K Layout

High Performance Layout Viewer And Editor

Version 0.21.16

Development and Documentation by Matthias Köfferlein [1]
Typesetting by Peter Ragosch

March 26, 2012

[1]Contact

Abstract

KLayout, the high performance layout viewer and editor, is continuously developed and improved by Matthias Köfferlein since the first official release, Version 0.09, dated April 2006 and published under the GNU public license GPL. The software is available for Linux®[1], Windows™[2] and Mac OS[3] operating systems. **KLayout**'s Home Page describes the application features, the build and use, the Ruby scripting interface and many more in detail.

This article is compiled with the intention to collect all available information about **KLayout** from the home page into one compact, and therefore, easy search able PDF document.

Document Revision History

Version	Date	Description
0.21.16	2012, March	Chapter 4: Release Notes and Tar-Kits, section 4.1: Version 0.21.16 and section 4.2: Version 0.21.15 added. Chapter 8: Quick Start Manual – Viewer Mode, section 8.3.15: Saving a layout or parts of it, dialog *Layout Writer Option* on *GDS2 Writer Options* dialog page: item ☐ Write current time to time stamps and description added. Some minor typesetting improvements.
0.21.14	2012, February	Initial Version

[1]Linux® is the registered trademark of Linus Torvalds in the U.S. and other countries.
[2]Windows is a registered trademark of Microsoft Corporation in the United States and other countries.
[3]Mac OS is a trademark of Apple Inc., registered in the U.S. and other countries.

Contents

List of Figures

List of Dialog Entries and Code Snippets

Part I

About The Project

Chapter 1

KLayout Highlights

Content

1.1 KLayout Features

1.1.1 General

- Fast and accurate: fast loading and drawing
- Support of GDS and OASIS file formats with automatic decompression of zlib compatible formats
- Full support of properties
- Full 64 bit support on Linux
- Extensible and configurable to a large degree by custom ruby scripts
- Support of DXF file format (still under construction)

1.1.2 Viewer

- Overlay capabilities: multiple layouts can be loaded into one window
- Very flexible layer configuration: many display options including choice of fill pattern and different frame and fill colors, animation, transparency, dimming/highlighting ...
- Layer grouping: the display properties of a group of layers can be changed at once
- Advanced layer display attributes: layers can be named, they can carry additional transformations, select certain hierarchy levels or select shapes by their properties
- Copy and paste of layers attributes to other panels
- Drawing order: select the layer that is show on top
- Descend into hierarchy: show a cell embedded into it's context
- Flexible rulers: unlimited count, flexible display styles. Multiple templates can be configured, rules can be edited (move, delete, copy & paste)
- Shape and instance browsers
- Bookmarks, various zoom modes, mouse wheel support, screen-shot function ...
- Undo/redo on layer properties, for rulers ...

- Save: save layout or parts (cells, layers) of it to a different format, with scaling or different database unit.
- Image overlay capabilities: image files (i.e. `jpg`, `png`, `gif`) can be loaded and placed at an arbitrary position in the layout.
- Marker browser: certain error report files can be loaded and a browser tool is provided.

1.1.3 Editor

- Smart drawing functions with many options: angle constraints, grid ...
- True, in-place editing in sub-cells
- Unlimited undo/redo
- Smart partial editing function to stretch shapes, move edges or vertices
- Copy and paste of shapes and whole cells, even to other layouts
- Many advanced editing functions: hierarchical operations, booleans, clip, corner rounding, sizing, alignment, layer operations ...

1.2 KLayout is a GDS and OASIS file viewer

Although a comparatively simple piece of software, a layout viewer is not only just a tool for the chip design engineer. Today design's complexity require not only a simple *viewer*. Rather, a viewer is the microscope through which the engineer looks at the design.

There are numerous viewers available, but sadly there are not many which satisfy a few basic requirements. Most of them are commercial and expensive. If there is need for a simple, yet powerful viewer - here it is.

The main objective was to focus on the basic functionality but adding some useful features that many, even commercial viewers don't have.

First rarely any tool allows to place two or even more layout files over each other. It often happens that you receive some layers in one file, the other layers in another. Some tools allow to load multiple layouts and switch between the windows. Well, this may help - but still the possibility of overlaying two layouts offers much more comfort.

Sadly, almost no viewer is really precise. There is not much more annoying than a layout that changes when you zoom into it. Or placeholder shapes appearing at some zoom level and disappearing at the next, cell labels that cannot be caught because they jump around when you try to zoom them into view, and many other surprising ways or creative interpretation and optimization. This viewer shows the design as it is.

Only some viewers allow to make layers *transparent*. Only this way, a stack of layers can be visualized effectively. In addition, this viewer can animate layers to make them blink or scroll the fill pattern. Animation is a good tool to highlight certain layers.

This viewer allows to display a layer *marked* by drawing a small cross on all shapes. There is not better way to visualize the distribution of a set of sparse error markers on a dense layout!

All comes wrapped in a nice, Qt based state of the art GUI. Usage of the viewer is simple and is similar to that of other tools.

1.3 KLayout is more

Starting with version 0.15, **KLayout** is also an **editor** that allows to change GDS and OASIS files and create them from scratch. See section 4.33, Release Notes of Version 0.15 and chapter 9: Quick Start Manual – Editor Mode, for a more detailed description.

KLayout also offers a Ruby-based scripting environment called *RBA* which allows to automate various tasks, mainly in the visualization area but also for layout generation. See chapter 11, an introduction into the ruby based automation API, for details about this feature.

1.4 KLayout is free

The viewer is published under GNU public license GPL version 2 or any later version in compliance with the requirements for using the Qt open source license. It may be copied and distributed freely.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of Merchantability or fitness for a particular purpose. Please use it AT YOUR OWN RISK.

1.5 Current status

The current version is 0.21. Although this low number reflects the early stage of development, the version is reasonably stable.

KLayout currently builds on recent Linux® installations, under Solaris and can be built on Windows™ using cygwin and mingw. For the precise requirements see below and chapter 2: Download and Build, or on the download and build page. For Windows™, a package is provided that contains the executable and DLL's required.

The issue list for version 0.21 can be found in section 5.6, Known Bugs and Issues List.

Currently there is no comprehensive documentation but I hope the user interface is intuitive enough to compensate this lack somewhat. However, there is a Quick Start Manual – Viewer Mode in chapter 8 available. Also for editor mode a Quick Start Manual – Editor Mode is provided in chapter 9.

The build is performed by a simple wrapper shell script rather than some sophisticated configuration setup. Some options allow to configure the script somewhat. This is definitely the weak spot of the current release. I hope I will be able to obtain a more elaborate setup in the next release.

The build requires the Qt4 GUI toolkit in the multi-threaded version and a recent gcc version to build. If required, the Qt4 toolkit can be obtained from Qt HOME.

Since the viewer is based on open platforms, I would expect that it also compiles on other platforms. The GUI abstraction through Qt even allows to compile it on Windows™with only very minor code specializations.

1.6 The future of the project

KLayout is a living project. The program is being used by people in their daily work already. As my time allows I will gradually enhance and extend the code. I personally like to add editing capabilities. However, this is a major step, but the basics are already set up in the current code.

Please feel free to issue feature requests to this mail address.

I am always eager for learning about use cases and potential new applications for this tool.

Chapter 2

Download and Build

Content

2.1 Download Current Version

Download packages for the current versions from these links. You find download links for other versions here.

Platform	Type	Version	Download
unix	source package	0.21.16	klayout-0.21.16.tar.gz
Windows 32 bit	binary package	0.21.16	klayout-0.21.16-win32.zip
	installer		klayout-0.21.16-win32-install.exe
Windows 64 bit	binary package	0.21.16	klayout-0.21.16-win64.zip
	installer		klayout-0.21.16-win64-install.exe
Windows 32 bit	binary package	0.21.16	klayout-0.21.16-mingw32.zip
legacy MinGW based build			
MacOS 10.6	binary package	0.21.16	klayout-0.21.16.intel-snowleopard.dmg.zip

2.2 Development Snapshot

A snapshot of the current development code base (last update 2012-03-05) can be obtained here: klayout-r1134.tar.gz.

2.3 Packaged Release for Windows

For the Windows platform, a zip archive is provided that contains all the required executable and DLL's. A description how to actually build **KLayout** on Windows using MinGW and Visual Studio 2010 can be found below.

Starting with version 0.15, an installer executable is provided as well. To install **KLayout** using the installer, download the executable and execute it. It will install the binaries at the target location, which can be selected in the installer user interface. In addition, it will create three **KLayout** shortcuts in the Programs section of the Start menu. It will also register itself as handler for file types `.gds` and `.oas`.

If the installer is executed from a normal user account, it will install itself for that user only. No particular rights are required in this case. If started with administrator rights, it will install itself for all users.

2.4 Building on MacOS

For building the executable on MacOS 10.5.7, the "mac-leopard-gcc-release" platform is provided. The build is based on the Xcode installation. This build script call was used successfully with Qt 4.5.2 from the Xcode package:

Console Input 2.1: Build Script on MacOS 10.5.7

```
./build.sh -platform mac-leopard-gcc-release \
  -qtbin /Developer/Tools/Qt \
  -qtlib /usr/lib \
  -rblib /usr/lib/libruby.dylib \
  -rbinc /usr/lib/ruby/1.8/universal-darwin9.0
```

On MacOS 10.6 this build script call was used successfully (Qt library is `qt-mac-cocoa-opensource-4.6.2.dmg`):

Console Input 2.2: Build Script on MacOS 10.5.6

```
./build.sh -platform mac-leopard-gcc-release \
  -qtbin /Developer/Tools/Qt \
  -qtlib /usr/lib \
  -rblib /usr/lib/libruby.dylib \
  -rbinc /usr/lib/ruby/1.8/universal-darwin10.0
```

Apparently, both 32 and 64 bit versions of Qt must be installed.

2.5 Building KLayout on Unix

System Requirements:

- Linux system (tested on Ubuntu 8.04LTS, 10.04LTS, RHE 4)
- on Linux: gcc Version 3.2 or later (tested with 3.4.5, 4.3.2, 4.4.3). Solaris is no longer supported.
- Qt Version 4.2.x or later (tested with 4.2.3, 4.4.3, 4.5.2, 4.6.2).
- gcc C++ compiler (package "g++" on Debian)
- zlib library and headers (package "zlib1g-dev" on Debian)

To build, the simplest way is to use the build.sh wrapper script provided. Unpack the tarkit, change to the directory created and type

Console Input 2.3: Simple Build on Unix

```
./build.sh
```

If the script complains about missing tools are libraries, the Qt installation needs to be specified. There are basically two ways: either a Qt package was configured or installed using the environment provided by TrollTech, or the system provides a Qt4 installation itself.

In the first case, the actual Qt installation path can be specified with the -qt option: i.e.

Console Input 2.4: Simple Build on Unix with Qt Path

```
./build.sh -qt ~/qt
```

will locate the Qt include files in ⊠/`qt/include`, the Qt library in ⊠/`qt/lib`. The installation path is the prefix that was specified on Qt's configure command line.

On LSB (Linux Standard Base) systems, the Qt4 library can be optionally installed. It is common to have different installation directories for include and library files. On Ubuntu 6.10 with Qt4 for example, the libraries are located in `/usr/lib`, the development tools like **uic** are installed in `/usr/bin` and the include files reside in `/usr/include/qt4`. In this case, use:

Console Input 2.5: Simple Build on Linux Standard Base Systems

```
./build.sh -qtbin /usr/bin -qtlib /usr/lib -qtinc /usr/include/qt4
```

Since the above settings are the default, this is equivalent to specifying nothing as shown above.

The build script does not determine the platform to build upon. By default, 32 bit Linux and **gcc** is configured as the build platform. To build for an other platform, use

Console Input 2.6: Simple Build on Unix for other Platform

```
./build.sh -platform <platform>
```

The platforms and build variants available are listed with

Console Input 2.7: Simple Build on Unix – Known Platform List

```
./build.sh -help
```

The build script will create the final executable in

Console Input 2.8: Simple Build on Unix – Final Executable Path

```
./bin.<platform>/klayout
```

To integrate other build variants, a new file can be created in the `config` sub-directory. This is a partial Makefile that defined the variables specific for a certain build.

To enable Ruby script automation capabilities (RBA), the Ruby library and path to the Ruby headers must be specified for the build script:

Console Input 2.9: Simple Build on Unix with Ruby Support

```
build.sh -rblib <ruby library path> -rbinc <ruby headers location>
```

For example:

Console Input 2.10: Simple Build on Unix with Ruby Support — Example

```
build.sh -rblib /usr/lib/libruby1.8.so -rbinc /usr/lib/ruby/1.8/i486-linux
```

For more details about RBA, see chapter 11: RBA Introduction.

2.6 Building KLayout for Windows 32 bit with MinGW

Starting with version 0.21, a build setup is provided for MinGW with the gcc. A good starting point for the MinGW build is the Qt SDK which comes with a Qt retail built for MinGW and the gcc compiler suite.

To enable Ruby support, the Ruby interpreter, preferably version 1.9 is required. A strange fact with building Ruby 1.9 is that one needs a Ruby interpreter to build it. One possible solution is first to build a 1.8 version, put the executable into the path and then run the Ruby build from the MinGW console.

To build **KLayout** on MinGW, simply open the MinGW shell (MSYS), unpack the **KLayout** source package and cd to the destination folder. Then use build.sh as on Linux. For example, if the Qt SDK was installed in `c:\Qt\2010.04`, the build script call is

Console Input 2.11: Build Script for Windows 32 bit with MinGW

```
build.sh -qt /c/Qt/2010.04/qt
```

For ruby support use the -rblib and -rbinc options accordingly to specify the ruby installation path.

2.7 Building KLayout for Windows 32 bit and 64 bit with Visual Studio

Starting with version 0.21, a Visual Studio solution is included in the source branch of **KLayout**'s source tar-kit (`klayout.sln`). The solution is provided for Visual Studio 2010.

To build **KLayout** with Visual Studio, the following requirements must be fulfilled:

- Qt for Visual Studio 2010 (VC++ version 10). Currently this version must be built manually. Qt version 4.7.1 is compatible with Visual Studio 2010 and building it is pretty straightforward.
- For a complete build including Ruby support, the ruby interpreter is required as well. Only version 1.9 is supporting the 64 bit platform. Building is straightforward, except that again a ruby interpreter must be installed before version 1.9 can be built. If no interpreter is at hand, a 1.8 version must be built before. The project files currently assume Ruby version 1.9.1.

  A pre-built package for VS2010 can be downloaded here: ruby1.9.1-p430.zip. It contains both the 32 bit and 64 bit builds in the `1.9.1-p430/x86` and `1.9.1-p420/x64` directories.
- For full performance, it is recommended to replace the standard STL implementation of VC++ with the STLPort implementation which has a 2x performance impact in some cases. Building STLPort on VC++ is straightforward and has been tested with version 5.2.1.

  A pre-built package for VS2010 can be downloaded here: STLport-5.2.1.zip. It contains both the 32 bit and 64 bit builds.

Before building **KLayout**, it is required to set the following environment variables:

- $QTDIR to the installation path of Qt for 32 bit build (`$QTDIR/bin` being the location of the executable, `$QTDIR/lib` being the location of the libraries and `$QTDIR/include` being the location of the header files).
- $QTDIR64 to the installation path of Qt for 64 bit build (`$QTDIR64/bin` being the location of the executable, `$QTDIR64/lib` being the location of the libraries and `$QTDIR64/include` being the location of the header files).

- $STLPORT to the installation path of the STLPort library (if required). `$STLPORT/stlport` must be the location of the headers. This variable is the same for 32 and 64 bit builds.
- $RUBY to the installation path of the Ruby library (if required) for the 32 bit build. The location of the ruby.h header must be `$RUBY/include/ruby-1.9.1`.
- $RUBY64 to the installation path of the Ruby library (if required) for the 64 bit build. The location of the ruby.h header must be `$RUBY64/include/ruby-1.9.1`.

After this preparation, **KLayout** can be build from Visual Studio using the Win32 platform for 32 bit and x64 platform for 64 bit. The configurations provided are:

- **Debug** for the normal debug build without Ruby and STLPort.
- **Debug (STLPort)** for the debug build with Ruby support and using STLPort.
- **Release** for the normal release build without Ruby and STLPort.
- **Release (STLPort)** for the release build with Ruby support and using STLPort.

2.8 All Downloads

All currently available downloads can be found here: http://www.klayout.de/build.html.

Chapter 3

Current Development

This chapter lists features that are developed currently and will go into the next release (Version 0.22).

Content

3.1 Development Snap Shot Tarkit

A snapshot of the current development code base (last update 2011-06-10) can be obtained here:

3.1.1 Tar-Kits

Sources for all systems klayout-r802.tar.gz

3.2 Multithreading for XOR tool

The XOR tool now can make use of multi-CPU architectures by using multiple threads for tiles and layers. The number of threads can be specified on the XOR tool dialog.

3.3 Diff tool performance enhancements.

The diff tool now uses a different scheme to identify identical cells. This algorithm is based on a signature and is much faster the the previous algorithm which was based on instance identity.

Chapter 4

Release Notes and Tar-Kits

This chapter lists available release notes and software packages.

Hint: Menu related items are updated to reflect the menu structure of Version 0.21.

Content

4.1 RN Version 0.21.16

Release Date: 2012-03-05

Tar-Kits

Sources for all systems	klayout-0.21.16.tar.gz
WIN32 binaries and DLL's	klayout-0.21.16-win32.zip
WIN32 installer	klayout-0.21.16-win32-install.exe
WIN64 binaries and DLL's	klayout-0.21.16-win64.zip
WIN64 installer	klayout-0.21.16-win64-install.exe

Features

- GDS format readers and writers now support time stamps: by default, the current time is written to the files to simplify comparison of binary files for example. This option can be turned off in the menu File » Save » Layout Writer Options » ☐ Write current time to time stamps. In addition, the time stamp of the BGNLIB record is read and displayed in the File » Layout Properties page.

- The GDS reader now is somewhat less strict and also accepts certain broken versions (i.e. missing ENDEL records).
- Several bug fixes related to scripting applications: Proc objects are held by the application now, Application does not abort in non-GUI mode in operations that take some time and try to display a progress bar.
- DXF bugfix: layer names now do no longer contain blanks which made files unreadable by other tools like AutoCad.
- Bugfix: foreground objects (i.e. rulers) are now correctly rendered in printout.

4.2 RN Version 0.21.15

Release Date: 2012-03-05

Tar-Kits

Sources for all systems	klayout-0.21.15.tar.gz
WIN32 binaries and DLL's	klayout-0.21.15-win32.zip
WIN32 installer	klayout-0.21.15-win32-install.exe
WIN64 binaries and DLL's	klayout-0.21.15-win64.zip
WIN64 installer	klayout-0.21.15-win64-install.exe

Features

- Bugfix: the correct initial cell now is selected. Formerly, the largest cell was selected even it is was not a top cell.
- Layers context > Show only selected in the layer panel does not clear the selection any longer.
- GDS reader now is less strict with respect to record order of STRANS, MAG and ANGLE.
- Excellon drill file reader is now conforming to the specification in many respects.
- Instances are not selected if the cell does not contain shapes in visible layers.
- Marker browser does now work correctly when layer view transformations are present.
- DXF reader enhancements: read LAYER table and assign GDS layers in that order, except for layer which got a layer name through their name (i.e. L1D100).

  Bugfix: don't suppress INSERT's if the layer is not mapped. Write TEXT and MTEXT correctly (multi-line support, small chunks for MTEXT, character alignment).

  New option: keep all cells for DXF reader. Added elliptic interpolation edge type (not really tested yet). HATCH objects with bulges and various edge types are implemented now. MTEXT supported now.

  New option: convert text to polygon for Unicode support.
- Changed default sorting of layers: always sort by layer number first, even if there is a name. If there is no layer number, sort by name.

4.3 RN Version 0.21.14

Release Date: 2011-11-28

Tar-Kits

Sources for all systems	klayout-0.21.14.tar.gz
WIN32 binaries and DLL's	klayout-0.21.14-win32.zip
WIN32 installer	klayout-0.21.14-win32-install.exe
WIN64 binaries and DLL's	klayout-0.21.14-win64.zip
WIN64 installer	klayout-0.21.14-win64-install.exe

Features

- View > Synchronized views and View > Select Top Level Objects configuration shortcuts added.
- Gerber reader bug fixes and enhancements: less strict parsing of aperture definitions, rotation of aperture macro elements is not considered correctly. Enhanced drill file reader.
- Disabled cell copy & paste in viewer mode (was not working correctly).
- Bug fix: it was possible to create an invalid configuration when removing all default stipples (lead to a crash on the next **KLayout** start).
- Bug fix: an error appeared when switching the tabs in the layer panel in certain configuration involving groups.
- Correct initialization of ruby interpreter to support Ruby 1.9.2 and later.
- Bug fix: marker browser was only partially reporting collected markers for certain categories.
- XOR now has an option to make use of multiple cores using a configurable number of threads.
- Bug fix: reset of configuration required a restart.
- Bug fix: OASIS reader now is more robust against overflow for g-deltas.
- GDS reader enhancement: An invalid angle (outside the range of -360 to 360 degree) now is no longer an error and the angle is automatically restricted to the valid range.

4.4 RN Version 0.21.13

Release Date: 2011-09-19

Tar-Kits

Sources for all systems	klayout-0.21.13.tar.gz
WIN32 binaries and DLL's	klayout-0.21.13-win32.zip
WIN32 installer	klayout-0.21.13-win32-install.exe
WIN64 binaries and DLL's	klayout-0.21.13-win64.zip
WIN64 installer	klayout-0.21.13-win64-install.exe
MacOS 10.7	klayout-0.21.13.intel-lion.dmg

Features

- A bug in the clip function was fixed which was related to empty cells (reported in the forum).
- By default, the first level of hierarchy is shown now if a new layout is opened. That feature can be adjusted using the Default levels of hierarchy setting on the File, Setup, General dialog page.
- For multiple top cells, the cell with the larges footprint is selected initially.
- A simple print function available in the File > Print menu.

- Support for command 93 in CIF (AREF).
- Improved handling of single point paths. In particular with round ends. They now render a circle in OASIS. In reverse, OASIS circles now render single-point paths with round ends in GDS.
- Ruby scripts now work more reliably under ruby 1.9 (i.e. Windows binary). Formerly, some operations failed due to improper initialization of the encoding system (i.e. Dir.glob).
- Clean uninstaller under Windows (removes all registry entries).

4.5 RN Version 0.21.12

Release Date: 2011-07-29

Tar-Kits

Sources for all systems	klayout-0.21.12.tar.gz
WIN32 binaries and DLL's	klayout-0.21.12-win32.zip
WIN32 installer	klayout-0.21.12-win32-install.exe
WIN64 binaries and DLL's	klayout-0.21.12-win64.zip
WIN64 installer	klayout-0.21.12-win64-install.exe

Features

- Bug fix: GDS2Text format was not recognized correctly in some cases.
- Texts: strings with line breaks can be edited now (line breaks are shown as \n). Text size is shown more realistic now. Alignment flags are supported in GDS2 and can be edited now.
- Layer and datatype is shown in addition to OASIS layer names in the layer list. The old behavior can be configured by deselect the File > Setup > Layer List > Always show layer and datatype check box.
- For most *File* dialogs, the specific filter is the default now (i.e. `*.lyp` instead of *All files* for the layer properties file dialogs.)
- There is a *all layout files* filter for the File > Open, File > Open In Same Panel and File, Open In New Panel dialogs.

4.6 RN Version 0.21.11

Release Date: 2011-06-26

Tar-Kits

Sources for all systems	klayout-0.21.11.tar.gz
WIN32 binaries and DLL's	klayout-0.21.11-win32.zip
WIN32 installer	klayout-0.21.11-win32-install.exe
WIN64 binaries and DLL's	klayout-0.21.11-win64.zip
WIN64 installer	klayout-0.21.11-win64-install.exe

Features

- Bug fix: command line option -p was not working correctly.
- Bug fix: writing layouts with large coordinates was producing invalid OASIS files in some cases.
- The tar-kit now contains the files necessary for a build with Visual Studio on Windows.

4.7 RN Version 0.21.10

Release Date: 2011-05-07

Tar-Kits

Sources for all systems	klayout-0.21.10.tar.gz
WIN32 binaries and DLL's	klayout-0.21.10-win32.zip
WIN32 installer	klayout-0.21.10-win32-install.exe
WIN64 binaries and DLL's	klayout-0.21.10-win64.zip
WIN64 installer	klayout-0.21.10-win64-install.exe

Features

- Bug fix: content of cell was not shown correctly when the cell was moved.
- Bug fix: PCB import was not working properly (Bottom mounting mode was broken, top cell and dbu were not set correctly when a PCB project file was imported directly).
- Bug fix: RVE reader was not correctly handling check names with a dot.

4.8 RN Version 0.21.9

Release Date: 2011-04-20

Tar-Kits

Sources for all systems	klayout-0.21.9.tar.gz
WIN32 binaries and DLL's	klayout-0.21.9-win32.zip
WIN32 installer	klayout-0.21.9-win32-install.exe
WIN64 binaries and DLL's	klayout-0.21.9-win64.zip
WIN64 installer	klayout-0.21.9-win64-install.exe

Features

- Bug fix: Loading of layer files with tabs was not working properly: the first tab's name was discarded.

4.9 RN Version 0.21.8

Release Date: 2011-04-06

Tar-Kits

Sources for all systems	klayout-0.21.8.tar.gz
WIN32 binaries and DLL's	klayout-0.21.8-win32.zip
WIN32 installer	klayout-0.21.8-win32-install.exe
WIN64 binaries and DLL's	klayout-0.21.8-win64.zip
WIN64 installer	klayout-0.21.8-win64-install.exe

Features

- Bug fix: navigator was broken. It was not functional if **KLayout** was closed with the navigator open.

4.10 RN Version 0.21.7

Release Date: 2011-03-24

Tar-Kits

Sources for all systems	klayout-0.21.7.tar.gz
WIN32 binaries and DLL's	klayout-0.21.7-win32.zip
WIN32 installer	klayout-0.21.7-win32-install.exe
WIN64 binaries and DLL's	klayout-0.21.7-win64.zip
WIN64 installer	klayout-0.21.7-win64-install.exe

Features

- DXF bug fixes (arc interpolation of polylines).
- Performance enhancement of Layer context > Test For Shapes In View feature.

