string(x)`	Any	Boolean	True if the argument is a string
`item(a,i)`	Array	Any	Access a certain item of an array
`join(a,s)`	Array, String	String	Join all array members in a into a string using the separator s
`len(x)`	String	Numeric	Return the length of a string
`log10(x)`	Numeric	Numeric	Base 10 logarithm function
`log(x)`	Numeric	Numeric	Natural logarithm function
`max(a,b ...)`	Numeric	Numeric	Maximum of the given arguments
`min(a,b ...)`	Numeric	Numeric	Minimum of the given arguments
`path(x)`	String	String	Returns the path component of a file specification
`pow(x,y)`	Numeric	Numeric	Power function (x to the power of y)
`rfind(s,t)`	String	Numeric	Finds the last occurrence of a t in s and returns the position (where 0 is the first character)
`round(x)`	Numeric	Numeric	Round up or down
`sinh(x)`	Numeric	Numeric	Hyperbolic sine function
`sin(x)`	Numeric	Numeric	Sine function
`split(t,s)`	String	Array	Splits t into elements using the separator s
`sprintf(f,a ...)`	String, Any	String	Implement of 'C' sprintf. Provides not all features but the most commonly used ones (precision, field width, alignment, zero padding, 'e', 'g', 'f', 'd', 'x', 'u' and 's' formats)
`sqrt(x)`	Numeric	Numeric	Square root
`substr(t,f[,l])`	String	String	Return a substring of t (starting from position f with length l). 'l' is optional. If omitted, the tail of the string is returned.
`sub(s,x,y)`	String	String	Substitute the first occurrence of a x in s by y
`tanh(x)`	Numeric	Numeric	Hyperbolic tangent function
`tan(x)`	Numeric	Numeric	Tangent function
`to_f(x)`	Any	Numeric	Convert argument to numeric if possible
`to_i(x)`	Any	Numeric (integer)	Convert argument to numeric (32 bit integer)
`to_s(x)`	Any	String	Convert argument to string