4.11 RN Version 0.21.6

Release Date: 2011-02-20

Tar-Kits

Sources for all systems	klayout-0.21.6.tar.gz
WIN32 binaries and DLL's	klayout-0.21.6-win32.zip
WIN32 installer	klayout-0.21.6-win32-install.exe
WIN64 binaries and DLL's	klayout-0.21.6-win64.zip
WIN64 installer	klayout-0.21.6-win64-install.exe

Features

- DXF enhancements (support for bulges for polylines).
- Bug fix: RBA::Edge.intersect? reports intersections correctly also if edges are collinear.
- Performance bug fix: selection was slow for certain cases of hierarchy.

4.12 RN Version 0.21.5

Release Date: 2011-02-03

Tar-Kits

Sources for all systems	klayout-0.21.5.tar.gz
WIN32 binaries and DLL's	klayout-0.21.5-win32.zip
WIN32 installer	klayout-0.21.5-win32-install.exe
WIN64 binaries and DLL's	klayout-0.21.5-win64.zip
WIN64 installer	klayout-0.21.5-win64-install.exe

Features

- DXF enhancements (some poly lines now have the correct width).
- An option to select how instances are placed (at origin or lower left corner of bounding box).
- Bug fix: when placing an instance at the lower left bounding box corner, it is guaranteed that the origin is on grid.
- Stable operation on UTF-8 file systems and with non-English locales on Linux (i.e. consistent use of dot as decimal point). However, UTF-8 file names are not correctly displayed although the file is opened correctly. This will be fixed in the next major release.

4.13 RN Version 0.21.4

Release Date: 2011-01-19

Tar-Kits

Sources for all systems	klayout-0.21.4.tar.gz
WIN32 binaries and DLL's	klayout-0.21.4-win32.zip
WIN32 installer	klayout-0.21.4-win32-install.exe
WIN64 binaries and DLL's	klayout-0.21.4-win64.zip
WIN64 installer	klayout-0.21.4-win64-install.exe

Features

- Correct installation of the image reader plug-ins for the Windows packages.
- Bug fix: rotate counterclockwise was clockwise and vice versa.
- Bug fix: File » Setup » Display » Optimization » Array » Draw only border instances in detailed view feature was broken.
- Fixed a ruby crash on some systems (related to an initial *require* on a ruby module loaded with option "-rm").
- CIF reader enhancement: "DS" statements are now accepted with a single value also.
- The Windows installation now also includes the standard Ruby modules.

4.14 RN Version 0.21.3

Release Date: 2010-12-27

Tar-Kits

Sources for all systems	klayout-0.21.3.tar.gz
WIN32 binaries and DLL's	klayout-0.21.3-win32.zip
WIN32 installer	klayout-0.21.3-win32-install.exe
WIN64 binaries and DLL's	klayout-0.21.3-win64.zip
WIN64 installer	klayout-0.21.3-win64-install.exe

Features

- DXF reader and writer enhancements. The reader now allows to specify how POLYLINE entities are read. In most cases, the "Automatic" mode will be appropriate. The writer was enhanced by providing an option which determines how to write polygons. The default method is POLYLINE. A comprehensive description of the DXF format, as **KLayout** understands it, together with a description of the modes, is given in section 6.5: DXF format.

4.15 RN Version 0.21.2

Release Date: 2010-12-19

Tar-Kits

Sources for all systems	klayout-0.21.2.tar.gz
WIN32 binaries and DLL's	klayout-0.21.2-win32.zip
WIN32 installer	klayout-0.21.2-win32-install.exe
WIN64 binaries and DLL's	klayout-0.21.2-win64.zip
WIN64 installer	klayout-0.21.2-win64-install.exe

Features

- RBA bug fix: RdbItem.each_value was not working on constant references.
- DXF reader and writer enhancements for improved interoperability with other tools.
- Unit option for DXF input (to specify the units of the drawing).
- Bug fix: layer mapping was not working correctly for DXF and CIF output.

4.16 RN Version 0.21.1

Release Date: 2010-12-06

Tar-Kits

Sources for all systems	klayout-0.21.1.tar.gz
WIN32 binaries and DLL's	klayout-0.21.1-win32.zip
WIN32 installer	klayout-0.21.1-win32-install.exe
WIN64 binaries and DLL's	klayout-0.21.1-win64.zip
WIN64 installer	klayout-0.21.1-win64-install.exe

Features

- Added cell margins for fill utility.
- A couple of bugs fixed (related to Diff tool, marker database reader, Ruby scripting).
- Source is compatible with earlier versions of Qt now (down to 4.2.3).
- Bug fix: persistence of reader options was broken.
- Enhanced DXF and CIF reader and writer functionality with improved compatibility with other systems.

4.17 RN Version 0.21

Release Date: 2010-11-28

Tar-Kits

Sources for all systems	klayout-0.21.tar.gz
WIN32 binaries and DLL's	klayout-0.21-win32.zip
WIN32 installer	klayout-0.21-win32-install.exe
WIN64 binaries and DLL's	klayout-0.21-win64.zip
WIN64 installer	klayout-0.21-win64-install.exe

Features

- Support for DXF format (reading and writing).
  ASCII and binary format are supported. There is an open issue how to represent layouts with multiple top cells. Currently, the ENTITY section is always empty and all cells are put into BLOCKS sections. DXF units will be micron and the database unit must be selected manually when layouts are read.
- Support for CIF format (reading and writing).
  There is an open issue how to deal with paths. By default, CIF states that paths are supposed to have round ends. Obviously that interpretation is not commonly used. Options are provided which control how path objects are read and written.
- Tabs for the layer panel. This feature is explained in section 8.3.2: Using multiple layer properties setups with tabs.
- Flat cell list and cell list sorting modes. The feature is explained in section 8.1.6: Configuring the cell list.
- Dockable tool boxes.
  Layer list, cell list, layer toolbox and navigator are now dock-able and can be dragged to another location, torn off the main window or closed with Qt's standard dock-able window controls. The position and the state of the dock-able windows is saved in the settings and session files.

- A ruler embedded in the background image.
  Now, a small ruler is embedded into the background which shows the dimension scale similar to a map. It can be disabled with the File > Setup > Display > Background > Show grid net > Show Ruler check-box.
- Image quality enhancement by oversampling. This feature is explained in section 8.3.13: More configuration options.
- The *Diff* tool. The Diff tool produces a marker database containing a description of the differences. A detailed description can be found in section 10.2: The Diff tool.
- Snapping to objects is provided as an option for edit mode. In this mode, the mouse snaps to vertices and edges of visible objects. This mode can be enabled with the Edit > Editor Options > Snapping > Objects > Snap to other objects check-box. This menu is available via keysF3 shortcut.
- The reference point for the placement of instances now is the lower left point of the placed cell's bounding box, not the origin.
- Dialog geometry persistent now.

  Marker, shape and instance browsers now save their geometries and splitter pane configurations when the application exits.
- Instance placement now uses bbox origin, not cell origin. This simplifies placement of cells with their origin not aligned with the content.
- The *Fill* (tiling) tool. The tool is found in Edit, Utilities, Fill Tool. A detailed description can be found in section 10.3: The fill (tiling) utility.
- In some places, particular in the ruler display string, expressions can be used. For rulers, the previous display string placeholder scheme is replaced by the more powerful expression expansion scheme (see section 8.3.7: Ruler properties for details).
- Build support for VC++ and Visual Studio 2010. The Windows 64 bit build now is based on that environment.

  Note: Visual Studio 2010 no longer supports Windows 2000.

  To use **KLayout** on Windows 2000, a legacy build based on MinGW is provided.

4.18 RN Version 0.20.2

Release Date: 2010-05-25

Tar-Kits

Sources for all systems	klayout-0.20.2.tar.gz
WIN32 binaries and DLL's	klayout-0.20.2-win32.zip
WIN32 installer	klayout-0.20.2-win32-install.exe
WIN64 binaries and DLL's	klayout-0.20.2-win64.zip **experimental**
MacOS 10.7	klayout-0.20.2.intel-snowleopard.dmg.zip

Features

- Bug fix: marker browser crashed when opened from menu and the maximum number of markers was set differently from default.

4.19 RN Version 0.20.1

Release Date: 2010-05-23

Tar-Kits

Sources for all systems	klayout-0.20.1.tar.gz
WIN32 binaries and DLL's	klayout-0.20.1-win32.zip
WIN32 installer	klayout-0.20.1-win32-install.exe
WIN64 binaries and DLL's	klayout-0.20.1-win64.zip **experimental**

Features

- Navigator now allows to drag a zoom box in the usual way.
- Paths with odd width (in database units) are shown correctly.
- Some bug fixes concerning the net tracer and the Gerber import feature. For a detailed list of fixed bugs see section 5.8: Version 0.20, Known Bugs).
- Support for Ruby 1.9 **experimental**.
- Support for 64 bit Windows **experimental**.

4.20 RN Version 0.20

Release Date: 2010-05-01

Tar-Kits

Sources for all systems	klayout-0.20.tar.gz
WIN32 binaries and DLL's	klayout-0.20-win32.zip
WIN32 installer	klayout-0.20-win32-install.exe

Features

- Import option for Gerber PCB data File > Import > Gerber PCB. Details about this function can be found in section 10.4: Importing Gerber PCB files.
- A function to import another stream file into the current file Import > Other File Into Current. This avoids having to use copy & paste and provides a couple of nice options. Details about this function can be found in section 10.5: Importing other layout files.
- A simple XOR tool providing a flat XOR between two layouts. A tolerance can be set to suppress small deviations. A tiling option is provided to reduce memory requirements for large layouts. The flat approach probably limits the application to "almost flat", small to medium sized layouts. Details about this function can be found in section 10.1: The XOR tool.
- A net tracing tool to trace single nets of conductive layers connected through via shapes. The function is intended for extracting single small nets and it's not performance optimized for the case of huge power nets. Details about this function can be found in section 10.6: The net tracing feature.
- A navigator window that shows current view's rectangle and allows to control the rectangle by dragging or resizing it.

- Multi threading support for drawing increases the drawing performance on multi-core CPU's by rendering different layers on different CPU's. The number of threads to use can be set in the File, Setup, Display, Optimization page. By default, a single thread is used.
- "Make array" method to multiply the selection into an array arrangement, menu Edit > Selection > Make Array.
- A function to add a layout already loaded to a view, menu File > Pull In Other Layout.
- A function to scan a layer's geometry and create a marker database from that. The menu function is found in the Tools > Verification menu. It will scan all selected layers either flat or hierarchical and create a marker database containing the shapes as markers.
- Layer mapping on input: the reader options now allow to specify a mapping together with a layer specification. In addition to specifying the layers to read, a target can be given which specifies which layer the shapes will be stored under, see the dialog page File, Reader Options, Layout Reader Options.
- A couple of new display options:
  - Abstract mode (shows only the outer interfacing shapes of child cells).
  - Child hierarchy level layout can be configured to be shown in different brightness, hollow fill or static neutral color to differentiate top level from bottom level layout.
  - Enhancements for the hierarchy level specifications in the layer source notation.
- A log viewer is provided to catch warning messages. The menu entry to open the log viewer is found in the File > Log Viewer menu.
- The File, Setup dialog has been reorganized to make room for more property pages.
- Instances now show the cell's content when being moved instead just a rectangle. This feature can be disabled.
- Various usability enhancements (i.e. mouse cursor now shows activity, status bar shows short shape statistics etc.).
- An option to not write empty cells.
- Support for missing cells in the input layout: those cells get marked and are not produced in the output unless they have received content. That allows to load and save layouts with missing cells without producing empty cells for those missing cells.
- Some new functions in the layer list context menu: Show All, Hide All, Show Only Selected.
- Various bug fixes (i.e. for clip function).
- A couple of new RBA methods.

4.21 RN Version 0.19.3

Release Date: 2009-12-17

Tar-Kits

Sources for all systems	klayout-0.19.3.tar.gz
WIN32 binaries and DLL's	klayout-0.19.3-win32.zip
WIN32 installer	klayout-0.19.3-win32-install.exe

Features

- Some bug fixes. For a detailed list of fixed bugs see section 5.10: Version 0.19.2, Known Bugs.
- To specify the installation path (where the `.rbm` files are looked up), a new environment variable KLAYOUT_PATH is provided. This variable currently supports a single entry only. In that sense it's not a "path" right now.

4.22 RN Version 0.19.2

Release Date: 2009-12-05

Tar-Kits

Sources for all systems	klayout-0.19.2.tar.gz
WIN32 binaries and DLL's	klayout-0.19.2-win32.zip
WIN32 installer	klayout-0.19.2-win32-install.exe

Features

- Some bug fixes. For a detailed list of fixed bugs see section 5.11: Version 0.19.1, Known Bugs.

4.23 RN Version 0.19.1

Release Date: 2009-11-30

Tar-Kits

Sources for all systems	klayout-0.19.1.tar.gz
WIN32 binaries and DLL's	klayout-0.19.1-win32.zip
WIN32 installer	klayout-0.19.1-win32-install.exe

Features

- Some bug fixes. For a list of fixed bugs see section 5.12: Version 0.19, Known Bugs.

4.24 RN Version 0.19

Release Date: 2009-11-21

Tar-Kits

Sources for all systems	klayout-0.19.tar.gz
WIN32 binaries and DLL's	klayout-0.19-win32.zip
WIN32 installer	klayout-0.19-win32-install.exe
Universal binary for Mac OS 10.5.7 including Qt	klayout-0.19-mac-leopard.tgz

Features

- Images: Now it is possible to load image files (JPG, GIF, TIFF etc.) and display them below the drawn layout. The display of the images can be adjusted in many ways, i.e. placement, scaling, rotation, mirroring, color/contrast/brightness adjustments and false color mapping for gray level images. Images are fully supported by RBA. The feature is described in detail in section 8.3.8: Adding images.
- Marker browser: The marker browser is based on the report database (RDB). This is a new concept that has been introduced as a container for report items, in particular marker objects but also for a number of additional annotations including screen shots for documentation. An import of Calibre DRC databases is provided. The RDB is fully supported by RBA. The feature is described in detail in section 8.3.11: The marker browser.
- Some enhancements for the layer views: Now invisible layers are shown differently in a "collapsed" way. Thus, the layer's colors and styles are still recognizable. The new features are described in detail in section 8.2.3: Telling used from unused layers.
- In the layer list, now unused layers can be hidden and layers without shapes in view can be marked unused or hidden. The normal mode of marking layers unused when the cell does not contain any shapes at all is still available. Both modes are available as check-able items in the layer list's context menu.
- A View menu has been added with fast access to certain display options, i.e. turn grid on and off View > Show Grid. A number of default grids can be defined File > Setup > Application > Default Grids for quick selection in the View menu. On request, the tool bar can now be hidden as well as the layer and hierarchy lists.
- A key binding editor is provided File > Setup > Application > Kex Bindings. This way, all menu functions can be assigned arbitrary key shortcuts in a more comfortable ways.
- The mid mouse button can be used to pan (drag) the view window now.
- Various RBA enhancements, i.e. a method to compute the intersection point between edges.

4.25 RN Version 0.18.2

Release Date: 2009-11-05

Tar-Kits

Sources for all systems	klayout-0.18.2.tar.gz
WIN32 binaries and DLL's	klayout-0.18.2-win32.zip
WIN32 installer	klayout-0.18.2-win32-install.exe

Features

- Enhancements for building with gcc 4.4.x and Qt 4.5.x.
- OASIS reader now also supports text objects with forward references to text string definitions.

4.26 RN Version 0.18.1

Release Date: 2009-08-02

Tar-Kits

Sources for all systems	klayout-0.18.1.tar.gz
WIN32 binaries and DLL's	klayout-0.18.1-win32.zip
WIN32 installer	klayout-0.18.1-win32-install.exe
Universal binary for Mac OS 10.5.7 - requires Qt 4.5.2	klayout-0.18.1-mac-leopard.gz

Features

- Some bug fixes. For a detailed list of fixed bugs see section 5.13: Version 0.18, Known Bugs, in particular for Mac OS users.

4.27 RN Version 0.18

Release Date: 2009-07-07

Tar-Kits

Sources for all systems	klayout-0.18.tar.gz
WIN32 binaries and DLL's	klayout-0.18-win32.zip
WIN32 installer	klayout-0.18-win32-install.exe

Features

- Some bug fixes. For a detailed list of fixed bugs see section 5.14: Version 0.17.2, Known Bugs.
- Added support for the GDS2 text format. This is a contribution by Romain Gauci from Oscillated Recall Technology, http://www.or-tech.co.jp.
- Session persistence: sessions (files, layers, bookmarks, setup) can be saved and restored. Details for this feature can be found in section 8.3.16: Saving and restoring a session.
- Reader options can be specified in a dialog now, File > Reader Options menu. Some GDS2 specific reader options are available: allow multi XY, allow big records and several box record handling modes. Details for this feature can be found in section 8.1.2: Loading a file.
- GDS2 specific writer options (max vertex option, multi XY record option, max cell name length).
- The stipple palette now is editable. A dialog page has been added File > Setup > Display > Stipple Palette.
- Flatten Cell function. A cell can be flattened which will remove the cell and all the sub-cells unless not specified otherwise. This function can be found in the menus Cell context > Flatten Cell and Edit > Cell > Flatten Cell. Details for this feature can be found in section 9.3.3: Flatten cells.
- The Edit > Selection > Make Cell operation provides a nice origin now. Before, the origin of the new cell was far off sometimes.
- Accept drag & drop of `.lyb`, `.lys`, `.lyp`, `.rb`, `.rbm` and layout files.
- Switched to standard file dialogs where this has not been the case yet. In particular on the Windows platform, the application will behave somewhat more consistent.
- Added an option to "draw only border instances of arrays". This option can be set by the check-box File > Setup > Optimization > Array > Draw only border instances in detailed view.
- Added the capability to define global ruby modules using the `.rbm` extension and putting them into the installation directory.

- Full cell copy & paste functions can now be found in hierarchy panel context menu.
- Enhancement of grid snapping in partial mode. Before, a off-grid vertex could not be brought on-grid because movement was confined to grid steps.
- Number of points for polygons is shown now in the polygon property dialog.
- Hole resolution for GDS2 and OASIS writer. Before, polygons with holes (which can be produced by scripts) were rejected.
- Some bug fixes for clip, using booleans for clip to overcome some problems with hole connectors and spikes - slow but safe.
- Character "#" is no longer used when creating cell variants – instead character "$" is used to enhance the compatibility with other tools which do not allow character "#" as part of cell names.
- On paste, a "fit selection" is done to show what has been pasted. This behavior can be set by check-box File > Setup > Navigation > New Cell > On Cell Change > Fit window to cell when cell is changed.
- Mouse wheel mode is configurable now (shift/ctrl modifier behavior). The Mouse wheel mode can be set by check-box Edit > Setup > Navigation > Zoom and Pan > Mouse wheel alternative mode.
- Option to clear all rulers when cell is changed. This option can be set by check-box File > Setup > Navigation > New Cell > On Cell Change > Clear all rulers.

RBA enhancements

- A new class: ICplxTrans and related functions.
- A recursive shape iterator simplifies hierarchical region queries and "as if flat" traversal of cells.
- Polygon: hull and hole can be assigned now, compress method, point accessors.
- The installation path is now available.
- Added a method which allows to display a message in the status bar.
- The transient selection now is available for RBA procedures.
- New events are generated if selection and transient selection changes.
- Added a generic assign method for copyable objects.
- Added new classes LayerMap and LoadLayoutOptions.
- Added two read methods to Layout class.
- Added a couple of methods to LayerInfo (constructors, compare, ...).
- Added a load_layout method to LayoutView.
- Added more cm_* methods to MainWindow.

4.28 RN Version 0.17.2

Release Date: 2009-04-20

Tar-Kits

Sources for all systems	klayout-0.17.2.tar.gz
WIN32 binaries and DLL's	klayout-0.17.2-win32.zip
WIN32 installer	klayout-0.17.2-win32-install.exe

Features

- Some bugs are fixed now. For a detailed list of fixed bugs see section 5.15: Version 0.17.1, Known Bugs.
- The GDS2 writer now normalizes AREF records. This way compatibility with other EDA systems is enhanced.

4.29 RN Version 0.17.1

Release Date: 2009-03-27

Tar-Kits

Sources for all systems	klayout-0.17.1.tar.gz
WIN32 binaries and DLL's	klayout-0.17.1-win32.zip
WIN32 installer	klayout-0.17.1-win32-install.exe

Features

- The display freezing bug on Windows is fixed now, see section 5.16: Version 0.17, Known Bugs.

4.30 RN Version 0.17

Release Date: 2009-03-23

Tar-Kits

Sources for all systems	klayout-0.17.tar.gz
WIN32 binaries and DLL's	klayout-0.17-win32.zip
WIN32 installer	klayout-0.17-win32-install.exe

Features

- Various layer operations are now available: Boolean operations AND, XOR, NOT, as well as layer merge and sizing. For a detailed description see section 9.3.4: Layer Boolean operations and section 9.3.5: Layer sizing. These operations are also available in RBA, see EdgeProcessor and ShapeProcessor.
- Boolean and sizing operations are also available, see section 9.3.6: Shape-wise Boolean operations and section 9.3.7: Shape-wise sizing.
- Objects can now be aligned. For a detailed description of the alignment function see section 9.3.8: Object alignment.
- The cell origin can be adjusted relative to the cell's bounding box. For a detailed description see section 9.3.10: Cell origin adjustment.
- A "corner rounding" function has been implemented to support soft-cornered layout which is common in power applications. For a detailed description see section 9.3.9: Corner rounding.

- Various layer operations are now implemented in edit mode: clear layer, delete layer and edit layer properties. For a detailed description see section 9.3.11: Layer operations: clear, delete, edit specification.
- The selection can now be scaled Edit > Selection > Scale.
- An option is available that allows to select all hierarchy levels automatically when a cell is opened Edit > Setup > Display > General > Hierarchy Depth > Initial hierarchy depth when opening a new panel. Checkbox File > Setup > Navigation > New Cell > On Cell Change > Select all hierarchy levels must be unchecked.
- Various bug fixes, see section 5.16: Version 0.17, Known Bugs.

4.31 RN Version 0.16.1

Release Date: 2009-01-07

Tar-Kits

Sources for all systems	klayout-0.16.1.tar.gz
WIN32 binaries and DLL's	klayout-0.16.1-win32.zip
WIN32 installer	klayout-0.16.1-win32-install.exe

Features

- Various bug fixes, see section 5.17: Version 0.16.1, Known Bugs in Version 0.16.1.

4.32 RN Version 0.16

Release Date: 2008-12-27

Tar-Kits

Sources for all systems	klayout-0.16.tar.gz
WIN32 binaries and DLL's	klayout-0.16-win32.zip
WIN32 installer	klayout-0.16-win32-install.exe

Features

- Some new editing capabilities: flatten, make cell, clear layer and a clip function.
- Some RBA extensions, in particular the ability to modify layout by deleting shapes and instances, replacing and transforming them, changing property handles etc.
- Support for global preset of configuration (through a file called `layviewrc` beside the executable binary) and a global RBA initialization file (a file called `rbainit` beside the executable binary).
- Transient selection: indicates by a faint selection marker what object is below the mouse (can be disabled).
- The layer specification (layer, datatype, name) can be now be edited which allows to move a complete layer to a different one.
- Undo buffering can now be disabled on the command line with the "-i" option. This saves the memory otherwise required for storing the replay information.

- Directional constraints can be modified by using the Shift and Ctrl modifiers on various operations such as rulers, movement, drawing etc.
- Now there is a "most recently used" list File > Open Recent which shows the last layouts opened for easy re-opening.
- Reduced memory requirements for particular OASIS constructs (random repetitions)
- The instance browser now has a "Choose cell" button where the cell can be chosen whose instances will be presented.
- For most editing operations, the status bar will now indicate more detailed information such as move distance.
- Pasted shapes and instances now are selected initially.
- Enhanced OASIS compression mode (can be chosen from the options dialog on saving). Reduces file size considerably by creating regular shape arrays if possible at the cost of higher memory consumption and longer writing times.
- Various bug fixes, see section 5.18: Version 0.16, Known Bugs.

4.33 RN Version 0.15

Release Date: 2008-08-16

Tar-Kits

Sources for all systems	klayout-0.15.tar.gz
WIN32 binaries and DLL's	klayout-0.15-win32.zip
WIN32 installer	klayout-0.15-win32-install.exe

Features

- Editing capabilities. **KLayout** in it's basic form still operates as pure viewer. However, a mode is available that enables editing capabilities. See chapter 9: Quick Start Manual – Editor Mode for a more detailed description.
- Some RBA extensions: i.e. conversion from "double" type polygons to "integer" type ones.
- A default layer table can be configured by File, Setup, Application, Layer List, ☐Use default layer table menu item. This table will be loaded whenever a layout is opened or created.
- A installer for Windows is provided now.
- Properties are supported on instances now as well.
- Code is compatible with gcc 4.3.0 now.
- Enhanced compatibility with SunStudio 11 compiler (in particular in RBA).
- Various bug fixes (i.e. in GDS reader and OASIS writer).

4.34 RN Version 0.14

Release Date: 2008-04-04

Tar-Kits

Sources for all systems	klayout-0.14.tar.gz
WIN32 binaries and DLL's	klayout-0.14-win32.zip

Features

- Several RBA extensions: i.e. alternative methods added replacing different "new" variants by one new method. Markers now can be filled.
- Fixes for the bugs mentioned in the issues list.
- OASIS and GDS writing capabilities. Menu functions are provided to save a layout and single cells. See section 8.3.15: Saving a layout or parts of it. RBA extensions added that allow to write a layout.

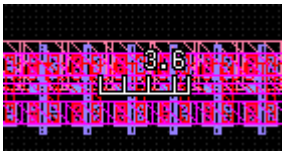

Figure 4.1. Ruler with halo

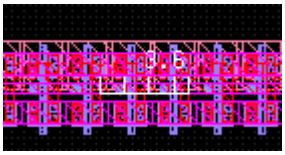

Figure 4.2. Ruler without halo

4.35 RN Version 0.13

Release Date: 2008-01-16

Tar-Kits

Sources for all systems	klayout-0.13.tar.gz
WIN32 binaries and DLL's	klayout-0.13-win32.zip

Features

- A couple of RBA extensions: API's for shape properties, rulers and geometry selections.
- A bug fix for the OASIS reader (sometimes crashing the reader on CBLOCK-compressed input)
- A function to descend into a cell in the hierarchy but keeping the context, which is shown in dimmed colors. This function can be found in Display > Descend. To use this function, select a shape or instance inside the cell into which to descend. Display > Descend then enters the next cell along the path that leads to the selected shape or instance.
- The mouse wheel can be used to zoom in and out.
- Shift+right mouse button pressed now allows to "drag" layout in a "Google maps" fashion.
- Copy & paste now is available for layers as well.
- A ruby console is provided where ruby commands can be executed interactively Tools > RBA Console.
- The way how the selection is displayed can be configured now (color, line width and vertex size).
- The interrupt signal now is enabled again even if Ruby is activated (Ctrl + C now works again).

4.36 RN Version 0.12

Release Date: 2007-11-02

Tar-Kits

Sources for all systems	klayout-0.12.tar.gz
WIN32 binaries and DLL's	klayout-0.12-win32.zip

Features

- Support for arbitrary angles on cell instances.
- A bug fix for the OASIS reader (CTRAPEZOIDS have not been read-in in some cases). The OASIS reader now uses shape arrays to achieve are smaller memory footprint in some cases.
- Multiple transformations are now allowed on layer source expressions, see section 6.3: Transformations in **KLayout** for a detailed description.
- Rulers now are configurable to a large extent. Multiple ruler templates may be defined from which a certain ruler type can be chosen. Some of the new ruler styles are no longer "rulers" but rather "annotations". See section 8.3.7: Ruler properties for a description of that feature.
- Last, but not least: **KLayout** now can be scripted to some extend using Ruby as the scripting language. For more information about this feature, see chapter 11: RBA Introduction.

4.37 RN Version 0.11

Release Date: 2007-06-26

Tar-Kits

Sources for all systems	klayout-0.11.tar.gz
WIN32 binaries and DLL's	klayout-0.11-win32.zip

Features

- Some bugs were fixed in the OASIS implementation. CBLOCK compression now is supported. Forward cell name references (numerical ID is used before being associated with a string) are supported as now. File global properties are now read correctly. A bug reading certain CTRAPEZOID objects was fixed. (Still, forward references are not allowed for text/property strings or property names).
- A layout properties dialog has been added that shows basic properties of the layout such as top cells, layers and others.
- The Display > Select Cell dialog has been enhanced with the capability to hide and show multiple cells in the cell list.
- A "zoom fit" mouse gesture was implemented: moving the mouse up and right with the right mouse button pressed now fits the current cell into the window. Moving up and left still zooms out.
- A bug in the GDS2 reader, dropping polygon points in rare cases, was fixed.

4.38 RN Version 0.10

Release Date: 2007-05

Tar-Kits

Sources for all systems klayout-0.10.tar.gz

Features

- The build script has been enhanced and supports Qt installations with different locations for binaries, libraries and headers. See chapter 2: Download and Build for details about this.
- OASIS files can be read as well now. The reader automatically determines the kind of file. OASIS layer names are supported. Due to the complexity of the OASIS specification, or more precisely the effort required to test a reader for meeting the specification, OASIS support currently is regarded to be in "beta" state.
- Layers can now be organized hierarchically. Layers can be grouped which allows to control visibility and other properties for the group as a whole. Layers can be removed from the layer view list or new layer views can be created.
- The layout can be transformed (rotated, shifted, mirrored) now. This transformation is specified in the layer panel as a part of the layer source specification. An optional transformation can be applied per layer or layer group. This way for example, different layouts can be aligned over each other.
- Shape properties now are supported. Each shape may carry optional properties that are read from GDS or OASIS files. A property selector can be applied per layer view, so that a layer only shows these shapes that match the given property selection.
- As an experimental feature, the inverse layout tree can be visualized by allowing the minimum hierarchy level to go into the negative range. The effect of this is, that a cell is shown in the context of all of it's direct parents, if the first hierarchy level is set to -1. If set to -2, the cell is shown in the context of all of it's parents and grandparents and so on. Since there is no particular optimization for this feature, performance may be poor, if a cell must be painted in a huge number of contexts. In addition, the context displayed does not participate in selection or ruler snapping.

4.39 RN Version 0.9

Release Date: 2006-04

Tar-Kits

Sources for all systems klayout-0.9.tar.gz

The first official release.

Chapter 5

Known Bugs and Issues

These are some lists of known, more or less, serious issues.

Please give me a hint, if there are some more. Or as Einstein said: "The only source of knowledge is experience" ...

Content

5.1 Version 0.21.5

5.1.1 DXF reader

Bulges are not supported for poly lines.

5.1.2 Performance issues on select

Select (transient or on click) is slow in some cases. This happens in particular if cells in the hierarchy overlap heavily and many layers are present.

5.1.3 RBA:Edge.intersect? delivers wrong results when the edges are collinear

The edges will be reported to intersect even if they don't. A workaround is to test if the bounding boxes overlap and the edges intersect.

5.2 Version 0.21.4

5.2.1 DXF reader

Some POLYLINE examples were using a global width which was overridden by a per-vertex width. In that case the global width is taken rather than the correct local one.

5.2.2 Problems with non-English locales and UTF-8 file names on Linux

On KDE, files cannot be opened when the path contains non-ASCII characters on UTF-8 file systems. In some cases, the decimal point is inconsistently "," instead of ".", which is the standard for **KLayout**.

5.3 Version 0.21.3

5.3.1 CIF reader

The CIF reader currently does not understand "DS" commands without a scale specification (i.e. "DS 20"). It always requires two additional numbers specifying the scale as a ratio of two integers (i.e. "DS 20 1 10").

5.3.2 Rotate methods swapped

The clockwise rotate method rotates counterclockwise and vice versa.

5.3.3 "Draw border instances of arrays" feature broken

Much is drawn and much more isn't ....

5.3.4 Ruby crash

On some systems, the program crashes when a ruby script is loaded (i.e. with the -rm option) with a message "[BUG] terminated node (0x2a9708ca70)" or similar. This seems in particular to **Comment: end of sentence missing**

5.4 Version 0.21.2

5.4.1 DXF reader still not complete

In particular, interpretation of POLYLINE and LWPOLYLINE entities is not clear yet. In comparison to other converters, no merging of separate lines into polygons is provided.

5.5 Version 0.21.1

5.5.1 RBA: RdbItem.each_value is not working on const objects

This is important, because const RdbItem objects are commonly encountered when scanning through a marker database with RBA.

5.5.2 DXF reader still not complete

The interpretation of certain features (i.e. array instances, extrusion direction) is not clear yet.

5.5.3 Layer mapping broken for DXF and CIF readers, writers

Incorrect layers are written for example when a layer subset is written. In addition, mapping or selection of input layers does not work correctly for DXF and CIF readers.

5.6 Version 0.21

5.6.1 Persistence of reader options is broken

The program does not remember reader options when the dialog is closed.

5.6.2 RBA: each_selected is broken

A segmentation fault occurs on Windows in the each_selected method of LayoutView.

5.6.3 DXF and CIF readers and writers incomplete

DXF and CIF readers and writers implement only very basic features. Some important capabilities are missing, in particular for the DXF part.

5.7 Version 0.20.1

5.7.1 Program crashes when the marker browser is opened

When the marker browser is opened with a maximum number of markers set to a value not equal to 1000 (the default), the program crashes if the marker browser window is opened from the menu (it works when the marker database is loaded from the command line with the -m switch).

5.8 Version 0.20

5.8.1 Net tracing does not extract net correctly

This bug is related to branching conditions. In such cases, a net might not be extracted correctly.

5.8.2 Gerber reader does not correctly read certain macros

The "outline" macro is read as a thin outline also in the "closed" case by the RS274X reader.

5.9 Version 0.19.3

5.9.1 Polygon cut algorithm for reducing the number of points per polygon in the GDS2 writer

When the GDS2 writer has to reduce the number of points of a polygon, it will cut the polygon into smaller pieces. Under some circumstances, this algorithm fails. To avoid this problem, use the Multi XY record mode if possible.

5.10 Version 0.19.2

5.10.1 Crashes on Qt 4.6.0

Due to a bug in Qt's QPixmap constructor in Qt 4.6.0, **KLayout** does not work with this version.

5.10.2 Ruby modules not loaded from the installation path on UNIX

In most cases, ruby modules are not looked for in the wrong path. The intention was to search for `.rbm` files in the directory where **KLayout** is installed. Instead, the current or any other directory is searched depending on how the **KLayout** executable is specified on the command line.

5.10.3 OASIS reader too picky

The OASIS reader does not accept files with forward references of the special "S_GDS_PROPNAME" property name.

5.11 Version 0.19.1

5.11.1 "Test for shapes in view" feature does not work properly for AREF's

In some cases, array references are not considered and layers appear to be empty even if they are not.

5.11.2 RBA scripts crash in tight loops on Ruby 1.8.7 (i.e. Ubuntu 9.10)

That is a ruby problem, see also http://www.ruby-forum.com/topic/198545. The problem is known and a new libruby version should be available soon.

5.11.3 GDS text reader problems

In some build environments, problems have been encountered with GDS text files with negative values.

5.11.4 Interactive stretching of images is broken

The interactive stretching of images with the square handles is sometimes leading to invalid results and does not work properly.

5.12 Version 0.19

5.12.1 Crash when selecting "..." node in the marker browser item list

The item list is abbreviated using a dummy item labeled "...". When clicking at this item, the application crashes.

5.12.2 "Test for shapes in view" feature in layer list is extremely slow in some cases

This feature marks a layer "unused" when no shape is shown on that layer in the view area. Currently, the application becomes very slow in some cases when this option is used.

5.13 Version 0.18

5.13.1 Crash when selecting "instance" mode on empty layout

When the layout is empty (no cell present, i.e. top cell was deleted) and "Instance" mode is selected in editor mode, the program crashes with an internal error.

5.13.2 Issues on Mac OS X

There are still some issues on Mac OS X, as well for the build as for the program itself — in particular with Qt 4.5.x. For example, with accessibility enabled, the program crashes when a file is loaded. This will be fixed in version 0.18.1.

5.14 Version 0.17.2

5.14.1 Sizing bugs

The sizing function sometimes produces invalid results, in particular when doing a strong undersize.

5.14.2 Build not working for Mac OS X

The 'ar' call has been changed such that the Mac OS X build should work now (not tested since no test system was available).

5.14.3 Crash when double-clicking a path end in partial mode

When double-clicking on a path end in partial edit mode (dragging just the path end), the program crashed in some cases.

5.14.4 "Fit selection" is not working properly

Not all instances are taken into account.

5.14.5 Wrong DBU read from GDS2 files

The DBU per user unit is used, which is not correct. Instead the DBU per meter unit should be used.

5.14.6 Round paths are not written properly to OASIS files

5.14.7 Windows repaint problem for hidden canvas content

Strange effects occur when a non-modal front dialog is moved over the canvas area.

5.14.8 Space representation in vector fonts

Space characters are not represented.

5.15 Version 0.17.1

5.15.1 Program hangs if the properties dialog is closed with the system menu

When the properties dialog is closed using the system menu (the "X" button in the window title bar), **KLayout** goes into an unusable state. This does not happen if the dialog is closed using the "Close" button.

5.15.2 Program crashes if many text objects have identical location

This happens if more than 100 text objects are present that have identical locations.

5.15.3 OASIS reader problems when property name and string ID's are defined after they are used

This was a known limitation but came up recently in a certain application.

5.15.4 AREF row and column description was swapped and misleading

In GDS files, row and column vectors can be arbitrary x/y value pairs. However, some tools implement a more strict interpretation in which only orthogonal row and column vectors are allowed. Also, row and column must be oriented in a certain way. In addition, the description of row and column vectors is swapped.

5.16 Version 0.17

5.16.1 Display freezes on some Windows installations

Apparently due to a problem with Qt's grabMouse function on some Windows installations the display freezes when a zoom box or selection box is opened. By switching to the Task manager using "Ctrl+Alt+Del", the display can be unfrozen but zoom or selection operations a not possible. This problem existed in all previous versions as well and apparently occurred in particular on Windows XP.

5.17 Version 0.16.1

5.17.1 Some flaws in partial edit mode and polygon or path creation

In certain cases, the closing point of polygons was not created correctly, path segments did not snap correctly to 45 degree edges or partial edit mode was behaving in a strange way.

5.17.2 Order of recent file list was latest last

...which is contrary to what other programs implement.

5.17.3 Selection of very large arrays happened to be very slow

This happened because many markers have been drawn for such arrays. This has been changed such that the array is not drawn as individual markers for large arrays. Instead, a representative geometrical description is given.

5.18 Version 0.16

5.18.1 Compile problems when ruby support is not enabled

5.18.2 "open recent" function is not working correctly on Windows

5.18.3 "change layer" function is not working properly

5.19 Version 0.15

5.19.1 Child cells are shown multiple times in cell hierarchy

Under certain circumstances, child cells are shown multiple times in the cell tree, i.e. a cell "A" which is a child of "TOP" might appear multiple times in the tree below "TOP". This is not intended — child cells are supposed to appear just once, even if instantiated multiple times.

5.19.2 "Save" saves all layers if none should be saved

If the layers to be saved are confined, i.e. to visible ones, it may happen that, if no layer is visible for example, all layers are saved instead. A workaround is to create a new layer (i.e. layer 1000, datatype 0) and save it. Such an empty layer will be saved but won't appear in the OASIS or GDS2 file, since it does not contain any shapes.

5.19.3 Text objects are not shown correctly if a scalable font is selected for them

Depending on the transformation of the text, the text may appear at unexpected locations for example. A workaround is to use the "default" font.

5.20 Version 0.14

5.20.1 Crash on Windows when the program is called first time

On windows, crashes have been observed, when the program is started the first time after installation. This indicates some problem with Qt installation in the registry. However, this bug was not tracked down yet, because it is not easy to reproduce. Since it only happens once, it is not considered pretty serious.

5.21 Version 0.13

5.21.1 Crash on Windows when the layer list becomes very small

The program crashes on Windows, if the layer list becomes too small to be displayed. This happens for example, if at the default size of the program window, the color panel, the frame color panel and the stipple panel are opened in that order. Then, the height of the layer list becomes a few pixels and the program crashes.

5.21.2 KLayout does not start on some platforms and exits with a segmentation fault

This problem has been found on the 64bit Ubuntu 7.10 platform for example. The program does not start up and exits immediately with a segmentation fault.

Here is a small patch that fixes that problem:

Console Input 5.1: C++ Patch – file layApplication.h. line 53, Version 0.13

```
/* use following code */
Application (int &argc, char *argv []);
/* instead of: */
Application (int argc, char *argv []);
```

Console Input 5.2: C++ Patch – file layApplication.cc, line 50, Version 0.13

```
/* use following code */
Application::Application (int &argc, char *argv [])
/* instead of: */
Application::Application (int argc, char *argv [])
```

5.22 General

5.22.1 Layout loading time

The viewer internally builds look-up tables for fast geometrical look-up in huge data sets. This "sorting" procedure takes considerable amount of time when loading a layout. How much time it takes depends on the "flatness" of a layout. On the other hand, these structures allow fast access to small clips of the layout.

5.22.2 Drawing speed versus high display precision

The objective of high display precision sometimes competes with high drawing speed. Usually however, drawing performance is quite good.

Part II

Documentation

Chapter 6

Resources

Content

6.1 Typographic Conventions

Comment: Where to place this section?

It is essential that the presentation of the very different material, covered by this document, conveys its function immediately in the framework of the text. Therefore, this section presents the typographic conventions used in this document.

Comment: verbalize the single items below into full sentences

A cross reference, e.g. to this section, is presented as section 6.1: Typographic Conventions, while a reference to an internet page can be displayed as URL, like this http://www.klayout.de/, or as named reference, like this **KLayout**'s Home Page.

A **KLayout** menu item is displayed like this Menu Item.
A menu item with sub menu item is shown as Main Menu > Sub Menu.
A menu item or option with check box is given as ☐ check this one or as ○ an option if only one option is selectable from a list.

A dialog *Dialog Name* or a dialog section *Input Options* is given in this way.
In case the key "Ctrl" or the button "OK" should be pressed this is visualized as key Ctrl or button OK.

A path and file `C:/Program Files/KLayout/klayout.exe`, as well as a file extension `lyp` is written in a mono spaced font.

Sometimes an important hint is given which looks like

Hint: This is a hint

6.1.1 Input Dialog Conventions

Angle brackets <> encloses parameters, e.g. **<layer>/<datatype>** – first the layer number, second the data type.

Curly brackets { } encloses optional entries, e.g. **<layer>/<datatype>{@<layout index>}** – the layout index. This is in opposite to the usual convention where square bracket are used, e.g. for displaying console command input. But become necessary because the input dialog uses square bracket as active characters.

Bar or Pipe | separates parameters given in a list from which only one can be selected at time, e.g. **r<angle>|m<angle>**.

6.1.2 RBA Typographic Conventions

The typographic conventions for the ruby based automation API are as listed below. Unfortunately, they doesn't math the conventions used in ruby code listings at present. For a detailed description see section 11.6: Brief overview over the API.

RBA Class A class name.

RBA Method A method name.

[const] The constant attribute of a method.

[static] The static attribute of a method.

[event] The event attribute of a method.

yield The iterator attribute of a method.

const A constant value like Π.

ref A reference, e.g. for return values.

boolean A Boolean value like true or false.

integer An integer value like 10. **Comment: explain in more detail, sign, bit count**

- **unsigned** Explicit an unsigned integer.
- **long** Explicit a long integer.
- **long long** Explicit a double long integer.
- **double** Explicit a double integer, i.e. a floating point value **Comment: same as above?** .

string A string like **KLayout**.

value A value like trans, in this case a transformation expression.

6.1.3 Listing Conventions

Console Input 6.1: Typographic Conventions Example – Console Input

```
klayout [-<options>] [<file>] ..
```

Angle brackets < > encloses parameters.

Square brackets [ ] encloses optional entries.

Bar or Pipe | separates parameters in a list from which only one can be selected.

Console Input 6.2: Typographic Conventions Example – XML File

```
<description>XML File Typographic Example</description>
```

Console Input 6.3: Typographic Conventions Example – DXF File

```
<Group-Code> <Value>
```

Console Input 6.4: Typographic Conventions Example – C++ File

```
/* C++ File Typographic Example */
Application (int &argc, char *argv []);
```

Console Input 6.5: Typographic Conventions Example – Dialog Input

```
{ ( {<dx>,<dy>} {r<angle> | m<angle>} {*<mag>} ) }
```

Console Input 6.6: Typographic Conventions Example – Ruby Code

```
# Comment
RBA::Application.instance.exec
```

6.2 Command-line arguments

Following a brief description of **KLayout**'s command-line options.

KLayout's command line basically looks like this:

Console Input 6.7: **KLayout** Command Line Input – Basics

```
klayout [-<options>] [<file>] ..
```

Options start with a hyphen ("-") and can be mixed with file names. Files given on the command line without an option are treated as layout files (GDS, OASIS, ...). Each option must be specified separately, i.e. "-ne" is not option "n" and "e". Option arguments must be separated by a space from the option itself. For example:

Console Input 6.8: **KLayout** Command Line Input – Example

```
klayout  -s file1.gds file2.gds -l layers.lyp
```

This command will open `file1.gds` and `file2.gds` in the same view (option "-s") and use the layer properties file `layers.lyp`.

A detail description of **KLayout**'s command-line options follows below.

6.2.1 General Options

-c <config file> Use the specified configuration file (reading it on start and writing it on exit) instead of the default configuration file. This option allows to switch between different configurations.

-d <debug level> Controls the verbosity of the log output. Values are:

0	silent
10	basic info
11	basic info plus basic timing
20	detailed info

	21 detailed info plus detailed timing **up to 40** more detailed info plus detailed timing **41** for noisy log output and timing respectively
-e	Enter edit mode even if non-edit mode was specified in the configuration as default mode.
-ne	Enter viewer mode even if edit mode was specified in the configuration as default mode. If neither option "-e" nor "-ne" is specified, the default mode from the configuration will be used.
-i	Disable undo buffering (less memory requirements).
-ni	Enable undo buffering. This is the default. This option overrides previously set "-i" options.
-l <lyp file>	Use the specified layer properties file instead of the default layer properties.
-lx	Used with option "-l": add other layers to the layer properties even if they are not defined in the properties file.
-lf	Used with option "-l": use the lyp file as it is (no expansion to multiple layouts).
-m <database file>	Load the given report database together with the previously defined layout. This option must follow a layout file argument.
-p <plugin>	Load the plugin (a shared object). This option can be used multiple times.
-r <script>	Run the given Ruby script in interpreter mode. In that mode, **KLayout** will exit after the script is executed. To start **KLayout**, the script must contain a Application.exec call. The script is executed after all other requisites from the command line have been loaded (files, plug-ins etc.) This option can be combined with "-z" (no GUI). That way, **KLayout** is converted into a ruby interpreter.
-rm <script>	Run the given Ruby script before **KLayout** starts the user interface. In contrast to option "-r", **KLayout** continues normal execution after the script is executed successfully. This is the preferred way to install user interface add-ons ("Modules"). In addition to the modules specified by "-rm", **KLayout** collects files with extension `.rbm` from various places, i.e. the place specified with $KLAYOUTPATH **Comment:** $KLAYOUT_PATH ? on Unix or the installation folder on Windows.
-rd <name>=<value>	Define the variable in the Ruby context with the given string value. The variable will be accessible as "$name".
-s	Load files into same view.
-u <file name>	Restore the session from the given session file.
-v	Print program version and exit.
-x	Synchronous drawing mode (non-threaded). This mode can be useful if scripts are run which produce screen snapshots. By using this option is made sure that all drawing operations have finished before the snapshot method returns.
-z	Non-GUI mode. **KLayout** will not bring up the user interface. See the "-r" option for useful applications of this option.

6.2.2 Special Options

-gr <file name>	Record GUI actions in the given file for test purposes.
-gp <file name>	Replay the GUI actions from the given file for test purposes.
-gb <line number>	Stop replaying GUI actions at the given line for test purposes.
-gx <milliseconds>	Replay rate for GUI test file for test purposes.
-gi	Incremental logs on the GUI record file (crash safe logging).
-rx	Ignore global `rbainit` and `.rbm` files. Used to establish a defined basis for tests.

6.3 Transformations in KLayout

A specification of affine 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.

Figure 6.1 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

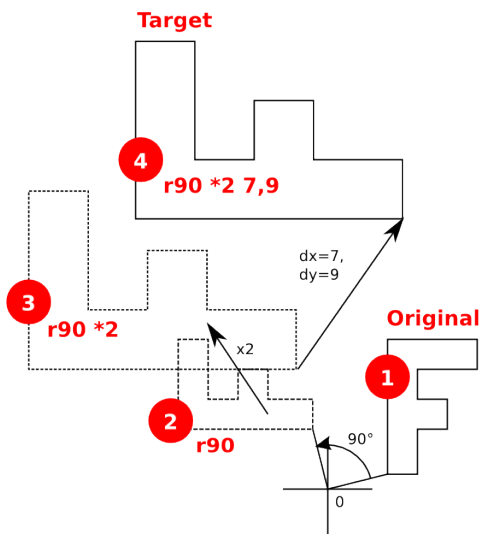

Figure 6.1. Illustration of Transformation – Overview

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	A shift by 100 units up-wards.
r90	A 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". Figure 6.2 illustrates rotation and mirroring with the eight basic transformations involving rotations by multiples of 90 degree:

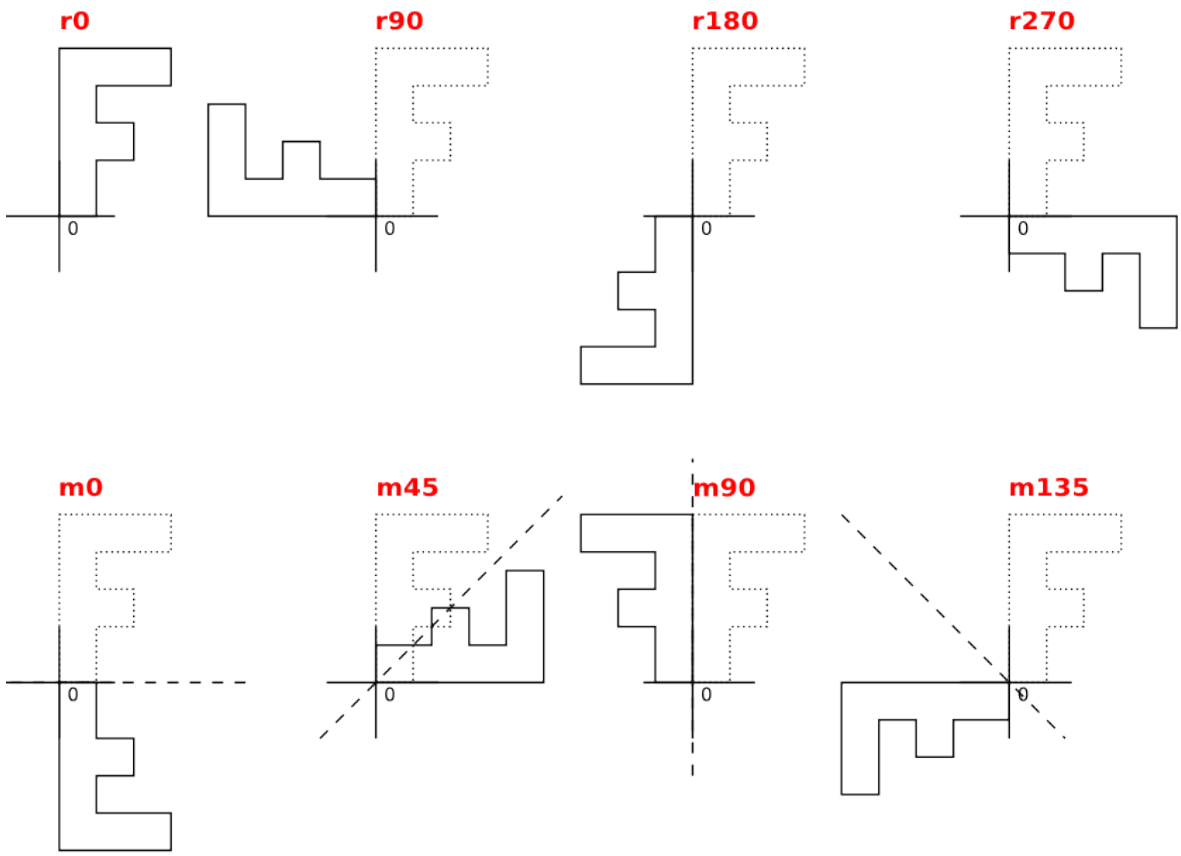

Figure 6.2. Illustration of Transformation – Basics

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.

6.4 RDB format

A description of the report database format.

This is a brief description of the report database format used by **KLayout** to represent the content of a report database. **KLayout** uses a report database to present results of checks or extraction operations. A report database can be viewed with the marker browser, available in the Tools > Verification menu. **KLayout** can import other report database formats. Writing is supported only in the format described here. This description covers the structure of the file. This structure closely matches the internal structure (for example accessible through RBA), and this document may be helpful to understand that internal API as well.

6.4.1 Basic structure

The suffix used by **KLayout** for report databases is `.lyrdb`. The file format is XML representing the object structure of the report database. The root element is "report-database". This is an abbreviated sample file:

Console Input 6.9: XML File – Report Database Sample

```
<?xml version="1.0" encoding="utf-8"?>
<report-database>
 <description>Diff of 'x.gds, Cell RINGO' vs. 'x.gds[1], Cell INV2'</description>
 <original-file/>
 <generator/>
 <top-cell>RINGO</top-cell>
 <tags>
  <tag>
   <name>red</name>
   <description>Red flag</description>
  </tag>
  ...
 </tags>
 <categories>
  <category>
   <name>1/0</name>
   <description>Differences in layer 1/0</description>
   <categories>
    <category>
     <name>A</name>
     <description>Shapes in A but not in B, on Layer 1/0</description>
    </category>
    ...
   </categories>
  </category>
 </categories>
 <cells>
  <cell>
   <name>RINGO</name>
   <variant>1</variant>
   <references>
    ...
   </references>
  </cell>
  ...
 </cells>
 <items>
  <item>
   <tags/>
   <category>'1/0'.A</category>
   <cell>RINGO:1</cell>
   <visited>true</visited>
   <multiplicity>1</multiplicity>
   <image/>
   <values>
    <value>text: 'item: polygon'</value>
    <value>polygon: (1.4,1.8;-1.4,1.8;-1.4,3.8;1.4,3.8)</value>
   </values>
  </item>
  ...
 </items>
</report-database>
```

The components of a report database are:

items Items represent one basic element of the report. Usually an item represents a marker in-

dicating a geometric entity with a shape. Items can also represent texts such as errors or warnings not related to geometry. Items carry information with a set of values. Values are the parts forming the information of an item. Currently, each item has an ordered list of values. **KLayout** does not make an assumption about the type or order of the values. Items can also be flagged with "tags" (see below) and have an image attached. Currently an image is a special property of the item, not part of the values.

values A value represents an information part of the database item. In the report database context, a value is a string encoding the type of the value and the actual value.

categories The report database defines a hierarchy of categories and sub-categories. Each database item is associated with a category or sub-category within that tree.

cells The report database also defines a hierarchy of cells. The cell hierarchy may be complete, i.e. a copy of a layout hierarchy or specify representative instances or no instances at all. Database items can be associated with a cell which allows **KLayout** to display a marker in the context of a certain cell. **KLayout** supports cell variants. A cell is not only identified with a name by may also carry a variant identifier. An item can be associated with a particular variant of a cell if necessary.

tags Tags are basically flags that can be attached to database items. **KLayout** uses tags to mark items as "waived" or "important".

Figure 6.3 shows how the marker database objects are related with elements of the marker browser dialog.

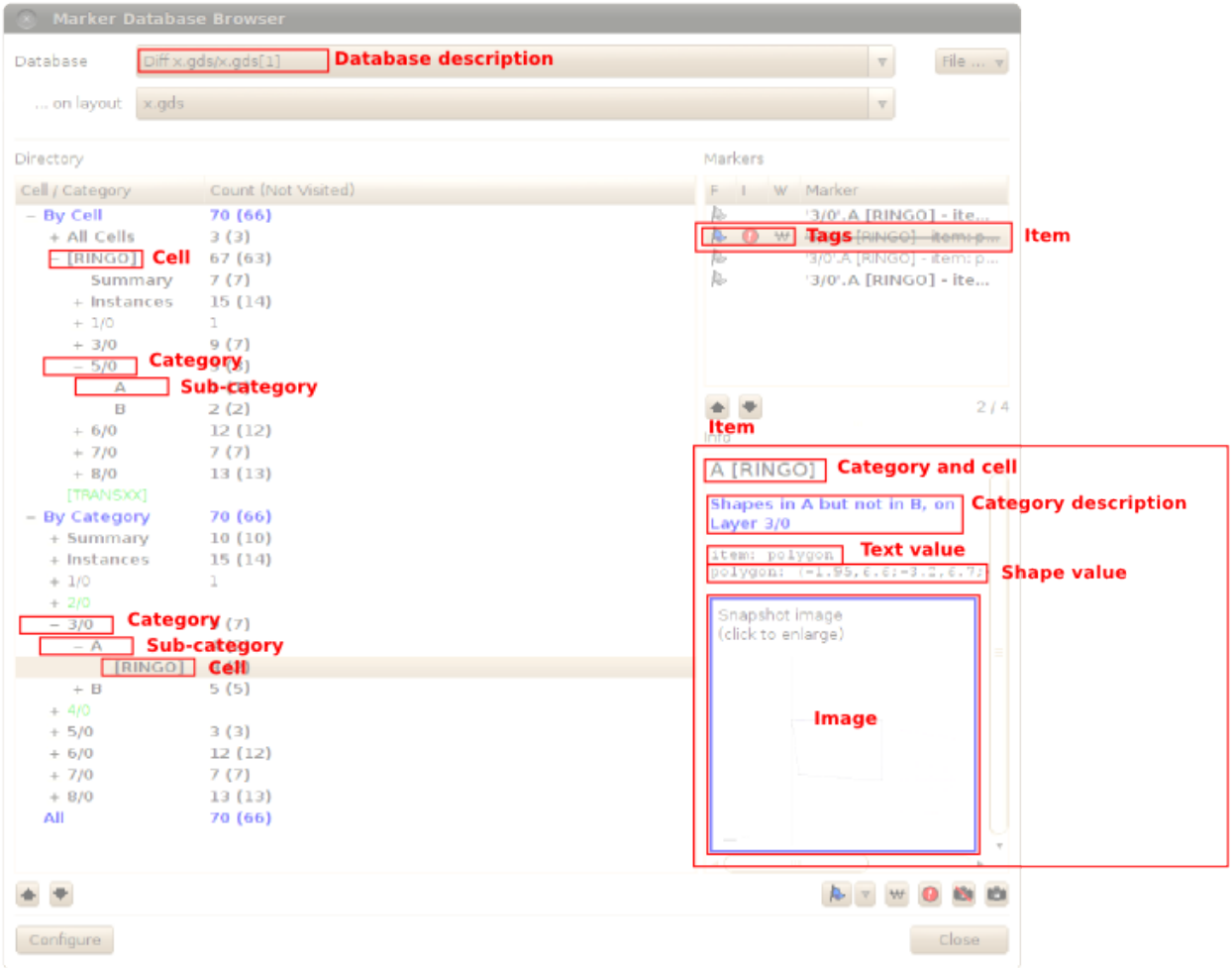

Figure 6.3. Marker Database Browser Dialog

6.4.2 Detailed description

The marker databases structure is conveniently described with a UML class diagram, see fig. 6.4. It shows the objects of the database and their relationship. Aggregation in XML is implemented by including the object in the XML, association is implemented with an element carrying a suitable reference string. In the class diagram, some container classes appear (i.e. "Cells") which represent a list of individual objects (in that case "Cell"). They are present to match the XML structure, which uses an enclosing element around the list (in that example "<cells>...</cells>").

The attribute names in the UML class diagram match the XML element names where the underscore is replaced by the hyphen (i.e. attribute "top_cell" is represented in XML as "top-cell"). This convention is a tribute to the usual XML convention which contrasts with the attribute names used in the code. The

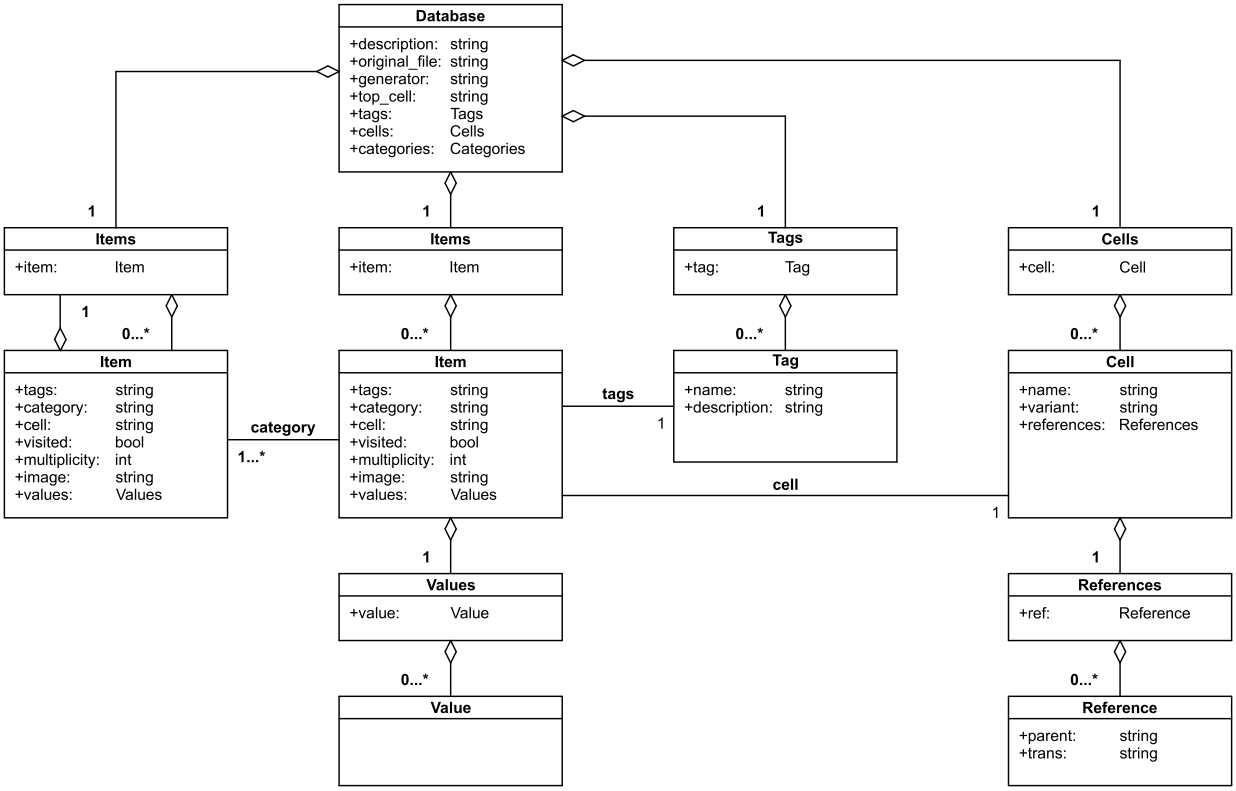

Figure 6.4. Marker Database Browser – UML Diagram

following is a detailed description of some classes and important attributes. As a general rule, the marker database uses micron units. It is independent of the layout database unit.

6.4.2.1 Class Database (element "report-database")

This is the root element of the XML file and the object representing the whole database. It has the following attributes (the XML element names are shown):

description A general description text shown in the marker database browser for that database.

original-file (optional) The file from which the report was generated.

generator (optional) A string describing information about the module that generated the report database. It is intended to formalize the generator information so it is possible to re-run a reporting tool.

top-cell The name of the top cell in the layout from which the report was created from.

tags A list of Tag objects (child elements "tag") declaring the tag identifiers available.

cells A list of Cell objects (child elements "cell") declaring the cells, optionally specifying a partial or complete hierarchy in the form of a cell graph.

category A list of Category objects (child elements "category") declaring the first level of categories.

6.4.2.2 Class Category (element "category")

A Category object specifies one category and optional sub-categories forming a branch in the tree of categories. It has the following attributes (the XML element names are shown):

name An arbitrary string identifying a category in a "category path" (see RdbItem class). The name is also shown in the category tree. A category name must be unique in the context of the category list (not across the category hierarchy).

description A description string shown in the title of the item panel.

sub-categories An optional list of child categories (further Category objects).

6.4.2.3 Class Tag (element "tag")

A Tag object declares a tag for the items. It has the following attributes (the XML element names are shown):

name An arbitrary string identifying a tag in item's tag list. The tag name must be unique in the context of the database.

description An optional description string.

6.4.2.4 Class Cell (element "cell")

A Cell object declares a cell and optionally the cell's relationship, hence forming a cell graph. It has the following attributes (the XML element names are shown):

name An arbitrary string identifying the cell. The cell name is matched against cell names in the layout when displaying geometrical markers to locate the marker in the layout. The instantiation information is used to locate the marker in the top-level context if the specific cell is not available. A geometrical marker is always specified in the context of the cell it refers to.

variant An arbitrary string identifying the variant of the cell.

references An list of Reference objects which specifies from which cells and how this cell is instantiated.

Hint: **If a cell exists with an empty name, it is displayed as "All cells". All items which are not associated with a cell (i.e. global warning messages), can be associated with this special cell by specifying an empty cell name for that item.**

6.4.2.5 Class Reference (element "reference")

A Reference object represents a cell reference and states parent cell and transformation. It has the following attributes (the XML element names are shown):

parent The parent cell name. If multiple variants exist for a cell, this must be a qualified name: the cell name, a colon and the variant id (for example "A:1").

trans The transformation by which this cell's content is transformed into the parent cell **Comment: (??? correct?)** . The transformation is specified in **KLayout**'s transformation notation.

The transformation specification follows the standard notation in **KLayout**, see chapter 2: Download and Build. For example, "r90 *1 17.5,-25" describes a rotation by 90 degree (in the mathematical sense), no scaling and a displacement of 17.5 micron in x-direction and -25 micron in y direction. Since "*1" is the default, this is equivalent to "r90 17.5,-25". Also, the order of the parts is not important, so "17.5,-25 r90" gives the same results.

6.4.2.6 Class Item (element "item")

Items are the basic elements of the report database. An Item class has the following attributes (the XML element names are shown):

tags A comma-separated list of tag names attached to this item.

category A category path describing the category this item is attached to. A category path is a list of category names joined with dots. For example "A.B" is the "B" sub-category of the "A" category. The category path notation allows to quote category names by single or double quotes so that category names can also contain dots.

cell The cell that this item is associated with. The cell name can be empty indicating that the item is not associated with a specific cell. In that case, the item is listed under "All cells". Currently, in that case a dummy cell declaration is required that declares a cell without a name (see Cell class). The cell name is a "qualified name". That means it consists of a cell name, optionally followed by a colon and the variant string. For example, "A:1" is the "1" variant of the "A" cell. This specification is only required if there are cell variants.

visited A value indicating whether the item has been visited already (true or false).

multiplicity This value specifies if an item represents multiple actual instances of an item. This value can be used to compute total number of markers within a category for example. The value can be necessary if for example the cell given by the "cell" attribute has just one reference instantiation but in reality represents a large number of actual instances. By specifying the multiplicity, the item is given the appropriate weight.

image An optional image attached to the item. This string is a text representation of a image file in one of the standard formats supported by **KLayout** (preferred format is PNG) in base64 encoding.

values The list of values for this item.

6.4.2.7 Class Value (element "value")

A value is not a class for it's own, although in the code, values are represented by specific classes. In the report database, a value is simply a string representing various types of values. The general format is a type code, followed by a colon and a specific value string.

If a value represents a geometrical object, the coordinates are given in micron units and the object is located inside the associated cell and is transformed by the marker browser into the currently active cell using the reference information derived from the database or the current layout. This implies that all values with geometric interpretation must be associated with a cell.

Currently these value formats are supported:

text: <text> A message text (no geometry).

box: (<x1>,<y1>;<x2>,<y2>) A box (geometrical object).

edge: (<x1>,<y1>;<x2>,<y2>) An edge (geometrical object).

polygon: (<x>,<y>;...) A polygon (geometrical object). The points in brackets form the polygons' outline.

polygon: (<x>,<y>;.../<x>,<y>;.../...) A polygon with holes (geometrical object). The points in brackets before the slash form the polygons' outline, the point sequences after the slash form the hole contours. Each slash enters a new hole.

label: ('<text>',<trans>) A text (geometrical object). "trans" is the text transformation in **KLayout**'s transformation notation.

path: (<x>,<y>;...) w=<width> bx=<begin-ext> ex=<end-ext> r=<round-flag>
A path (geometrical object). The points in brackets form the path's center line. "ex" and "bx2" specify begin and end extension, "w" specifies the width and "r" is true, if the path has round ends.

The value string of the geometrical objects is derived from **KLayout**'s string representation which can be created within RBA with the to_s method for example.

6.5 DXF format

A brief description of how **KLayout** understands DXF input.

This is a brief description of the DXF format or more precisely: the subset of DXF that **KLayout** understands. Syntactically, DXF is a very simple format. The basic problem when reading DXF however is how to interpret it. Apparently, it is more or less a dump of the object properties of the CAD tool, and many questions regarding the interpretation of the properties are left open.

The implementation of **KLayout** is based on a number of test cases and comparison with other viewers. In some cases, the interpretation of features was varying (for example, the interpretation of array instances). In that case, TrueView (see link below) was believed to be the main authority in DXF interpretation.

Here are some links and references to other free viewers:

- DXF page on Wikipedia
- DXF specification page
- TrueView viewer for Windows
- Online viewer (www.ShareCAD.org)
- eDrawings viewer SolidWorks eDrawings 2009 supports Microsoft® XP Service Pack 2 or later and SolidWorks eDrawings 2009 for Mac supports Macintosh® Mac OS 10.4 (Tiger) or later.

6.5.1 General DXF structure

The DXF format consists of records based on a very simple structure:

Console Input 6.10: DXF Code – Simple DXF Record Structure

```
<Group-Code> <Value>
```

The group code is an integer which implicitly defines the value type and acts as a key for the value. In ASCII DXF, group codes and values are written in a single line each. These are the group codes and values relevant for layout data:

0 string (keyword)
2 string (arbitrary)
6 string
8 string
10...13 double precision coordinate
20...23 double precision coordinate

30...33 double precision coordinate
39 double precision coordinate
40...45 double precision floating point value
50 double precision angle (in degree)
62 16 bit integer value
66 16 bit integer value
70 16 bit integer value
210 double precision coordinate
220 double precision coordinate
230 double precision coordinate

Pure ASCII DXF allows the lines to contain leading or trailing blanks. **KLayout** also tries to identify stray blank lines and skip them. Some systems generate such files.

There is also a binary version of the DXF format with this basic structure:

Header (22 Bytes) "AutoCAD Binary DXF<CR><LF><SUB><NULL>"
16 bit integers 2 btyes, LSB first **Comment: misspelled "first" on Home Page**
double 8 bytes, LSB first **Comment: misspelled "first" on Home Page**
string zero-terminated

Apparently the binary format is rarely used. It suffers from a pretty basic issue: since the data type of a value - hence the byte count - is implicitly determined by the group code, exact knowledge of the data type associated with each group code is required.

6.5.2 DXF structure that KLayout understand

This is the general structure of a DXF file as **KLayout** understands it. This is a schematic representation. Lines starting with "//" are comments and not part of the file - they are provided for readability. In addition, group codes and values have been written on one line. In ASCII DXF, group codes and values are on separate lines. The indentation indicates coherent sections. Lines containing "..." indicates that other group code / value pairs may be present which are read over:

Console Input 6.11: Simple DXF Record Structure

```
// header section
0  "SECTION"
2  "HEADER"
...
0  "ENDSEC"
...
// tables section
0  "SECTION"
2  "TABLES"
...
  // layer table
  0  "TABLE"
  2  "LAYER"
  70 (# of layers - do not use for reading)
  ...
    // each layer
    0  "LAYER"
    2  (layer name)
    62 (color code)
    6  (line style)
    ...
  0  "ENDTAB"
```

```
...
0  "ENDSEC"
...
// blocks section
0  "SECTION"
2  "BLOCKS"
...
  // each block
  0  "BLOCK"
  8  (layer name - always 0?)
  2  (block name)
  70 (flags - always 64?)
  10 (base point X)
  20 (base point Y)
  ...
    // each entity
    0  (entity type)
    ... (specific for entity)
  ...
  0  "ENDBLK"
...
0  "ENDSEC"
...
// entities (top level cell)
0  "SECTION"
2  "ENTITIES"
...
  // each entity
  0  (entity type)
  ... (specific for entity)
...
0  "ENDSEC"
...
// end of file
0  "EOF"
```

KLayout reads following entities which are described in detail below:

POLYLINE	polygon, paths
LWPOLYLINE	polygon, paths
INSERT	cell ref
LINE	parts of polygon contours, path
SOLID	triangle or tetragon
HATCH	a filled area (polygon)
CIRCLE	a circle (a round-ended path with one point)
TEXT	text

6.5.2.1 POLYLINE entity

Console Input 6.12: DXF Code – DXF Record Structure – POLYLINE

```
0  "POLYLINE"
8  (layer name)
210 (extrusion direction x)
220 (extrusion direction y)
230 (extrusion direction z)
70 (flags - bit 0 for closed (1) or open (0) polyline)
40 (start width - 0?)
41 (end width - 0?)
```

```
...
  // each vertex
  0  "VERTEX"
  8  (layer name - what for?)
  10 (position X)
  20 (position Y)
...
0  "SEQEND"
```

A closed poly-line with a width of 0 usually creates a polygon (unless in “keep lines” mode, see below). A poly-line with a width > 0 creates a path. A non-closed poly-line with a width of 0 creates a path with width = 0 or contributes to the edges that will be merged in “merge lines” mode.

Individual widths are not supported – all widths must be equal or just a common width must be given.

For paths: no round ends are provided. Variable extensions have to be emulated by extending the first and last segment.

6.5.2.2 LWPOLYLINE entity for polygons

Console Input 6.13: DXF Record Structure – LWPOLYLINE

```
0  "LWPOLYLINE"
8  (layer name)
210 (extrusion direction x)
220 (extrusion direction y)
230 (extrusion direction z)
70 (flags - see POLYLINE)
43 (common width)
...
  // each vertex
  10 (position X)
  20 (position Y)
  40 (start width of segment)
  41 (end width  of segment)
...
0  "SEQEND"
```

LWPOLYLINE entities are alternative representations of POLYLINE entities and are treated alike.

6.5.2.3 SOLID entity

Console Input 6.14: DXF Record Structure – SOLID

```
0  "SOLID"
8  (layer name - what for?)
210 (extrusion direction x)
220 (extrusion direction y)
230 (extrusion direction z)
2  (block name)
10 (position1 X)
20 (position1 Y)
11 (position2 X)
21 (position2 Y)
12 (position3 X)
22 (position3 Y)
13 (position4 X)
23 (position4 Y)
...
```

To get a correctly ordered tetragon, points 3 and 4 must be swapped. A triangle is formed by setting position 3 and 4 to the same coordinates.

6.5.2.4 INSERT entity

Console Input 6.15: DXF Record Structure – INSERT

```
0  "INSERT"
8  (layer name)
2  (block name)
10 (position X)
20 (position Y)
41 (scale factor X - can be negative for mirroring)
42 (scale factor Y - can be negative for mirroring)
50 (rotation angle)
70 (number of columns - optional)
71 (number of rows - optional)
44 (column spacing - optional)
45 (row spacing - optional)
...
```

The layer specified overrides the "0" layer inside the block. This requires layer specific variants. This override is inherited by child cells as well.

The array vectors specified by number of column spacing and row spacing is rotated by the given angle, but not scaled or mirrored.

6.5.2.5 LINE entity

Console Input 6.16: DXF Record Structure – LINE

```
0  "LINE"
8  (layer name)
210 (extrusion direction x)
220 (extrusion direction y)
230 (extrusion direction z)
10 (start position X)
20 (start position Y)
11 (start position X)
21 (start position Y)
41 (scale factor X - can be -1 for mirroring)
42 (scale factor Y - can be -1 for mirroring)
39 (thickness - can be 0)
...
```

Lines are converted into paths with the specified width or contribute to the lines merged in "merge lines" mode.

6.5.2.6 CIRCLE entity

Console Input 6.17: DXF Record Structure – CIRCLE

```
0  "CIRCLE"
8  (layer name)
210 (extrusion direction x)
220 (extrusion direction y)
230 (extrusion direction z)
10 (center position X)
```

```
20 (center position Y)
40 (radius)
...
```

Circles are converted to single-point, round-ended paths.

6.5.2.7 TEXT entity

Console Input 6.18: DXF Record Structure – TEXT

```
0  "TEXT"
8  (layer name)
210 (extrusion direction x)
220 (extrusion direction y)
230 (extrusion direction z)
10 (position X)
20 (position Y)
40 (height)
50 (rotation)
1  (text string)
```

6.5.2.8 HATCH entity

Console Input 6.19: DXF Record Structure – HATCH

```
0  "HATCH"
8  (layer name)
210 (extrusion direction x)
220 (extrusion direction y)
230 (extrusion direction z)
91 (number of loops (contours))
  //  each loop:
  92 (flags, usually 3: External (bit 0) | Polyline (bit 1))
  93 (number of edges in the first loop)
    //  each point:
    10 (position X)
    20 (position Y)
    ... more points with 10/20 group codes
  ... more loops (group codes 92, 93, 10, 20 ...)
...
```

6.5.3 Other topics

6.5.3.1 Polygon formation and LINE/POLYLINE interpretation

There are several ways to form polygons from DXF input, which are controlled by the LINE/POLYLINE mode setting on the reader options page. The following modes are provided:

- Automatic
- Keep lines
- Create polygons from closed poly-lines with width = 0
- Merge lines with width = 0 into polygons
- Merge lines and auto-close open contours

"Automatic" mode will select the following modes:

- "Keep lines" if at least one SOLID or HATCH entity is present
- "Create polygons from closed poly-lines with width = 0" if at least one closed POLYLINE or LWPOLYLINE entity with width = 0 is present
- "Merge lines with width = 0 into polygons" otherwise.

The modes have the following effect:

- SOLID entities always form filled tetragons or triangles.
- HATCH entities always form complex polygons.
- Closed POLYLINE or LWPOLYLINE entities with width = 0 form polygons unless "keep lines" mode is selected. In "auto-close" mode, non-closed polylines will form a closed polygon by connecting the first and last point.
- Multiple segments specified by either LINE or POLYLINE/LWPOLYLINE entities with width = 0 are joined and, if they form a loop, create a polygon in the "merge lines" modes. In "auto-close" mode, open contours will be closed by connecting the first and last point.

6.5.3.2 Extrusion direction

The extrusion direction specified by the group codes 210, 220 and 230 is by default (0,0,1). This is the normal case. Extrusion direction (0,0,-1) is also supported. In this case, the shapes will be mirrored at the Y axis.

6.5.3.3 INSERT entities with layer specification

Layer "0" is a "wild-card" layer and can be overridden on a per-instance basis by a instance specific layer. If the instance has itself "0" layer assigned, no override takes place (or it does not have any effect).

6.6 Expression syntax

A brief description of **KLayout**'s expression syntax used, for example, to format ruler labels.

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.

6.6.1 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)	1.4142136
1+2:	$(1+2)	3.

6.6.2 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

6.6.3 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

6.6.4 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	Bit-wise NOT (evaluated as 32 bit integers)
3	–…	Numeric	Numeric	Negation
4	…^ …	Numeric	Numeric	Bit-wise XOR (evaluated as 32 bit integers)
4	…**&**…	Numeric	Numeric	Bit-wise AND (evaluated as 32 bit integers)
4	…|…	Numeric	Numeric	Bit-wise 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

Prec.	Operator	Data types	Result type	Description
8	...<...	Any	Boolean	Less
8	...>=...	Any	Boolean	Greater or equal
8	...>...	Any	Boolean	Greater
9	...&&...	Boolean	Boolean	Logical AND
9	...||...	Boolean	Boolean	Logical OR
10	...?...:...	Boolean?Any:Any	Any	Conditional evaluation

6.6.5 Functions

KLayout's expression supports 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
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 base-name 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 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 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

Function	Data types	Result type	Description
max(a,b ...)	Numeric	Numeric	Maximum of the given arguments
min(a,b ...)	Numeric	Numeric	Minimum of the given arguments
path(x)	String	String	Return 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	Find last occurrence of *t* in *s* and return 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	Split t into elements using the separator s
sprintf(f,a ...)	String, Any	String	Implement of C-like sprintf. Provides not all features, but the commonly most used ones: precision, field width, alignment, zero padding and the *e*, *g*, *f*, *d*, *x*, *u* and *s* formats
sqrt(x)	Numeric	Numeric	Square root
substr(t,f[,l])	String	String	Return a sub-string 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 first occurrence of *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 (int.)	Convert argument to numeric (32 bit integer)
to_s(x)	Any	String	Convert argument to string

Chapter 7

Useful Ruby Modules

This is a collection of hopefully useful ruby modules. These scripts may also serve as a starting point for custom developments. All scripts are installed the following way:

Windows by copying the file to the installation path of **KLayout** (the folder where `klayout.exe` is located). This is usually `C:/Program Files/KLayout`.

Unix by copying the file to an arbitrary folder and setting $KLAYOUT_PATH to it's path.

Alternatively, **KLayout** can be started with the -rm option to load the ruby module:

Console Input 7.1: **KLayout** Command Line Input — Ruby Module

```
klayout -rm script.rbm [other options]
```

Available Ruby Scripts

7.1 Compute the total area of all selected shapes

This script installs a new sub menu entry Tools > Compute total area of selected shapes. It sums up the area of all shapes selected.

Caution: This is a simple sum of areas. Areas where the shapes overlap are counted twice.

Download: calc_area.rbm

7.2 Compute the total area of all selected layers (hierarchical)

This script installs a new sub menu entry Tools > Compute layer area. It computes the total area of all layers selected.

Caution: This is a simple sum of shape areas, weighted with the cell's instant counts. Areas where the shapes overlap are counted twice.

Download: calc_area_hier.rbm

7.3 A layer processing framework

This script installs a new menu entry Tools > Processing Scripts. This menu entry asks for a processing script and executes it. Such a layer processing script contains commands to process layers such as sizing, Boolean operations and similar. The exact syntax of the scripts is described in the header of the ruby module script.

The module also maintains a list of recently used processing scripts and presents them below the menu item Tools > Processing Scripts > Processing Scripts.

Download: layer_proc.rbm

7.4 Import a Cadence techfile

This script requires at least version 0.21.13.

This script installs a new menu entry File > Import Cadence Techfile. It asks for the path of a Cadence technology file. It also requires a display resource file which it looks for in the folder where the technology file is located. If it finds multiple files with `drf` extension, it asks for a specific one. Also, if no stream layers are specified in the technology file, the script will try to find and load a layer mapping file (extension `lyp`).

The script will import the technology file and set the layer properties accordingly. These properties can then be saved using menu item File > Save Layer Properties.

Note: The script is able to parse simple forms of technology files but will not execute embedded Skill code correctly. The best way is to dump a Cadence ASCII technology file and import that file.

Download: import_tf.rbm

7.5 Import a LEF file

This script installs a new menu entry File > Open LEF, also available via shortcut Ctrl + ⇧ + L. It asks for the path of a LEF file and imports it into a new layout.

Download: LEF.rbm

7.6 A simple technology manager

This script installs a new menu Technology. It allows to summarize some configuration settings and associate them with a technology. If a technology is selected, it will switch the following configuration settings:

Grids	Current grid plus default grids.
Default layer properties	Takes the one set on the File > Setup > Layer List dialog page. This will not be the currently loaded ones, but rather switch the ones applied on the next load of a layout.
Database unit	To be used for new layouts.
Net tracer setup	Layer stack.

To set up a new technology, select the respective settings in the File > Setup > Settings dialog page, close this dialog and choose Technology > Save. A dialog pops up asking for the technology name. The given name will appear as new sub menu entry, e.g. like Technologies > My New Tech.

To remove a setup, choose Technology > Remove. A dialog pops up and provides a list of available technology settings. Select the one to remove.

To apply a setup, choose the respective entry in the Technology menu. Please note, that the effect of a setting, specifically the default layer properties, will become active on the next load of a layout, not on the currently loaded one.

Hint: The settings are stored in the file `$HOME/.klayout_tech_info.txt`.

Download: tech_manager.rbm

7.7 Search for odd-width paths

This script installs the sub menu Tools > Find Paths With Odd Width. It will find all paths with an odd width in database units in the current layout and report them. Such paths cannot be saved to OASIS, hence it's important to remove them before a layout can be written to OASIS.

Download: search_odd_width_paths.rbm

7.8 Replace cells with others from another file

The script installs a new menu entry Cells Context > Replace Cells With Others at the end of the cell list context menu. This function asks for a file containing a couple of other (top) cells, even with their own hierarchy. It will copy these cells into the existing layout and replace the corresponding cells in the current layout with the ones from the replacement library.

Hint: The script requires the database unit of the replacement and original file to be identical.

Download: replace_cells.rbm

7.9 Write all child cells of the current cell to new files

The script installs a new menu entry Cells Context > Write Child Cells at the end of the cell list context menu. This function asks for the hierarchy level and writes all cells at this level (below the current cell) to files called `<cellname>.gds`.

Download: write_childcells.rbm

7.10 Dump all shapes of the current cell recursively to a XML file

This script installs a menu entry Tools > Dump Flat Shapes. It asks for a file name and writes a flat dump of the current cell to this file. This dump contains all shapes of the cell and their chip cells projected into the top cell.

The format of the dump file is XML with that layout:

Example 7.2: XML File – Cell Shape Dump File

```
<shape_dump cell="{cell name}" dbu="{database unit}">
  <layer source="{layer}">
    .. shapes on that layer using these XML elements: ..
    <box>{box description}</box>
    <path>{path description}</path>
    <polygon>{polygon description}</polygon>
    <text>{text description}</text>
    .. more shapes ..
  </layer>
  .. more layers ..
</shape_dump>
```

Download: dump_flat_shapes.rbm

7.11 List all layers under a ruler

This script will install a new entry Tools > List Layers. Before this function can be used, a single ruler must be drawn. The script looks for shapes that are crossed by this ruler and reports the layers of those shapes. The script can operate on multiple layouts as well.

Download: list_layers.rbm

7.12 Rename all cells

This script will install a new entry Tools > Rename Cells. It will ask for a rename expression and rename all cells of the current layout. In the expression, "*" is a placeholder for the current cell name and "#" a placeholder for the cell index. Hence it is possible, for example. to add an "A" prefix by using an expression of "A*". Also it's possible to remove all traces of macro names by using "CELL#" as the expression.

Download: rename_cells.rbm

7.13 Compute the bounding box of a cell

This script will install a new entry Tools > Cell Bounding Box. It will compute and output the bounding box over all layers of the current cell (the one that is shown in the layout view and which is in the active cell tree). The output will include the corner coordinates as well as width and height.

Download: cell_bbox.rbm

Part III

Manuals

Chapter 8

Quick Start Manual – Viewer Mode

A brief recipe-type description of the functionality.

The first section describes the main window. Further sections describe simple use cases starting from scratch based on viewer mode, but likewise valid on edit mode.

Content

8.1 Basic viewing operations

8.1.1 Main window

The main window is divided into four parts by default, compare to fig. 8.1:

- The left panel host the hierarchy browser, labeled *Cells*, which depicts the cell hierarchy. Cell nodes can be expanded showing the child nodes. The cell related Cells Context menu is available with mouse right-click in the *Cells* sub window. The cell selected in the cell browser is shown in the center panel.

  Below the hierarchy browser is placed the *Navigator*. In this window the loaded layout is always shown entirely. A rectangle marks the layout part displayed in the canvas.

In case the hierarchy browser or the navigator is not visible check the View ☐ Cells or View ☐ Navigator check-box, respectively or the related check-box in the Widgets Context menu, which will appear after mouse right-click on the main menu.

- The center panel is the actual canvas. There, the layout is drawn. Click there to zoom or to draw rulers for measuring distances.

  Multiple layouts can be shown at once. Either they can be overlay-ed or they can be shown in separate views. In this case, a tab panel appears at top of the main window. Switch between the views by selecting the related tab.

- The right panel host the layer list and the layer drawing style, the *Layers* sub window. The layer related Layers Context menu is available with mouse right-click in this window. Below, a set of control panels are located in the *Layer Toolbox* sub window. The control panels are minimized per default and can be expanded by checking the check-box placed in front of the label on each header bar.

  Several control panels are available allowing to control colors, fill and drawing styles etc. Select one or many layers in the layer list to apply the selections from the control panels to.

  In case the *Layers* sub window or the *Layer Toolbox* is not visible check the View ☐ Layers or View ☐ Layer Toolbox check-box, respectively or the related check-box in the Widgets Context menu.

- The *Toolbar* is placed above the three panels, but below the main menu. In viewer mode it is composed of the three speed-bar buttons *Select*, *Move* and *Ruler*. In case the Toolbar is invisible check the View ☐ Toolbar check-box or the related check-box in the Widgets Context menu.

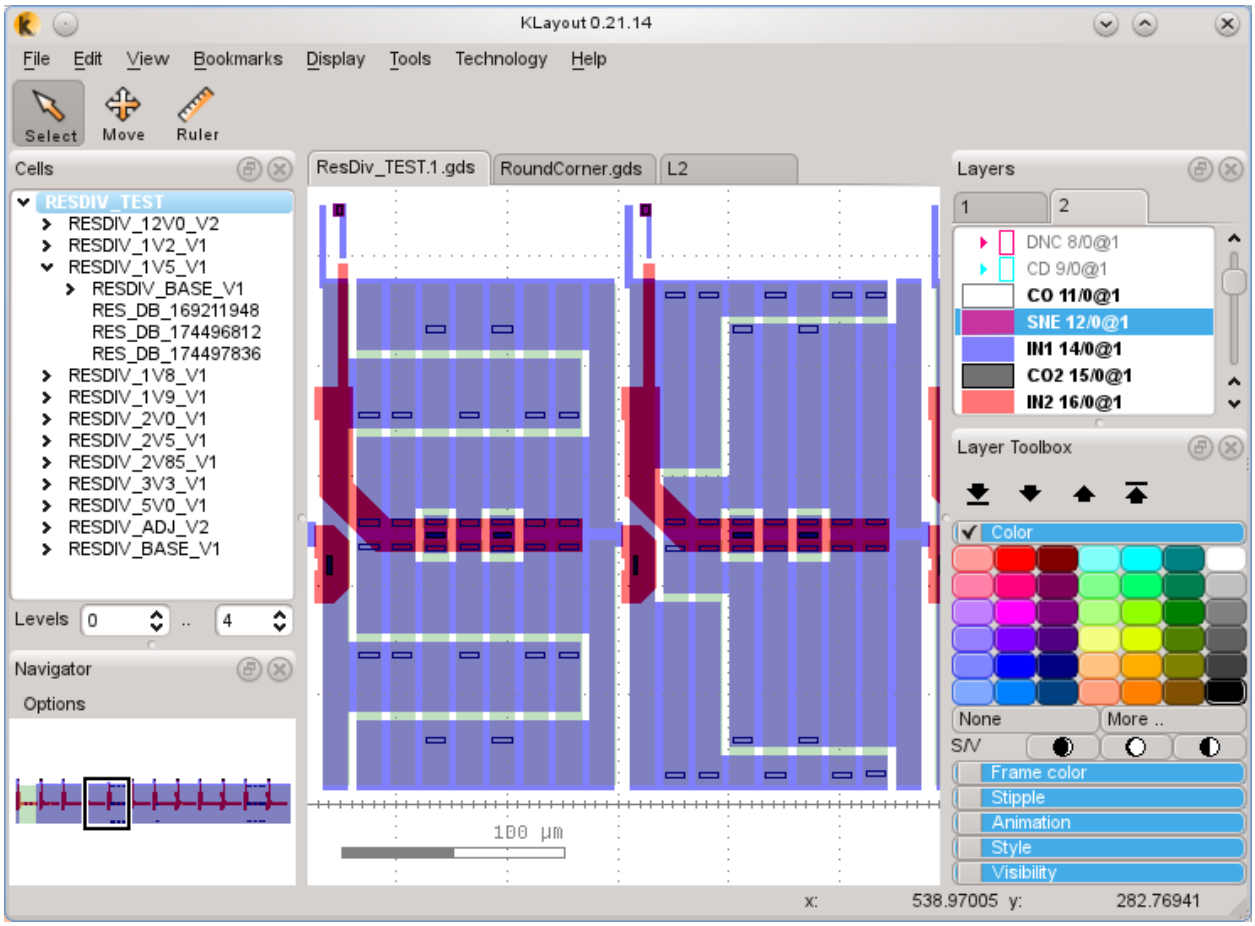

Figure 8.1. KLayout Main Window

The left and right panels width is widely adjustable by positioning the mouse over their inner vertical border. Over the border the mouse courser will change and, after right-click and hold, the sub window

follows the mouse movement. Likewise, the horizontal border between to sub windows is adjustable as well.

Each sub window can be totally exempted from the main window, or moved inside the main window to another position by mouse right-click and hold on the header bar. Now the sub window follows the mouse movements. Valid deposition areas in the Main Window will be marked by a blue rectangle if the mouse comes to there vicinity. For switching off one sub window choose the related sub menu check-box in the View menu.

A new window arrangement, differently from the default, can be stored as session File > Save Session, but is also stored on exit File > Exit or Ctrl + Q and reused at next start-up of **KLayout**.

8.1.2 Loading a file

Choose File > Open to close the current view and open a new layout instead of the currently loaded one.

Choose File > Open in Same Panel to open a new layout in addition to the currently loaded one.

Choose File > Open in New Panel to open a new layout in a new view.

Either way, a file selection dialog *Load Layout File* 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 and Gerber PCB formats are supported.

Certain options can be specified for the file loader. Choose File > Reader Options to open the *Layout Reader Options* dialog page. This dialog allows to specify certain options for all "Open" actions. Format independent options are:

Feature Subset

☐ **Enable text objects** Enable/disable reading of text objects. Disable this option objects to reduce the memory consumption if you are interested in pure geometrical information.

☐ **Enable properties** Enable/disable reading of properties. Disable this option to reduce the memory consumption if properties are not required.

Layer Subset And Layer Mapping

☐ **Read all layers** Enable/disable reading of all layers. Disable this option if only a subset of layers should be read-in or layers should be mapped to a different *layer/datatype* specification or *name* during read-in. The mapping rules may base on a layer specification set loaded from a layer properties file, see also section 8.3.4: Loading and saving the layer sets.

Format dependent options which mainly control the level of compatibility with other tools are found in the related tabs:

GDS[1] Compatibility

BOX records Defines how BOX records to be handled: Ignore, Treat as rectangles, Treat as boundaries or Treat as errors. This setting depends strongly on the stream-out settings used to generate the GDS file to be read-in.

Big records Check this option if big records (>32767 bytes) should be not allowed. **KLayout** can handle such big records, therefore, a save decision is to allow big records during read-in, but deny them at write-out for compatibility reasons, see section 8.3.15: Saving a layout or parts of it.

[1]GDSII stream format is a binary database file format.

Big polygons Check this option if big polygons with multiple XY records for *BOUNDARY* elements. As before, **KLayout** can handle such big polygons, therefore, a save decision is to allow big polygons during read-in, but deny them at write-out for compatibility reasons, see section 8.3.15: Saving a layout or parts of it.

GDS2Text[2] Compatibility

No specific options available for this format.

OASIS[3] Compatibility

No specific options available for this format.

DXF[4] Input Options

Database unit Defines **KLayout**'s database unit in micron. The default value is 0.001 micron.

DXF file unit Defines the DXF file unit in micron. The default value is 1 micron.

Arc interpolation Defines the number of points per full circle used for arc interpolation. Arc interpolation is mandatory because there are no *circle* or *arc* elements defined in the GDSII format

CIF[5] Input Options

Wire objects Defines how wire objects (path) to be handled as Square-ended paths, Flush paths or Round-ended paths.

Database unit Defines **KLayout**'s database unit in micron. The default value is 0.001 micron.

GerberPCB[6], see also section 10.4: Importing Gerber PCB files.

No specific options available for this format.

8.1.3 Managing the panels and loaded layouts

Choose File > Close to remove a layout of a panel and close the panel unless there are still layouts loaded. If multiple layouts were loaded into the current panel, a dialog appears. This allows to select one or many layouts for closing.

Choose File > Clone 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 File > Reload to reload a file if the contents have changed. This does not happen automatically.

Choose File > Pull In Other Layout to combine other layouts already loaded into the current panel. Basically, **KLayout** allows to view a layout in multiple panels, either on it's own in different configurations or together with other layouts. Pull In Other Layout function allows to configure 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.

8.1.4 Choosing a cell

To show a certain cell, select the cell in the cell hierarchy browser to the left. Then, right-click in the cell tree to bring up the cells context menu and choose Cells Context > Show As New Top or simply select the cell

[2]The binary GDSII format converted to a human readable ASCII format.

[3]Open Artwork System Interchange Standard is a binary data format. The OASIS file format is not as common as the GDSII file format.

[4]Drawing Interchange Format, or Drawing Exchange Format is a binary CAD format. The DXF Reader is just under construction and therefore should not be used for production.

[5]Caltech Intermediate Format is a recent form for the description of integrated circuits.

[6]The Gerber format is a file format used by printed circuit board (PCB) industry software to describe the images of a printed circuit board.

with the middle mouse button.

To select a cell by name, choose Display > Select Cell. A *Select Cell* dialog will appear that allows to select a cell by name or choose from an alphabetically sorted list. Additionally, this dialog allows to navigate the cell tree by choosing one of the child or parent cells.

8.1.5 Choosing a hierarchy depth

By default, only the bounding box of the cell selected is shown. This corresponds to zero hierarchy levels being shown. To select more hierarchy levels, choose one of the following methods.

Display > Full Hierarchy or press the * key to show all hierarchy levels,
Display > Box Only or press the 0 key to show only the bounding box (the default),
Display > Top Level Only or press the 1 key to show the top level elements,
Display > Increment Hierarchy or press the + key to show one more hierarchy level,
Display > Decrement Hierarchy or press the – key to show one hierarchy level less,

or use the hierarchy level entry fields below the cell list to change the current minimum or maximum level.

8.1.6 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 Cells Context > ☐ Flat Cell List option.

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 whereas macro blocks are usually large. By using cell size sorting in ascending order, the leaf cells will be shown first. To change the sorting order, check the corresponding option on the Cells Context > Sorting sub-menus.

8.1.7 Hiding cells

Independent of the hierarchy levels shown, cells can be hidden. In this case, the cell itself is not shown but its bounding box. To do so, select the cell from the cell list and choose Cells Context > Hide. To show a cell again, choose Cells Context > Show. To make all cells visible, choose Cells Context > Show All.

8.1.8 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, following functions are available by hot-keys or on the Display sub-menus:

Pan to the left, right, top or bottom using the arrow keys ←, →, ↑, ↓ or choose one of the menu items Display > Pan Left, Display > Pan Right, Display > Pan Up, Display > Pan Down. Alternatively, pan left and right by pressing Ctrl or pan up and down by pressing ⇧ while using the mouse wheel if available.

Fit the selected cell into the window by pressing F2 or choose Display > Zoom Fit.

Zoom in or out by a fixed amount by pressing Enter or ⇧+Enter or choose Display > Zoom In or Display > Zoom Out, respectively. Alternatively, 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 ⇧ while dragging the mouse with the right mouse button pressed will drag the layout around in the canvas, similar to the behavior of recent map service web applications.

8.1.9 Return to a previous view state

Choose Display » Last State to return to the last window shown or press ⇧ + ⇤⇥. Each key press walks one step back through the shown window stack.

Choose Display » Next state to switch to a more recent state again or press ⇤⇥. Each key press walks one step forward through the shown window stack.

8.1.10 Bookmarking views

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

The list of bookmarks defined can be loaded or saved by using the Bookmarks » Load Bookmarks or Bookmarks » Save Bookmarks functions.

8.1.11 Descending into a cell with context

A cell can be shown in three ways. Isolated, which is the default if the cell is the current cell, embedded as a sub-cell 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 Display » Descend 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 Display » 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 shown underlined in the cell tree, while the context cell is shown in bold font in the cell tree as usual.

The reverse operation of this is Display » Ascend or Ctrl + A.

The way how the context layout is shown can be adjusted on the File » Setup » Display » Background dialog page.

8.2 Changing the layers display style

8.2.1 Choosing a layer color

Select one or more layers for which to change the color and open the Color chooser panel in the Layer Toolbox to the right. Use ⇧ or Ctrl at mouse left-click to add or take off a layer from the selected layer list. If the Layer Toolbox is not visible check the View » ☐ Layer Toolbox check-box and in case the Color chooser is not visible in the Layer Toolbox, select the small check-box on the right side of the ☐ Color header bar. Then the Color chooser panel will be expanded.

To change the color, click on the desired color. To select a color not offered in the list, select the More ... button. A *Select Color* dialog will open.

To choose the color of the frame that is drawn 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, press ◐ or ○ button on the color panel. Pressing the button multiple times makes the colors darker or brighter each time. The darkness or brightness settings can be reset with the ◐ button.

8.2.2 Bringing layers to the front or pushing them to the back

Layers can be brought to the front so they get obscured. To do so, select the layers and push the button below the layer list. This will bring the selected layers to the end, the top of the stack, thus making them the last to be drawn. Analogous, layers can be pushed one level to front using the button or one level to back using the button. Furthermore, selected layers can be pushed to bottom of the stack thus making them the first one to be drawn by using the button.

The layer stacking order is saved with the File > Save Layer Properties function.

8.2.3 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 Layers Context > ☐ Test For Shapes in View.

If a layer is determined to be empty, it is either grayed out or it is not shown at all. The latter option keeps the layer list short and is selected with option Layers Context > ☐ Hide Empty Layers.

8.2.4 Choosing a fill pattern

To choose a fill pattern, select one or more layers for which to change the fill pattern and choose the fill pattern from the ☐ Stipple panel.

More predefined fill pattern are selectable from the *Select Stipple* dialog after pressing the More ... button.

Custom defined pattern can be created with the Custom Pattern ... button. A pattern editor will come up that allows to create new pattern. While predefined pattern cannot be changed, new created pattern will be add to the list of the *Select Stipple* dialog. To select a new pattern, select More ... from the pattern selection panel and choose the new pattern from the list. New fill patterns are saved with the layer properties File > Save Layer Properties function.

8.2.5 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. To make the fill pattern of a layer scrolling choose Scroll. For blinking mode, two phases can be selected: Blink and /Blink. Choosing different phases for two layers makes the layers appear alternatively. Choose None to reset an animation.

8.2.6 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 two pixel wide and 3px a more thicker line three pixel wide.

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.

8.2.7 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. Alternatively, choose Layers Context 〉 Hide or Layers Context 〉 Show or double left-click on the layer entry in the layer list toggles the layer's visibility as well, which is the fastest way to do so.

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, which is the default setting.

8.3 Advanced viewing operations

8.3.1 Organizing layers hierarchically

Layers can be organized hierarchically. For example, certain layers can be grouped together. Select the layers to be grouped, right-click in the *Layers* panel and choose Layers Context 〉 Group. 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 Layers Context 〉 Ungroup.

A variety of automatic grouping methods are provided. The Layers Context 〉 Regroup Layer Views 〉 By Layout Index, for example, will collect all layers and put them into one group per layout shown in the panel. Automatic grouping can be reset with the Layers Context 〉 Regroup Layer Views 〉 Flatten function.

8.3.2 Using multiple layer properties 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 Layers Context 〉 Tabs 〉 New Tab in the layer list context menu. 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, but it can be renamed with the Layers Context 〉 Tabs 〉 Rename Tab function. To remove a tab choose Layers Context 〉 Tabs 〉 Remove Tab.

8.3.3 Manipulation on layer views

The layers shown in the layer list are rather *pointers* to the actual layout layers and representing them only. Because of this, these layers are more precisely referred to as *views*. Layer views can be removed and created again without affecting the actual layout data.

8.3.3.1 Removing and adding layers to the layer set

To create a layer, choose Layers Context > Insert Layer View from the layer list context menu by right mouse button click on the layer list. Then, an input dialog *Select Source* 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 with "@1" or by omitting this specification. The source specification "10/5@2" therefore refers to layer 10 and datatype 5 of the second layout loaded in the panel.

Listing 8.1: Dialog Select Source – Layer Source Specification

```
<layer>/<datatype>{@<layout index>} // general valid
*/<datatype>{@<layout index>}       // valid in group context, see next paragraph
<layer>/*{@<layout index>}          //   for detailed description
<Layer Name>                        // valid if OASIS layer name exists
```

Source specifications can be wild-carded. That means, either layer, data type 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 as "10/*" and the parent is specified as "*/5", the effective layer looked for will be "10/5". Unlike the behavior 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 Layers Context > Clean Up Views from the context menu. Similar, layers that are present in the layout, but no view created, can be added using the Layers Context > Add Other Views method.

8.3.3.2 Transforming views

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 (not necessarily in this order):

Listing 8.2: Dialog Select Source – Transformation

```
{ ( {<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 comprehensive explanation of the transformation syntax can be found in section 6.3: Transformations in **KLayout**.

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.

8.3.3.3 Property selectors

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

Listing 8.3: Dialog Select Source – Expression

```
{ [<expr>] }
```

In GDS2 files, the property is always named with an 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.

10/5 [#43==X] is an example for a valid property selector for GDS files. With this source specification, the layer will show all shapes from layer 10 and datatype 5, which have a user property with number 43 and value string "X".

10/5 [!(#43==X&&(#2==Y||#2==U))] is a more complex example.

With OASIS files, the properties can be named as string. In this case, the property selector can be written like this **[prop==X]**, for example. In addition, the value can be an integer or a double value. This is reflected by the choice of the value and will check, if the property named "prop" has an integer value **[prop==#200]**, which is "200" in this case, or a 32 bit integer value of "0.5" in this case **[prop==##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.

8.3.3.4 Specifying explicit hierarchy levels for one layer or a 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:

Listing 8.4: Dialog Select Source – 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.

Following 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 provided 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.

8.3.4 Loading and saving the layer sets

The visual layer properties can be saved to a file with the default extension `lyp` using the function File > Save Layer Properties. The saved list can be loaded again using the File > Load Layer Properties function.

8.3.5 Creating a screen-shot

To save the canvas as a PNG file, choose File > Screenshot or press the Print key. A file dialog box will appear in which the file can be specified where the screen-shot is saved to.

8.3.6 Doing measurements

A measurement can be performed by choosing Ruler mode in the toolbar and left-clicking a point in the layout followed by left-clicking at another point. A ruler will be shown that indicates the distance measured.

Various options can be specified for the rulers. Choose the setup dialog File > Setup > Ruler And Annotations menu and select one of the sub entries or choose Edit > Ruler And Marker Setup which shows all available options on one page.

On the *Rulers And Annotation* dialog, various options can be selected. A ruler can be made to snap to edges of objects by checking ☐ Snap to edge / vertex or the ruler orientations can be constrained by using the Angle constraint options, by example.

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

The number of rulers displayed in the canvas can be limited by entering a number in the field of menu Appearance > Limit number of annotations to <number>. If the number of rulers specified is two, for example, only the last two rulers are shown.

All rulers can be cleared using the Edit > Clear All Rulers And Marker function or by pressing Ctrl+K.

Ruler dragging can be canceled with the Esc key or by using the Edit > Cancel function.

Rulers can be moved by selecting Move mode with the speed-bar buttons in the toolbar or by choosing the Edit > Mode > Move sub-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 Del.

Rulers can be modified in a variety of ways. For example, rulers can be shown as arrows or as box. To edit the properties of a ruler, double-click the ruler or select it and use Edit > Properties or press Q. See section 8.3.7: Ruler properties for a detailed description.

Multiple templates can be configured to be available for rulers. If multiple templates are available, the Ruler toolbar button will show a drop-down menu which allows to select one template to be used. Templates can be edited in the ruler setup page available with File > Setup > Rulers And Annotations > Templates or with Edit > Ruler And Marker Setup.

8.3.7 Ruler properties

These are the properties that can be configured for rulers:

Labels Depending on the outline of the ruler, up to three labels can be present. Each label can be configured 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").

Style The style determines how the ruler or it's components are drawn. This can be "ruler-like" (with ticks), arrow style or a plain line.

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.

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, but can 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, however, be disabled for each individual ruler.

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 support are the basic operations (*, /, +, –, ..), bit-wise operations (|, **&**, ..), the conditional operator (**x:y?z**), as well as some functions, like i.e. **abs**, **sqrt**, **exp** and includes a **sprintf** function. Here 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 section 6.6: Expression syntax.

Following 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.

8.3.8 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, and others). Color and monochrome images are supported. Internally an image is stored as a matrix of float values. It is possible to write custom importers using RBA.

To add an image, use the Edit 〉 Add Image function. An *Image Properties* 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.

Offset This is the point where the lower left corner of the image is placed (in micron units).

Rotation An arbitrary angle by which the image is rotated.

☐ **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.

☐ **Preview (Auto apply)** If this option is checked, the image settings are applied immediately.

Reset The Color mapping, Brightness, Contrast and Gamma, and RGB channel gains settings can be reset to the default values with this button.

Once an image is placed, it can be moved and re-sized using the Move function from the speed-bar. The images properties can be adjusted using the Edit 〉 Properties function or by double-click the image.

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.

8.3.9 Browsing shapes

A simple shape browser allows to browse all shapes on a layer. To do so, select the layer to browse in the layer list and choose Tools 〉 Browse shapes.

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 *Shape Browser Setup* dialog

which is available via the button Configure of the *Browse Shapes* dialog or on the respective page in the File > Setup > Browsers > Shape Browser dialog page.

8.3.10 Browsing instances

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

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 *Instance Browser Setup* dialog which is available via the button Configure of the *Browse Instances* dialog or on the respective page in the File > Setup > Browsers > Cell Instance Browser dialog page.

8.3.11 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's:

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, as example.

Image A marker can be assigned to a screen-shot 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 organization of such categories.

Marker items are usually associated with a cell, i.e. where an 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 Tools > Marker Browser function. The marker browser tracks whether a marker has already been visited similar to the "read" flag in a mail client. This allows to track 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.

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 on the File > Setup > Marker Database Browser > 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, assumed 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 screen-shot 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.

8.3.12 Selecting rulers, shapes or instances

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

Pressing the ⇧ key in addition to selecting shapes or instances will extend the current selection. Pressing Ctrl key will remove all selected shapes or instances from the selection.

Only such cells will be selected as instances, of which the bounding box is shown. With the check boxes of the Edit » Select sub menu, the kind of shapes that participate in the selection can be changed. In addition, selection of instances or rulers can be enabled or disabled.

The properties of the selected objects can be browsed with the Edit » Properties function. A dialog appears that shows the properties of the first object selected. In case of a rectangle, for example, these are the coordinates of the corners. Additionally, the instantiation path of the object can be shown by pressing the Instantiation button. The dialog that shows up then will state the cell that contains the object (this is the lowest cell) and the cells in which these cell is instantiated up to the top cell. Similar, User properties shows a list of properties attached to this object.

8.3.13 More configuration options

The option dialog available with the File » Setup function offers numerous configuration options from background color 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 down-sampled 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 Edit » Setup » Display » General dialog page. Two times and three times oversampling is provided. The following screen-shots illustrate the effect of oversampling:

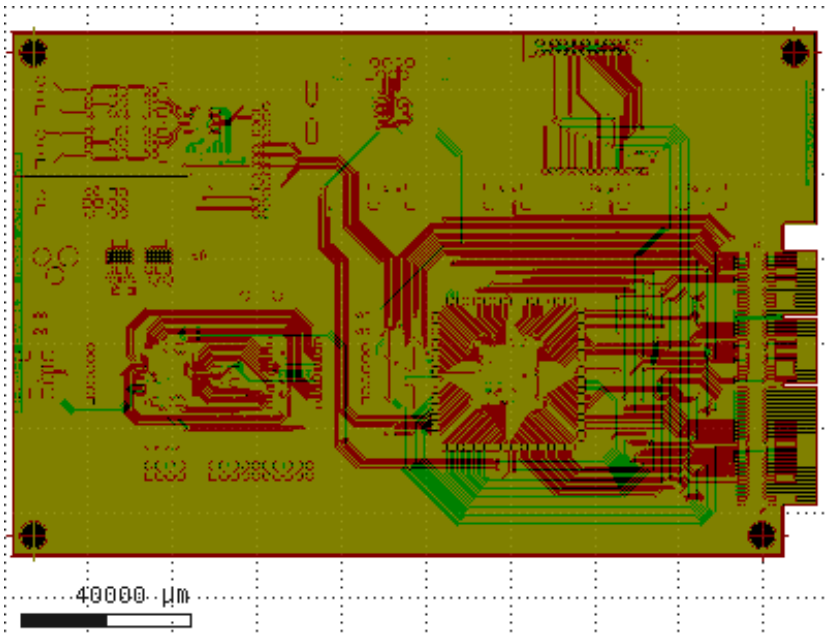

Figure 8.2. Display without Oversampling (1x, Normal)

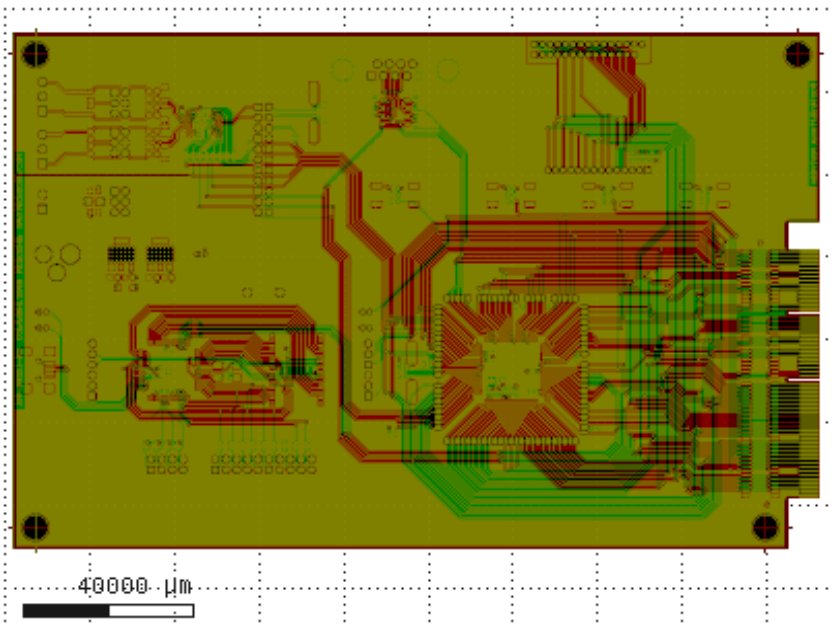

Figure 8.3. Display with 2x Oversampling

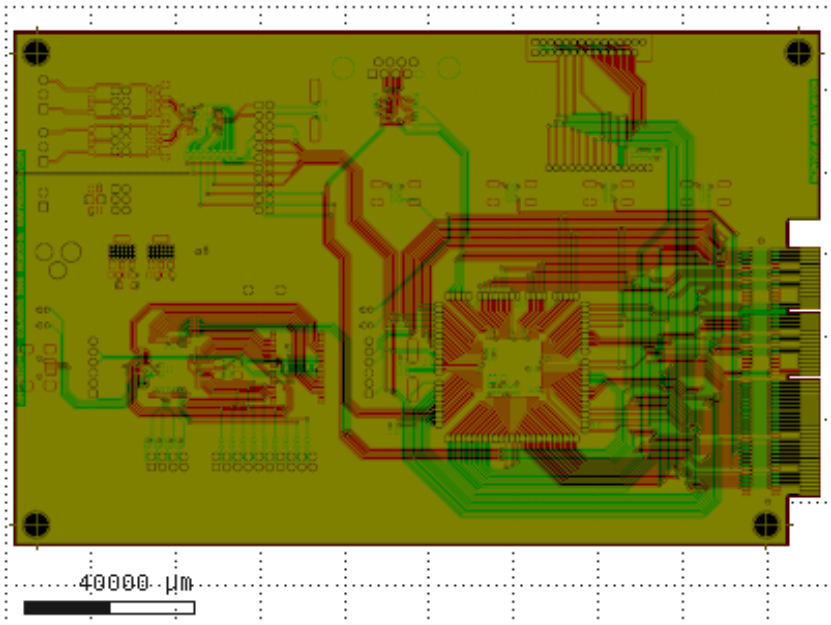

Figure 8.4. Display with 3x Oversampling

8.3.14 Undo and redo

Most operations such as changing of layer colors can be undone using the Edit > Undo function or keys Ctrl + Z. Analogous, the operations can be redone again using the Edit > Redo function or keys Ctrl + Y.

8.3.15 Saving a layout or parts of it

A layout or a sub-cell of it can be saved to several formats. In general, to save a layout, choose File > Save As function. To save just a cell, select the cell in the cell tree (it does not need to be the currently shown one) and select Cells Context > Save Current Cell As by mouse right-click on the cell tree.

A *Save Layout File* dialog will pop up to enter or select the file name to which to write the cell or layout. After a file name and file extension has been selected, a format dependent *Layout Writer Option* dialog will be shown to specify format dependent 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.

Format independent options are as follows:

Layout Writer Option

- **Format** — Select the output format, preset according to file extension.
- **gzip** — Select compression with GNU zip.

Generic Options

- **Layers to save** — Select one option which layers to save:
  - **All layers** — even empty layers, or
  - **Layers shown in list** — a sub-set of layer, or
  - **Visible layers only** — another sub-set of layer.
- **Database unit** — Enter a database unit in micron, preset is current database unit.
- **Scaling factor** — Enter a scaling factor, preset is 1.0.
- **Write non-empty cells only** — Select this check-box to skip empty cells.

Format dependent options for GDS2 stream or GDS2 text stream format are:

GDS2 Writer Options

Library name Enter the GDS library name, preset is "LIB".

Max. cell name length Enter the maximum allowed cell name length, preset is 32000.

Max. vertices Enter the maximum allowed number of vertices per object. A number less than 4000 is recommended, while 8191 is the absolute limit. Preset is **Comment: ????** or the last entry.

Multi-XY record mode for boundaries Select this check-box to enable infinitely large polygons at the cost of compatibility.

Write current time to time stamps Checked by default. The current time is written to the file to simplify comparison of binary stream files for example.

for OASIS stream format:

OASIS Writer Options

Compression level Select a compression level.

- **Level 0** No particular attempt is made to compress shapes.
- **Level > 0** Shapes are classified and array compression is tried.
- **Level >> 0** The higher the level, the more tests are made to compress shapes into arrays. In particular for flat layouts, compression of shapes requires some memory and slows down OASIS writing considerably.

for DXF[7] stream format:

DXF Writer Options

Polygon handling Select how to handle polygons:

- **Write POLYLINE entity** use the original entity type.
- **Write LWPOLYLINE entity** use this entity type.
- **Decompose into SOLID entities** use this entity type – a 2D solid entity with three or four sides (triangle or tetragon).
- **Write HATCH entity** use this entity type – a filled area.

and for CIF stream format:

CIF Writer Options

none

8.3.16 Saving and restoring a session

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

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

[7] The DXF Writer is just under construction and therefore should not be used for production.

Chapter 9

Quick Start Manual – Editor Mode

This user manual is a brief description of how to use **KLayout** in editor mode. **KLayout** can be put into editing mode by simply supplying the "-e" option on the command line:

Listing 9.1: **KLayout** Command Line Input — Layer Property File

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

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

KLayout can be configured to use editing mode as default when started. To enable editing by default, check the File » Setup » Application » Edit mode » ☐ Use editing mode by default check box.

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.

This document covers the functionality in a basic section and as brief recipe-type descriptions of the main editing functions.

Basic Editor Mode, Basic and Advanced Editing Operation

9.1 Basic principles of editor mode

9.1.1 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, ⇧ 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 ⇧ key forces **KLayout** into orthogonal mode: movements are restricted to horizontal or vertical unless not applicable. While Ctrl key forces **KLayout** into diagonal mode: movements are restricted to horizontal, vertical or the diagonal axes. Pressing Ctrl + ⇧ will release all directional constraints - movements will be allowed in any direction.

9.1.2 Basic editor mode 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 Edit 〉 Editor Options 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:

Snapping

- **Grid** — Every editing operation is confined to that grid. It can be either
  - **No grid** — disabled,
  - **Global grid** — aligned with the global grid (used i.e. for rulers and display) or
  - **Other grid** — specified explicitly. It can even be anisotropic, i.e. there can be a different grid in y than in x direction.
- **Objects** — Snap to other objects can be either
  - ☑ — Snap to grid and to other objects.
  - ☐ — Snap to grid only.

Angle Constraints

- **Connections** — 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. It can be either
  - **Any Angle** — there is no such confinement.
  - **Diagonal** — the edge or segment can be vertical, horizontal or in one of the two diagonal directions.
  - **Manhattan** — only horizontal and vertical edges or segments are allowed.
- **Movement** — When something is dragged (i.e. moved), this mode determines if the movement is confined to certain directions. It can be either
  - **Any Direction** — unconfined, or
  - **Diagonal** — restricted to orthogonal and diagonal directions, or

Manhattan restricted to orthogonal directions.

Selection Mode

Hierarchy Select top level objects only can be either

☑ Top level selection mode: only elements on the level of the currently shown cell are individually selectable, where top level refers to the top level of the currently shown cell here. That means, If shapes from a sub-cell are selected, the whole instance of this sub-cell is selected.

☐ Hierarchical selection mode: elements are selected from sub-cells as well. This mode allows to in-place edit sub-cells which is a powerful feature but also can create strange side effects because all other instances of this cell placed anywhere changes as well.

Instance Display

☑ Show shapes when moving (max. <number> shapes), with 1000 shapes as default.

☐ Don't show shapes when moving.

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

9.1.3 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 ⇧ button in addition to selecting elements), by removing elements (press Ctrl in addition) or by toggling the selecting (press ⇧+Ctrl in addition: remove already selected ones and add new ones).

9.1.4 Partial editing vs. full element editing

Partial editing is a powerful feature that allows to modify shapes. It allows to move edges or segments of polygons resp. paths, to delete vertices, edges or segments from polygons or paths and to insert new points 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 in *full element* mode. By this, the whole shape is being moved or deleted. Only in *full element* mode, shapes or instances can be sent to the clipboard.

9.2 Basic editing operations

9.2.1 Creating a layout from scratch

To start with a fresh, empty layout, choose File 〉 New function. 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 and 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 on the File > Setup > Application > Layer List dialog page, 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 Edit > Layer > New Layer, before any shapes can be drawn.

9.2.2 Creating a new layer

You can create a new layer using the Edit > Layer > New Layer function. You are prompted to enter GDS layer and data-type number and optionally an OASIS layer name. Clicking OK, the layer will be created and inserted into the layer panel.

9.2.3 Creating a new cell

You can create a new cell using the Cells context > New Cell function by right mouse click on the cell list. You are prompted to enter the new cell's name, whereby a cell with that name must not exists yet, and to enter a window size to that the canvas will be set.

9.2.4 Creating a polygon

Select Polygon mode from the speed-bar and choose a layer from the layers panel in which to create the 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 mouse left-click. Move to the next vertex. Depending on the connection mode, the edges created are confined to certain directions. See connection angle constraints description in section 9.1.2: Basic editor mode options for a detailed description of the modes. Use the Edit > Object Editor Options dialog (shortcut F3) to change the mode, even during editing.

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.

9.2.5 Creating a box

Select Box mode from the speed-bar. 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.

9.2.6 Creating a path

Select Path mode from the speed-bar. The *Object Editor Options* dialog, *Path* tab will open that additionally prompts for basic path parameters, such as width and extension scheme. When a path is being drawn, it will receive the settings entered into this dialog. The path properties can even be changed, while the path is being drawn. Don't forget to click on Apply to take over the current entries. If the dialog has been closed unintentionally, it can be reopened with the F3 shortcut.

To actually draw a path, choose a layer from the layer panel in which to create the new path. Left-click at the first vertex, move the mouse to the second vertex, click to place this one and continue to the last vertex. Double left-click at the last vertex to finish the path. Press the Esc key to cancel the operation.

For paths, as for polygons, the segments created are subject to certain direction restrictions as imposed by the connection angle constraints. See connection angle constraints description in section 9.1.2: Basic editor mode options for a detailed description of the modes. Use the File » Objects Editor Options dialog page (shortcut F3) to change the mode, even during editing.

9.2.7 Creating a text object

Select Text mode from the speed-bar. The *Object Editor Options* dialog, *Text* tab will open that additionally prompts for the text string. Don't forget to click on 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 and text scaling is enabled, whereby the "Default" font is not scalable. In order to do so, choose a scalable font from the File » Setup » Display » Texts dialog. Check ☐ Show texts or properties check-box and check the ☐ Apply text scaling and rotation check-box on the same page.

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 each click.

9.2.8 Placing an instance of a cell

Select Instance mode from the speed-bar. The *Object Editor Options* dialog, *Instance* tab 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 each click. Press the Esc key to cancel the operation.

9.2.9 Moving the selection

The whole selection can be moved in Move mode. If some elements are already selected, choose Move mode from speed-bar 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. If 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 mouse right-click. The Esc key will cancel the operation.

For movements, the movement direction constraint apply. See movement direction constraint description in section 9.1.2: Basic editor mode 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 ⇧ key for orthogonal, Ctrl for orthogonal and diagonal or both keys ⇧+Ctrl for any angle direction constraints while moving the mouse.

9.2.10 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 Edit > Selection sub-menu. For example, Edit > Selection > Flip Vertically flips the selection at the x-axis. A selection can be rotated by an arbitrary angle using the Edit > Selection > Rotation By Angle function.

9.2.11 Partial editing

When objects have to be modified after they 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 to delete 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 movement direction constraint description in section 9.1.2: Basic editor mode 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 ⇧ key for orthogonal, Ctrl for orthogonal and diagonal or both keys ⇧+Ctrl for any angle direction constraints while moving the mouse.

To enter partial mode, click on the Partial button in the speed-bar. 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 keys ⇧ 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 description of top level selection mode in section 9.1.2: Basic editor mode options.

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 key 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 Edit > Delete function or pressing the Del. 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 center-line, a single click on the newly formed segment and a single click to drop it at the new position.

9.2.12 Moving the selection to a different layer

Selected shapes can be moved to a different layer as a whole. For this, choose Edit > Selection > Change Layer function. All selected shapes are moved to the layer that is the current one in the layer list (marked with a rectangle or blue underlay-ed). 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, see section 9.2.14: Copy and paste of the selection.

9.2.13 Other layer operations

The layer specification can be edited using the Edit > Layer > Edit Layer Specification method. 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 cell-wise, on a cell's hierarchy or for all cells using the Edit > Layer > Clear Layer method.

9.2.14 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 in *hierarchical selection* mode. This mode is enabled if Edit > Editor Options > Selection Mode > Hierarchy > ☐ Select top level objects only is deselected, now select the shapes to flatten and copy everything to a different cell.

In *non-hierarchical selection* mode, this mode is enabled if Edit > Editor Options > Selection Mode > Hierarchy > ☐ Select top level objects only is checked 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 by pressing Ctrl + C, two possible methods are offered by the *Copy Options* dialog:

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.

9.2.15 Delete a cell

To delete a whole cell, select the cell in the hierarchy browser and choose Cells Context > Delete Cell by mouse right-click. This time, three possible modes are offered by the *Delete Cell Options* dialog:

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 sub-cells are deleted, unless the sub-cells are referenced otherwise (by cells that are not deleted). In this delete mode a complete hierarchy of cells can be removed without any side effects.

Complete delete The cell and all it's sub-cells are deleted, even if other cells would reference these sub-cells.

9.2.16 Rename a cell

To rename a cell, select the cell in the hierarchy browser and choose Cells Context > Rename Cell by mouse right-click. The *Rename Cell* dialog prompts for a new name which must not exist yet.

9.2.17 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 Edit » Copy, shortcut Ctrl + C, or Edit » Cut, shortcut Ctrl + X. To paste such a cell into a target layout, choose Edit » Paste, shortcut Ctrl + V.

Copying a cell 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.

9.3 Advanced editing operations

9.3.1 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 » Selection 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 ☐ Select top level objects only 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, see section 9.2.14: Copy and paste of the selection.

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 is lost for all other instances of the cell. This may create undesired side effects and it is likely that this behavior will change in future implementations.**

Make Cell Removes the currently selected objects and places them into a new cell whose name can be specified in the *Make Cell* dialog.

9.3.2 Creating clips

KLayout provides a utility to create rectangular clips from a given cell Edit » Utilities » Clip Tool. 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.**

9.3.3 Flatten cells

The Edit > Cell > Flatten Cell operation flattens a cell into all of it's parents. This basically removes a cell by promoting its shapes and instances up in the hierarchy.

The flatten operation offers three options on the *Flatten Instances* dialog, how deep to go through the hierarchy levels to flatten and one option how to deal with child cells which become obsolete through this operation. By enabling this ☐ Prune option, all child cells are removed when they are no longer needed. Otherwise, new top level cells will appear - these are the cells which are not longer instantiated.

9.3.4 Layer Boolean operations

KLayout now comes with a set of Boolean operations. The Boolean operations are available in the Edit > Layers > Boolean Operation menu functions). A *Boolean Operation Setup* dialog will open that allows to specify input layers, mode, output layer and certain other options.

Union (OR) The output layer will contain all areas which are covered by shapes from layer A and layer B.

Intersection (AND) The output layer will contain all areas where shapes from layer A and layer B overlap.

Difference (A NOT B) The output layer will contain all areas where shapes from layer A are not overlapping with shapes from layer B.

Difference (B NOT A) The output layer will contain all areas where shapes from layer B are not overlapping with shapes from layer A.

Symmetric difference (XOR) 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 special Boolean operation is provided, the *merge* operation Edit > Layers > Merge. A *Merge Operation Setup* dialog will open that allows to specify input layer, overlap threshold, output layer and certain other options. This function is a single-layer operation that joins (merges) all shapes on the layer. As a special feature, this operation allows to select 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 because self-overlaps of polygons are not detected in this mode.

All Boolean operations can be performed in three hierarchical modes:

Flat Both layers in Boolean operation or the layer in merge operation are flattened and the results are put into the current top cell.

Top cell only Perform the operation on shapes in the top cell only.

Individually for current and sub cells Perform the operation on shapes of all cells below the current top cell individually. This mode is allowed only if the layout of input layer(s) and output layer 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 to combine layers of different layouts, i.e. compare them using a XOR function.

As a special feature, **KLayout**'s Boolean implementation allows to choose how *kissing corner* situations are resolved. **KLayout** allows two modes:

☐ **Minimum coherence** Checked: 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.
Unchecked: The output will contain as much, potentially touching polygons as possible.

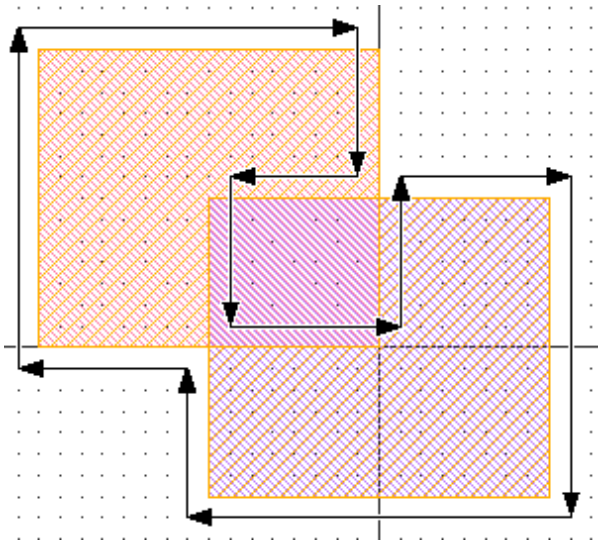

Figure 9.1. Illustration of maximum coherence

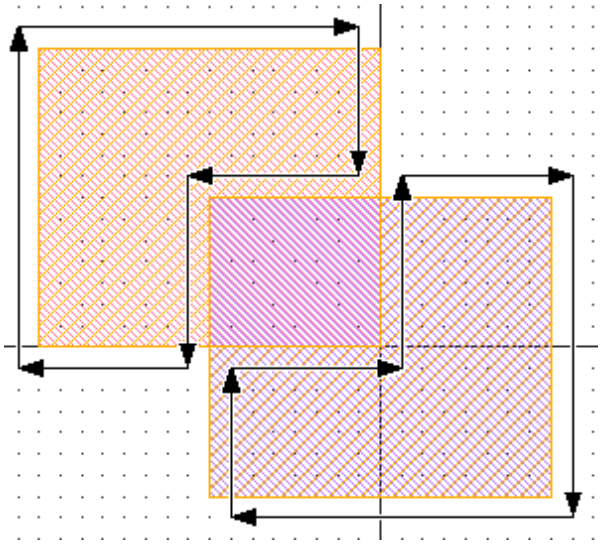

Figure 9.2. Illustration of minimum coherence

The screen-shots fig. 9.1 and fig. 9.2 illustrate the maximum and the minimum coherence modes for a XOR operation between two rectangles.

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 scan line implementation. Performance penalty is about two times slower compared to an efficiently implemented single-pass algorithm.

9.3.5 Layer sizing

A sizing operation allows to grow or shrink the shapes of a layer by a given offset, which is applied per edge. Choose the sizing function by left-click on the Edit > Layer > Size menu. A *Sizing Operation Setup* dialog will open that allows to specify input layer, sizing value, cutoff mode, output layer and certain other options.

The sizing value must be given in micron, where positive values will enlarge the shapes while negative values will shrink the shapes. A single value stands for same sizing in x and y direction while a comma-separated list of two values stands for different sizing in the two directions (i.e. "0.2,0.1"). 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 cutoff strategy for sharp edges can be chosen from strict to virtually unlimited. The screen-shot fig. 9.3 demonstrates the effect for Strict (diagonal) (red curve) to Weak (sharps bends >135 deg.) (purple curve) cutoff modes.

As for the Boolean operations, hierarchical mode and *kissing corner* resolution can be specified, see section 9.3.4: Layer Boolean operations for a description of these modes.

9.3.6 Shape-wise 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 ⇧ modifier key.

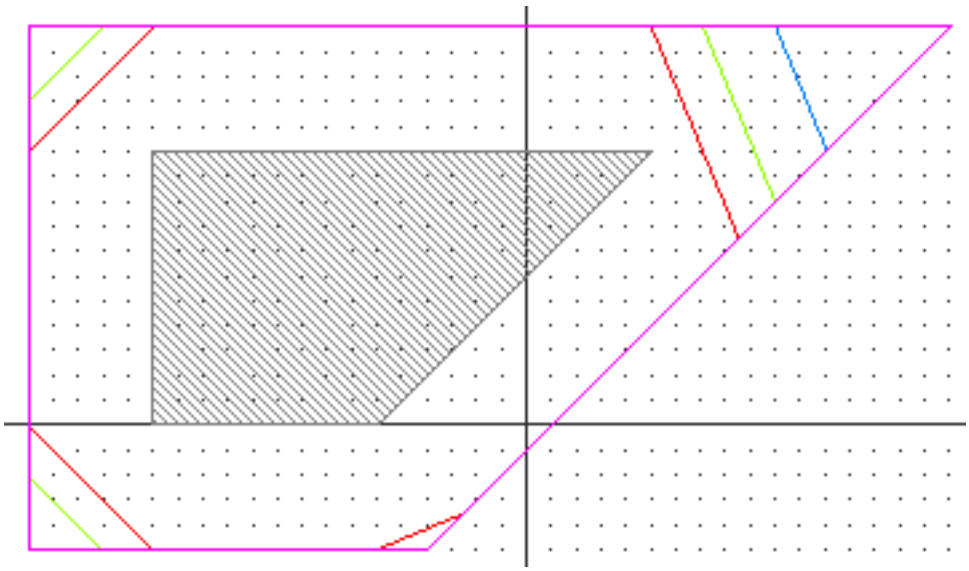

Figure 9.3. Illustration of "strict" (red curve) to "weak" (purple curve) cutoff modes

The following operations are available in the Edit > Selection sub-menu:

Merge Shapes Merge all shapes in the primary and secondary selection and write the results to the layer of the primary selection.

Intersection – Other With First Compute the intersection (AND) of primary and secondary selection and write the results to the layer of the primary selection.

Subtraction – Others From First Compute the difference (A NOT B) of primary (A) and secondary (B) selection and write the results to the layer of the primary selection.

9.3.7 Shape-wise 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 is given in the Edit > Selection > Size Shapes menu. A *Sizing* dialog will open that prompts for the sizing value in micron, where one value stands for same sizing in x and y direction, while two comma-separated values stands for different sizing in x and y direction. In contrast to section 9.3.5: Layer sizing, the sign of both values must not be identical, but joining to points or negative values, as result of this operation, are not supported and generates invalid shapes.

9.3.8 Object alignment

Object alignment is available on selected object sets. This operation use the concept of *primary* and *secondary* selection. The primary selection contains all objects that are selected in the first step. The secondary selection contains all objects that are selected in additional steps using the ⇧ modifier key.

The object alignment function allows to align 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). A valid object can be a shape or an instance of a cell.

Choose the alignment function by left-click on the Edit > Selection > Align menu after selection of a set of objects. An *Alignment Options* dialog will open which allows to specify the alignment mode and bounding box computation mode for cell instances. The dialog offers following settings:

Horizontal alignment

- ⊙ **none** no changes, or
- ○ **left** align left sides, or

○ **center** align centers, or
○ **right** align right sides.

Vertical alignment

⊙ **none** no changes, or
○ **top** align top sides, or
○ **center** align centers, or
○ **bottom** align bottom sides.

Layers for alignment of instances

⊙ **Use all layers** for cell instance bounding box to referee to, or
○ **Use visible layers only** for cell instance bounding box to referee to.

9.3.9 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 is otherwise modified, the original geometry can be reconstructed and the corner radius can be changed again. 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 screen-shot fig. 9.4 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 chosen by mouse left-click on Edit ⟩ Selection ⟩ Round Corners menu. A *Dialog* will open which allows to specify the outer corner radius, the inner corner radius, both in micron, as well as desired number of points (for full circle). 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.

9.3.10 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 to shift 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.

The cell origin adjustment function can be chosen by left-click on the Edit ⟩ Cell ⟩ Adjust Origin menu.

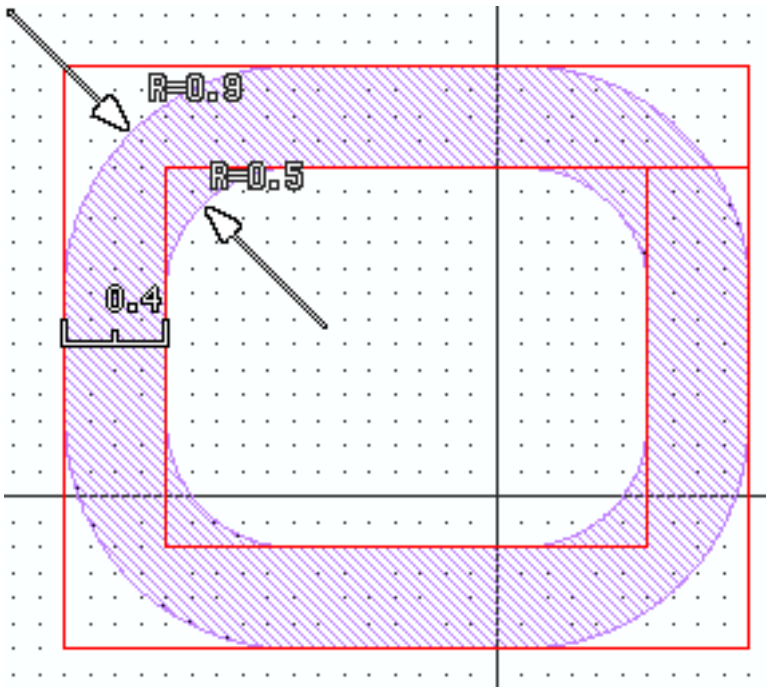

Figure 9.4. Illustration of round corners function

9.3.11 Layer operations: clear, delete, edit specification

Three full-layer operations are implemented and available in the Edit > Layers sub-menu:

Clear	Clear but don't delete the currently active layer in the layer list.
Delete	Clear and delete the currently active layer in the layer list.
Edit Layer Specification	Edit the layer specification of the currently active layer in the layer list.

The layer specification describes how a layer is saved to GDS or OASIS streams and, if chosen, a *(New) Layer* will open which allows to specify or change the **Layer Properties**. It consists of a layer and data type number and optionally a layer name for OASIS streams. Only layers with valid layer and data type specification are written to GDS or OASIS files.

Chapter 10

Advanced Functions

This chapter briefly describes a couple of **KLayout**'s advanced features.

Content

10.1 The XOR tool

The XOR tool performs a geometrical XOR (also A NOT B and B NOT A for asymmetric differences) on two layouts by performing the respective Boolean operations layer by layer. The XOR tool is started using Tools > Verification > XOR Tool 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 to specify tolerances. Basically a tolerance is an undersized 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 1 nm, 5 nm and 10 nm.

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 to send 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.

10.2 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.
- 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 are not compared.
- The diff tool can also work in "XOR" mode. 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 super-set of the true XOR's results. It may report some locations as being different which are not in fact, 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 > Verification > Diff Tool menu. A *Diff Tool* dialog will open that allows to specify the two layouts to be processed and certain other option.

Input

Layout A	select the first layout and
Layout B	select the second layout to process.

Options

☐ **Run XOR on differences**	check to select the "XOR" mode, which disables the following options only available in pure "Diff" mode.
☑ **Summarize missing layers**	check to have missing layers reported as one difference instead of one per shape.
☑ **Detailed information**	check to receive detailed information about every difference. Without that option, only the number of differing shapes or instances are reported.
☐ **Expand cell arrays**	check to compare individual instances of array instances.
☐ **Exact compare**	check 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.

10.3 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 can be found in the Edit > Utilities > Fill Tool menu and is available in edit mode only. A *Clip Tool* dialog **Comment: Wrong dialog name** will open and offers the following settings:

- In section *Fill Area* select the outer boundary of the fill region ("what to fill"). Available choices are:

  All (whole cell),

  Shapes on layer ... the interior or the polygons on a given layer, <layer > select the layer,

  Selected shapes the interior of all selected polygons,

  Single box with ... a single box and select the box boundaries or

  Ruler bounding boxes an area defined by a ruler.

- Specify a border distance if the fill area should keep a certain minimum distance to the border of the fill region in the Keep distance to border of fill area of <distance> µm entry field.
- In section *Exclude Area* specify the regions within the fill region which must not be filled. Available choices are:

  All layers which doesn't create fill over any polygon drawn,

  All visible layers which doesn't create fill over any polygon visible,

  Selected layers or

  No exclude don't exclude anything.

- Specify a spacing distance if the fill tiles must keep a certain minimum distance from the exclude regions in the Spacing around exclude areas <distance> µm entry field.
- In section *Fill cell* specify the fill cell.
  - Enter a cell name in the Fill cell entry field or chose one using the dialog *Select Cell* by left-click on ... button.
  - Specify the Fill cell margin <distance> µm.
  - Choose the Boundary layer <layer > which defines the cell's footprint and controls tiling raster of the cells.
- in section *Options* specify ☐ Enhanced fill option which allows the fill tool to leave fixed raster for enhanced fill of small regions.
  - By default, unchecked, the fill utility operates on a fixed raster. This can lead to a poor fill efficiency in some cases.
  - Checked, the fill utility tries to find a cell arrangement which is not necessarily on a common raster but provides a better fill performance.
- Check ☐ Second-order fill cell for remaining regions option and a second – usually smaller – fill cell can be specified, which is used to fill the remaining areas of the layout. The boundary layer must be the same for the second order fill cell.
  - Enter a second cell name in the Fill cell entry field or chose one using the dialog *Select Cell* by left-click on ... button.
  - specify a Fill cell margin <distance> µm.

The screen-shots figs. 10.1 to 10.3 show the effect of the different fill modes for some artificial fill problem.

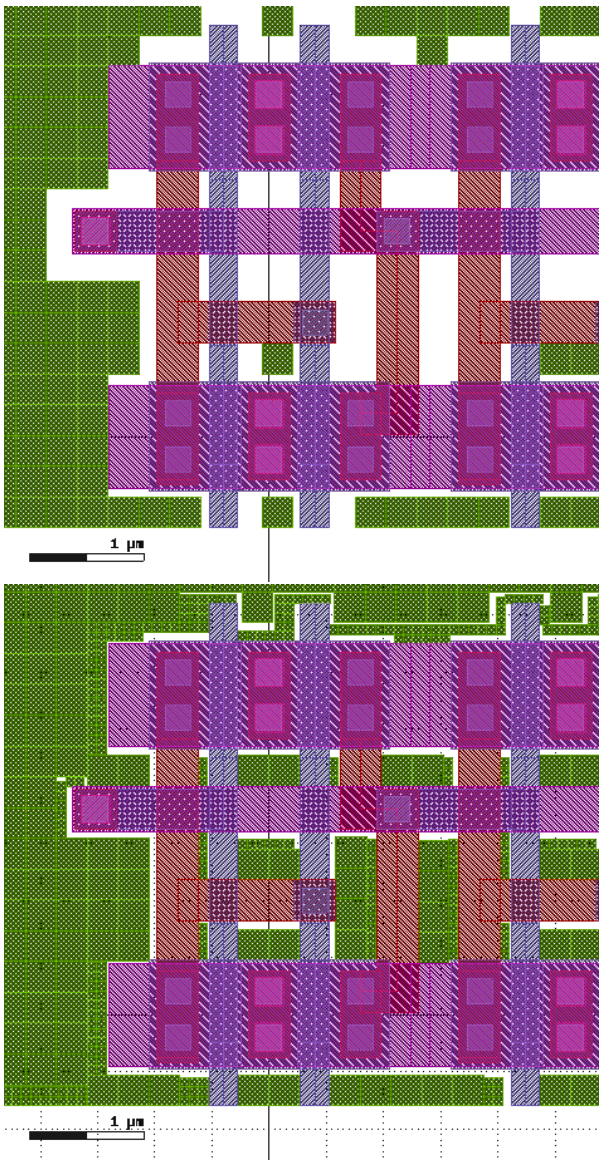

Figure 10.1. Illustration of Default Fill Option (Top Left)

Figure 10.2. Illustration of Enhanced Fill Option (Top Right)

Figure 10.3. Illustration of Second Order Fill Option (Bottom Left)

10.4 Importing Gerber PCB files

Gerber PCB import allows to create 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 in XML format which contains the importer specifications. The suggested suffix for this file is `.pcb`. 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 functions are available as sub menus of the menu File > Import > Gerber PCB. Different entries are given that start a new project New Project or a new project that allows to specify arbitrary mapping between PCB files and layout layers (free layer mapping) New Project – Free Layer Mapping, open an existing project Open Project or continue with the last project Last Project.

The work-flow to import PCB data is as follows and meant as an overview, whereby the basic decision is how to specify the layer mapping. Each dialog is explained in detail in the subsections later on.

- On dialog page *General*, section *Base Directory*, specify the directory where the PCB data files are located (the "base" directory) using the entry field or choose one by left-click on [...] button which offers a file browser dialog *Get Base Directory*.
- In section *Import Mode* specify the import mode, which means, select the destination of the layout data. Available choices are:

  ○ Import into current layout where layers are added or overwritten,

  ⊙ Import as new layout in same panel or

  ○ Import as new layout in new panel.
- Decide about the layer mapping mode ☐ Free layer mapping.

  This option checked allows an arbitrary mapping between PCB layers and GDS layers:

  – Specify the files to load on the next dialog page *Files*.
  – Specify the target layers for the layout on the *Layout Layers* dialog page.
  – Fill-in the input to output mapping matrix which assigns one or many output layers to each input file on the *Layer Stack* dialog page.

  While unchecked allows metal stack mapping which 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 target layers for the layout, the GDS layer stack, which means the complete stack available for mapping PCB data into on the *Layout Layers* dialog page. The idea is basically to put another set of metal-via-metal layers series on top of the GDS layer stack. The target layers should reflect the physical layer stack as seen from the chip for flip-chip mounting. Metal layers interleave with via layers. The first layer specified will be the closest to the chip surface.
  – Specify the chip mounting that determines the order by which the artwork layers are assigned to layout layers on the *Layer Stack* dialog page. To assign the top PCB layer to the first layout layer select Top Mounting or else, select Bottom Mounting to assign the top PCB layer to the last layout layer.

    On the same dialog page enter the number of metal layers and via types.
  – Enter the file names of the artwork files on the *Artwork Files* dialog page.
  – Specify drill types, i.e. the start layer, the stop layer and the related drill file on the *Drill Types And Files* dialog page. Specify 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*.
- On the second last dialog page *Coordinate Mapping*, specify up to three reference point coordinates on PCB and layout, each. Leave fields empty to specify less reference points. One point is used to derive the displacement, further points are used to derive the orientation. Currently no magnification is implied and only simple rotations are derived from the mapping points.

  Alternatively a transformation imported to existing layout can be specified using the entry field Explicit transformation <transformation expression>, whereby reference points have a higher priority. For the transformation expression use the common notation, i.e. "(*2 r90 10,-100)", referee to section 6.3: Transformations in **KLayout**.
- On the last dialog page *Options* specify a layer properties file to load or leave this entry field empty to not load any file. A loaded file is applied to the final layout. Hence, if PCB data are imported to an existing layout, the layer properties file should not only contain the PCB layer properties but the layout layer properties as well.

On the *Import Options* section specify the number of points per circle, where the minimum number is four points. Select weather polygons should be merged to remove all overlaps after importing or not. Specify the database unit for new layouts, whereat the preset value is 0.001 micron. And enter the top cell name for new layouts, whereat the preset name is "PCB".

- After filling in all specification save the settings to a file for later re-use by use the File ... button, entry Save Project. After that import the PCB Gerber data by pressing Import button.

10.4.1 The import dialog

The import dialog is organized in multiple pages that reflect the work-flow for the import specification. On every page, the File ... button allows to save the current settings as a PCB import project Save Project, to open an existing project Open Project or to create a new project and restart from scratch New Project.

The first dialog page *General* offers some basic options, compare with fig. 10.4:

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, in this case, layers will be added to the current layout. Alternatively, a new layout can be created which will be either added to the current panel or placed into a new one.

○ Import into current layout where layers are added or overwritten,
⊙ Import as new layout in same panel or
○ Import as new layout in new panel.

Layer mapping mode Specify here whether to use free or layer stack mode ☐ Free layer mapping. Check the box to use free layer mapping mode.

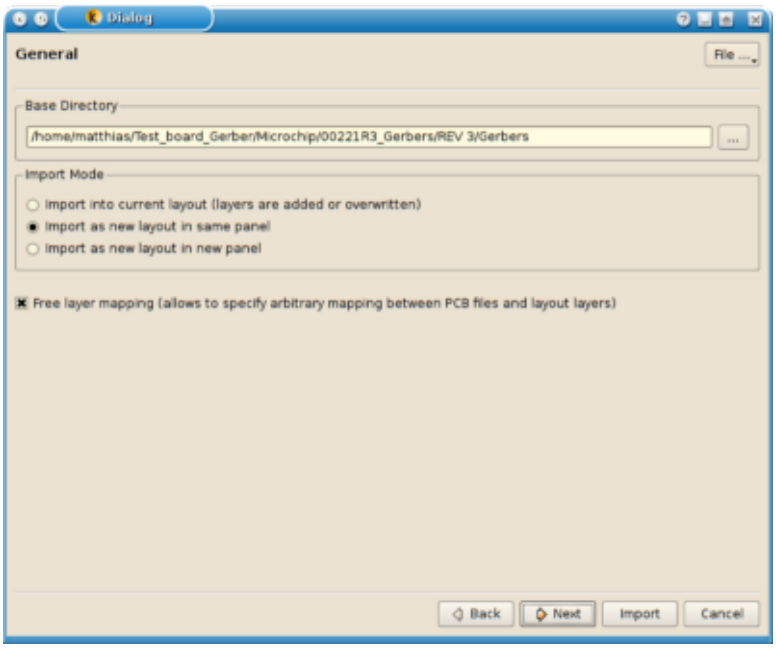

Figure 10.4. Import Dialog — General

10.4.2 The layer stack flow

In the layer stack flow, on the first dialog page, *Layout Layers*, compare with fig. 10.5, 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 [↑] and [↓]. Use the [✖] button to remove all selected layers.

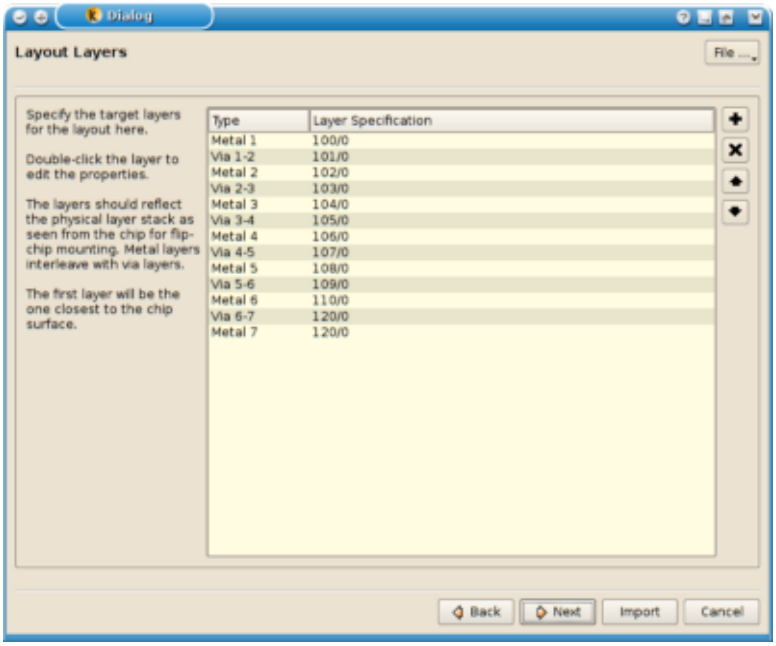

Figure 10.5. Import Dialog — Layout Layers

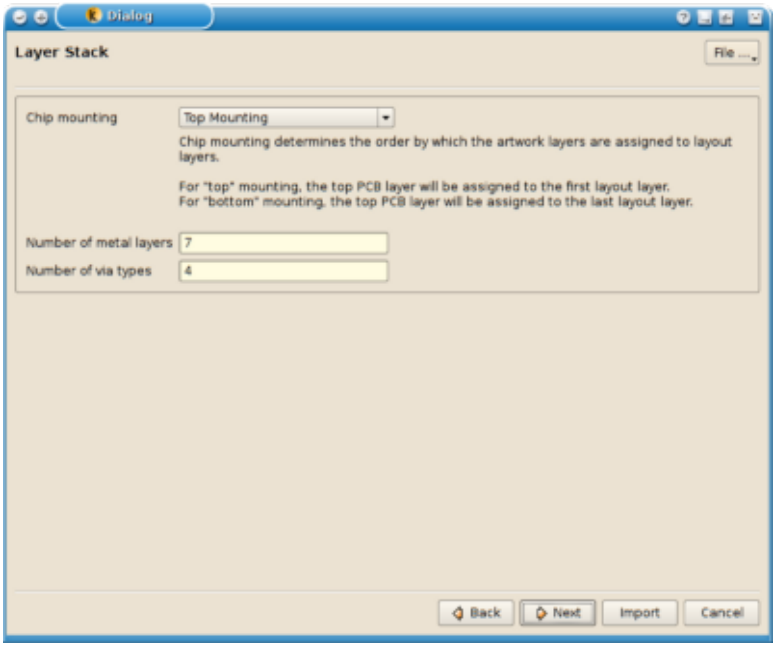

Figure 10.6. Import Dialog — Layer Stack

On the next dialog page, *Layer Stack*, see fig. 10.6, 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.

In addition 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.

On the *Artwork Files* dialog page, see fig. 10.7, the artfile file names must be entered. They are automatically assigned to the respective metal layers. The assignment order depends on the mounting mode.

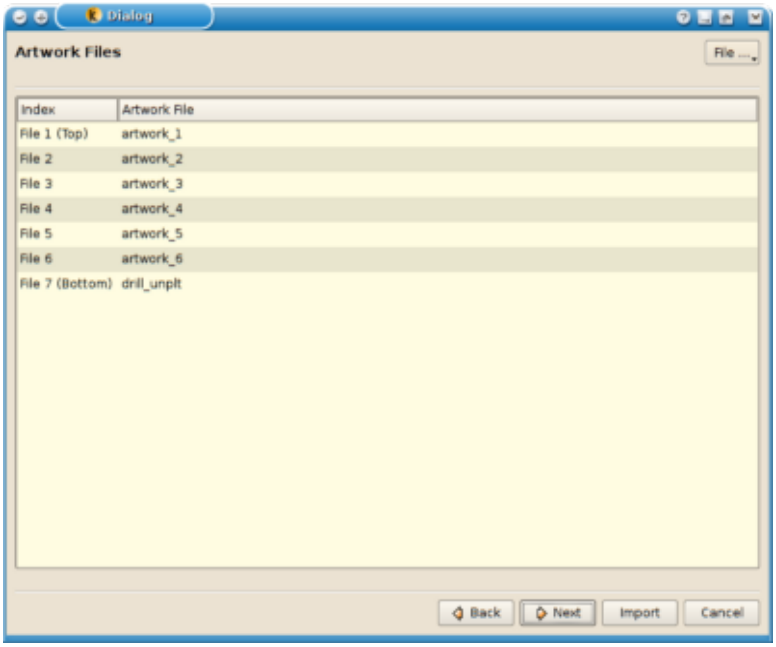

Figure 10.7. Import Dialog — Artwork Files

On the *Drill Types And Files* dialog page, compare fig. 10.8, 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.

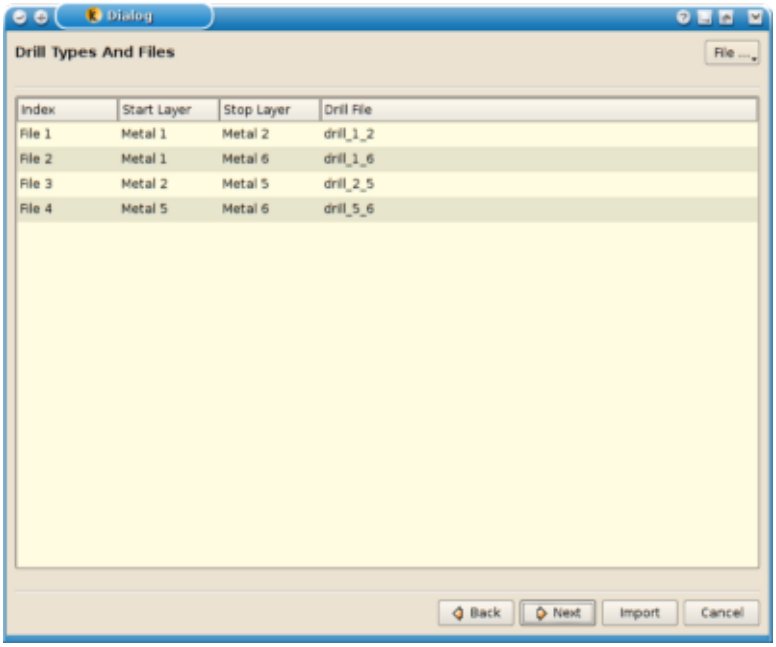

Figure 10.8. Import Dialog — Drill Types And Files

10.4.3 The free layer mapping flow

On the *Files* dialog page, see fig. 10.9, 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 ⬆ and ⬇ to move the selected entries up or down. Use the ✖ button to delete files from the list and use the ✚ button to add new files.

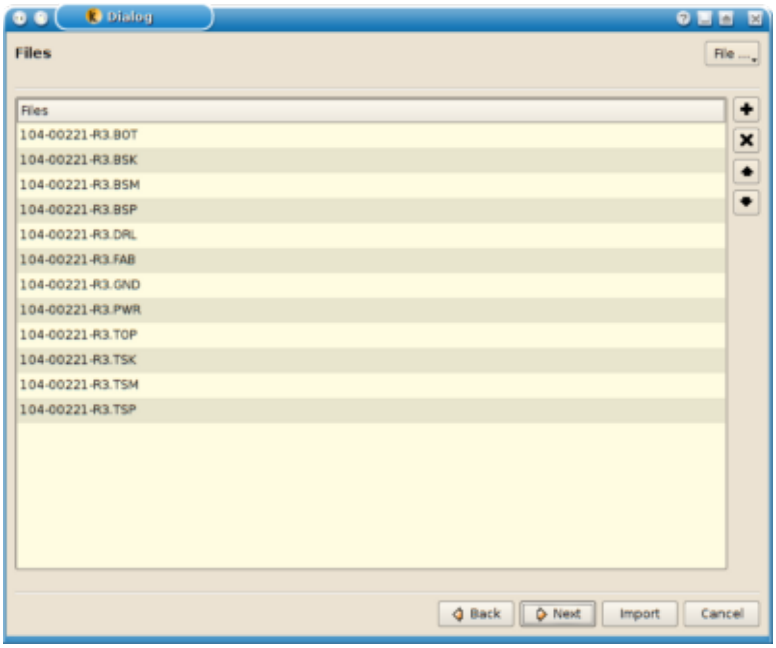

Figure 10.9. Import Dialog — Files

On the *Layout Layers* dialog page, compare fig. 10.10, 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 ⬆ and ⬇ to move selected entries and the ✚ or ✖ button to add new entries or deleted the selected ones, respectively.

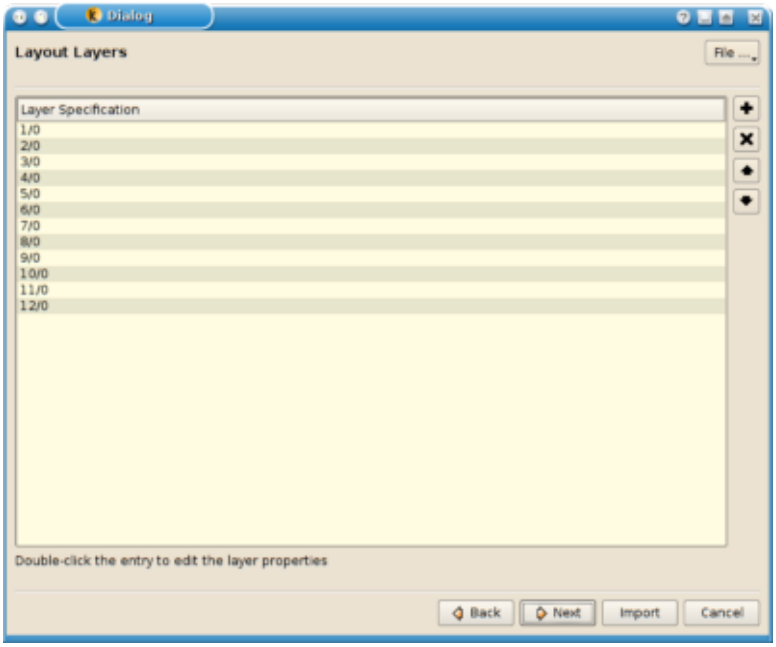

Figure 10.10. Import Dialog — Layout Layers

On the *Layer Mapping* dialog page, see fig. 10.11, each file can be assigned to one or more 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 ✖ button on the left to reset all marks for the rows selected.

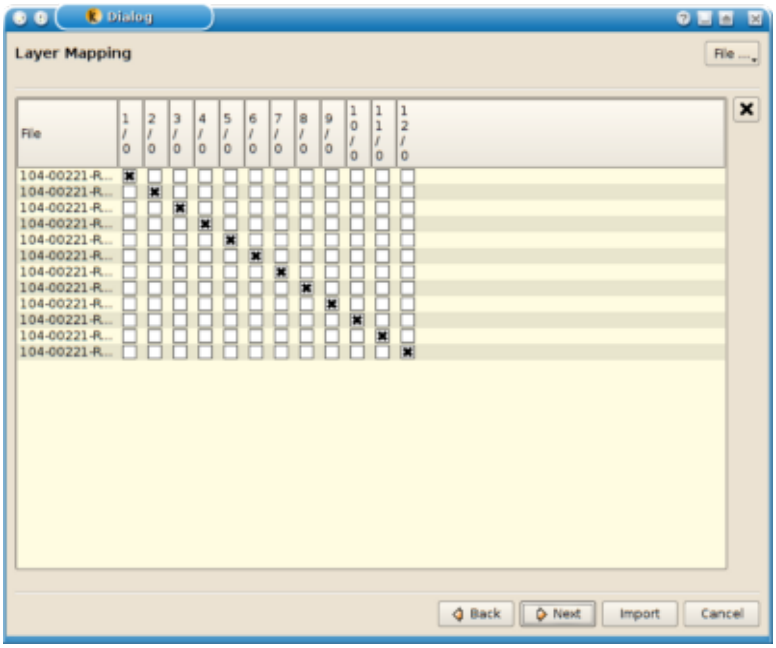

Figure 10.11. Import Dialog — Layer Mapping

10.4.4 General options

The *Coordinate Mapping* dialog page, see fig. 10.12, allows to specify 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.

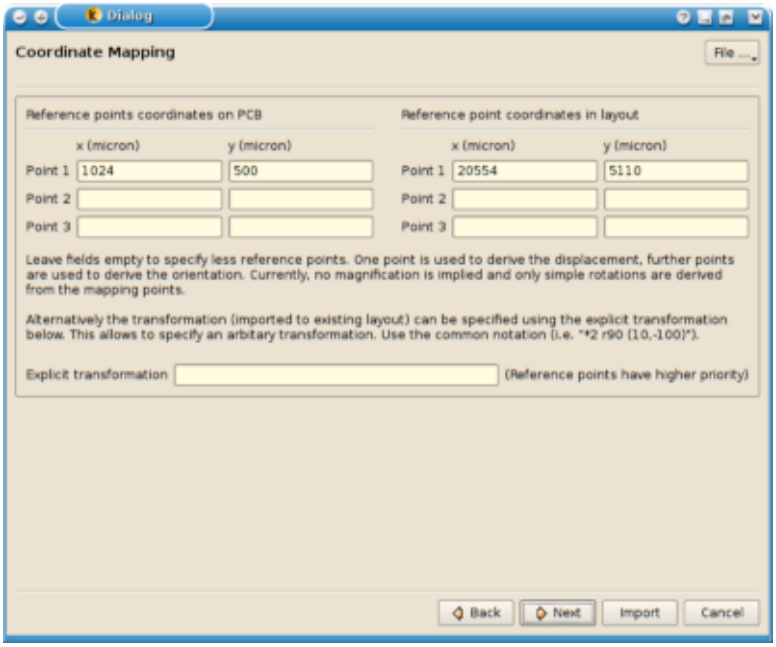

Figure 10.12. Import Dialog — Coordinate Mapping

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 entry field at bottom. 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 section 6.3: 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* dialog page, compare with fig. 10.13, 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 to choose 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 to choose the database unit and top cell name for new layouts. This applies only, if the import mode implies a new layout.

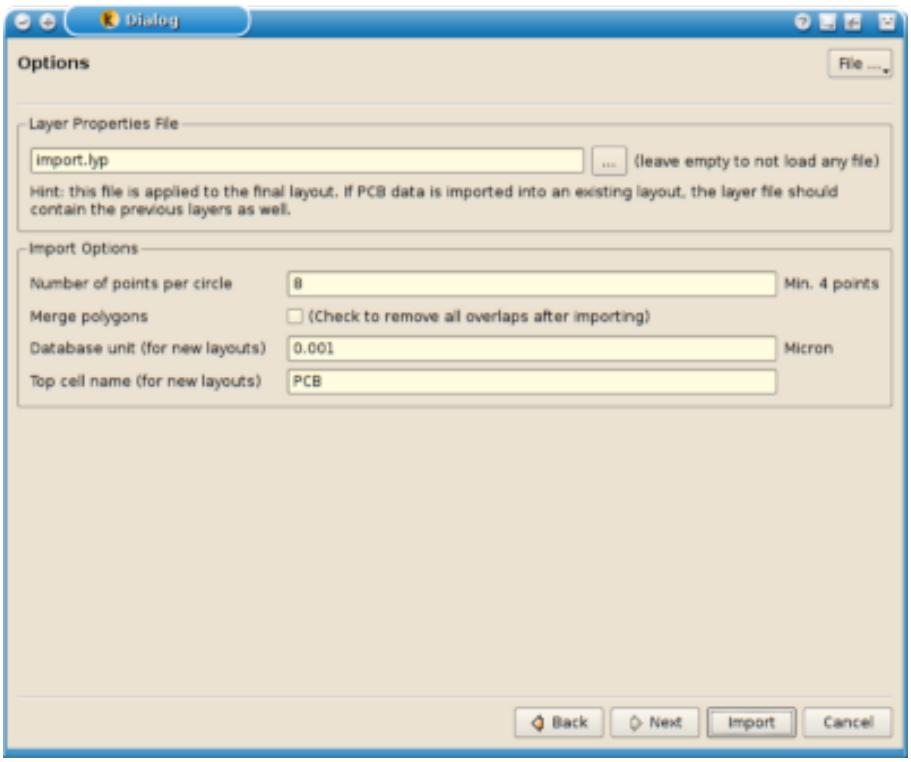

Figure 10.13. Import Dialog — Options

10.5 Importing 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 as File > Import > Other File Into Current.

The work-flow 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 data-type.
- 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 cells 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.

10.6 The net tracing feature

The net tracing function allows to trace a net by detecting touching shapes that together form a conductive region. It allows to specify a metal 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 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 to trace 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 *Net Trace* 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.

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 info 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 net info is displayed in more details if button ☐ Detailed is pressed and can be exported as new cell using button Export or as text to a file using Export To Text. In the first case a dialog *Export Net* opens where the cell name to export to can be entered, while in the later a dialog *Save Export Net* opens where a file name to export to in XML format can be entered using the default extension `1yn`.

The Trace Path function works similar but allows to specify 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 *Net Trace* 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.

Part IV

Ruby Scripting Interface (RBA)

Chapter 11

RBA Introduction

An introduction into the ruby based automation API.

Content

11.1 Using RBA scripts

To use RBA scripts, **KLayout** must be compiled with the ruby interpreter. This is done by giving the build script the paths to the ruby headers and library.

For example:

Listing 11.1: Command Line Input – Build Script for Ruby Support

```
build.sh -rblib /usr/lib/libruby1.8.so \
         -rbinc /usr/lib/ruby/1.8/i486-linux
```

Build script option "-rblib" takes the path to the ruby shared object, option "-rbinc" the location of the ruby headers, specifically `ruby.h`. Currently, ruby version 1.8 is required.

To use RBA, the script location must be passed to **KLayout** using the "-r" option. In this example the file `hello_world.rb` is placed in the directory defined by $KLAYOUTPATH:

Listing 11.2: **KLayout** Command Line Input – Ruby Script

```
klayout -r hello_world.rb
```

If used this way, all RBA functionality must be put into one script. Usually, this script will provide all the classes and definitions required and register new menu items and handlers.

11.2 Basic RBA

The ruby script given with the "-r" option is executed before the actual application is started. In fact, the application execution is initiated by the script, if one is given. In order to make the application start, the ruby script must contain at least this statement:

Listing 11.3: Ruby Code – Application Start

```
RBA::Application.instance.exec
```

"RBA" is the module provided by **KLayout**. Application is the main controller class (a singleton) that refers to the application as a whole. It provides the exec method which runs the application and returns if the main window is closed.

In most cases, the script will perform initialization steps before calling exec and may do cleanup once the application returned. Initialization may involve loading of layouts, registering menu items, initializing other resources etc.

In larger applications however, source code is usually organized into libraries and a main code part. Libraries and supplementary code can be loaded prior to the loading of the main source with the "-rm" option. In contrast to Files containing main source code, and therefore loaded with "-r" option, Files loaded with "-rm" option do not need to (and in fact must not) contain the RBA::Application.instance.exec call. This allows to provide independent libraries and initialization code to a RBA script environment:

Listing 11.4: **KLayout** Command Line Input – Ruby Libraries And Module

```
klayout -rm setup1.rb -rm setup2.rb -r hello_world.rb
```

RBA code can be installed globally by creating a file called `rbainit` in the same directory than the **KLayout** binary. If such a file is encountered, it will be executed as the first and before all files specified with "-rm" and "-r" are read.

11.3 A simple example

This example script registers a new menu item in the toolbar, which displays a message box saying "Hello, world!" when selected, and runs the application:

Listing 11.5: Ruby Code – New Menu – Hello World

```
class MenuHandler < RBA::Action
  def triggered
    RBA::MessageBox::info( "Info", "Hello, world!",
            RBA::MessageBox::b_ok )
  end
end

app = RBA::Application.instance

$menu_handler = MenuHandler.new
$menu_handler.title = "RBA test"

menu = app.main_window.menu
menu.insert_item("@toolbar.end", "rba_test", $menu_handler)
menu.insert_item("tools_menu.end", "rba_test", $menu_handler)

app.exec
```

This simple example already demonstrates some important concepts:

Reimplementation The menu item's functionality is implemented by reimplementing the Action object's triggered method. This method is called when the menu item is selected.

Delegation The menu item is not implemented directly but the implementation is delegated to an Action object. The action provides the "slot" that the menu item refers to. One action may be used for multiple menu items. The action does not only provide the implementation but the title, keyboard shortcut and other properties of the menu item. This way, the action may be used in multiple places (i.e. menu and toolbar) and still appear the same.

Menu item addressing The menu item is addressed by a "path" expression. In this case, the path is used for specifying the place where to insert the item. The path "@toolbar.end" instructs the menu controller to insert the item at the end of the toolbar. The path "tools_menu.end" instructs it to insert the item at the end of the Tools menu. The second string passed to "insert" is the name of the new item. After inserting, the new item can be addressed with the path "@toolbar.rba_test" and "tools_menu.rba_test".

Ownership of objects RBA is not able to guarantee a certain lifetime of an object, because Ruby and C++ implement different lifetime management models. Specifically, for the action object this means, that the menu controller, which is implemented in C++ cannot tell ruby that it keeps a reference to the action object. Without further measures, ruby will ignore this relationship and delete the action object – the menu item will disappear. To overcome this problem, an explicit reference to the action object must be held. In this case, a global variable is used ("$menu_handler"). This could as well be a member of an object or an array member.
It is very important to keep this aspect in mind when designing RBA applications.

Documentation for the various classes involved can be found in chapter 13: RBA Reference.

11.4 Extending the example

To give the menu callback a more "ruby style" look, a wrapper can be created what allows to attach code to the menu in the style of a ruby iterator. Now the callback uses "yield" to execute the code attached to the menu. In addition, the menu item now uses an icon and the keyboard shortcut ⇧+F7:

Listing 11.6: Ruby Code – New Menu – Hallo World Extended

```
class MenuHandler < RBA::Action
  def initialize( t, k, i, &action )
    self.title = t
    self.shortcut = k
    self.icon = i
    @action = action
  end
  def triggered
    @action.call( self )
  end
private
  @action
end

app = RBA::Application.instance

$menu_handler = MenuHandler.new( "RBA test", "Shift+F7",
        "icon.png" ) { RBA::MessageBox::info( "Info",
        "Hello, world!", RBA::MessageBox::b_ok )
        }

menu = app.main_window.menu
menu.insert_item("@toolbar.end", "rba_test", $menu_handler)
```

```
menu.insert_item("tools_menu.end", "rba_test", $menu_handler)

app.exec
```

11.5 Events

Starting with version 0.21 RBA features "events". Events allow to specify a Ruby block which is called when a certain condition takes place. Using events eliminates the need for deriving a method from an existing class. In particular, with version 0.21 RBA::Action features one event called on_triggered. A block associated with this event is called, when the action is triggered.

With events the example looks like that:

Listing 11.7: New Menu – Hallo World Using Events

```
app = RBA::Application.instance

$menu_handler = RBA::Action.new
$menu_handler.title = "RBA test"
$menu_handler.shortcut = "Shift+F7"
$menu_handler.icon = "icon.png"

# install the event
$menu_handler.on_triggered {
  RBA::MessageBox::info( "Info", "Hello, world!",
          RBA::MessageBox::b_ok )
        }

menu = app.main_window.menu
menu.insert_item("@toolbar.end", "rba_test", $menu_handler)
menu.insert_item("tools_menu.end", "rba_test", $menu_handler)

app.exec
```

11.6 Brief overview over the API

This section describes the main classes that the API provides. The link provides detailed information about the classes. The documentation uses a special notation to describe the characteristics or a method and the arguments:

[static] A class method is "static" (this is the terminology used in C). Such a method can be called without an object using the notation Class.Method or Class::Method. Often these methods are constructors, i.e. they create objects given a set of parameters.

[event] This definition is an "event". An event is a block of code that is executed when the specified event happens. See the events example above how to use events. The parameters specified in an event declaration describe the block arguments that are passed to the event handler block.

[const] A method is "const", if it does not change the state of an object. This for example applies to read accecsors that just retrieve information but do not alter the object's state.

ref (for return values) Some methods return references to objects. This means that Ruby does not receive a copy of the object but rather a pointer. From the Ruby perspective, this does not make a difference. From the C++ perspective it means, that the C++ code is the owner of the object and controls the object's lifetime.

[const] ref (for return values) Constance references are similar to references. However, on such references, only "const" methods may be called.

ref (for arguments) Such arguments receive a reference to the given object. From the C++ perspective this means, that Ruby is controlling the object's lifetime. Specifically that means that ruby must maintain an explicit reference to such an object since otherwise the object gets destroyed by Ruby's garbage collection mechanism which will either withdraw the object from C++ context or (worse) leave an invalid reference within C++.
The Action objects are special in this respect: Technically, Action objects are references itself. Even through Action objects are passed by value, they behave as being passed by reference.

yield ... Some methods are iterators. This means that code can be attached to them, which is called for each object are value delivered by this iterator. This follows the philosophy of Ruby. However, in some places, "real" iterators are used, i.e. LayerPropertiesIterator.

Following a brief description of the main classes and the concepts connected with them:

Class	Description
Application	This is the main application class, see section 13.5. There is only one instance representing the application (a "singleton"). The instance can be retrieved with the instance method. The Application object allows to configure the application on a high level and to retrieve the MainWindow object, the next basic object.
MainWindow	This class represents the main window, see section 13.44. Since there is only one main window per application currently, there is only one MainWindow object. This object is managed by the Application object. The main window mainly acts as a container for the "layout views", represented by LayoutView objects. Each view is equivalent to a tab panel in the main window. The main window manages the views and allows to close views, open new ones and allows to retrieve references to the corresponding LayoutView objects.
LayoutView	A Layout View represents the "canvas" on which one or more layouts are drawn, see section 13.42. The layouts to draw are called "cell views", because basically they show a single cell from a collection of cells. A cell view is represented by a CellView object, see section 13.13. Multiple cell views can be present in a single LayoutView object. The "layer views" control, how the cell views are drawn. Basically each layer view is a recipe how to draw one layer of one cell view and how to show it (colors, fill pattern, transformations etc.). Layer views can be arranged hierarchically such that groups are formed with parent nodes controlling the appearance of a group of layer views from a central point. Layer views are represented by LayerPropertiesNode objects, see section 13.40.
Layout	The Layout object represents the layout database, see section 13.41. Layouts are associated with CellView objects. In principle, multiple CellView objects may refer to the same Layout. A layout is organized in cells and layers. Each cell contains shapes on the same set of layers and optionally a set of instances of other cells. Layout layers must not be confused with the layer views: a layer view is the recipe how to display a layer from a layout object. A set of various classes comprise the layout API. The main classes are: Cell, Shape, CellInstArray, Trans, Box, Polygon and others.

11.7 RBA and QtRuby

QtRuby is a binding of the Qt API which has been made available for Ruby. This project also supports the Qt4 API (qtruby4). It is available as package for all major Linux distributions. Since **KLayout** is built upon Ruby, it integrates very well with QtRuby. In particular:

- QtRuby can access **KLayout**'s widget hierarchy and use Qt's meta object interface to identify **KLayout**'s widget classes.

- QtRuby and **KLayout** share the same message loop which enables advanced applications such as running a TCP server within **KLayout**'s process for IPC purposes.
- QtRuby can modify **KLayout**'s widget hierarchy and modify or alter the appearance of **KLayout**. This feature has to be used carefully however since **KLayout** does not take only limited care of foreign code modifying the UI.

I have prepared two examples which demonstrate how to use QtRuby:

1. Using QtRuby I – Adding a custom dialog section 12.8 and
2. Using QtRuby II – Transforming **KLayout** into a HTTP server section 12.9.

The following sub sections describes a few technical notes in detail.

11.7.1 Execution context

By default, the **KLayout** application runs outside the Ruby interpreter's context. The interpreter is entered only on request (i.e. if a menu is bound to a ruby script and the script needs to be executed. For QtRuby however, it is necessary that the whole application runs in the interpreter context. Otherwise Ruby code being executed in response to a UI event can crash the application (because it runs outside the interpreter). In particular error handling is not provided in that case and the application will issue a segmentation fault.

To run **KLayout** in the interpreter context, provide a central script that contains this line as the last line of code:

Ruby Code 11.8: Application Start

```
RBA::Application.instance.exec
```

Run this script with the "-r" option, so **KLayout** does not use it's own exec() call. Then, the whole application will run inside the interpreter and Ruby errors are handled properly.

11.7.2 Interfacing between QtRuby and RBA objects

Although RBA and QtRuby seem similar on the first glance, they are built upon a different system. For some objects, namely the main window object, QtRuby and RBA provide two different views to the same basic Qt object. The RBA view gives access to the methods and properties exported by RBA while the QtRuby view accesses the QtMainWindow interface. Both can interact but usually that is a bad idea because it will interfere with **KLayout**'s internal bookkeeping. It's safe however to control Qt features (such as adding dialogs as logical children) through the QtRuby interface and **KLayout**'s features through the RBA interface.

Because it's particular interesting, here is the code to obtain the main window's QtRuby and RBA interface:

Ruby Code 11.9: Ruby Code – QtRuby interface of the main window

```
    # QtRuby interface of the main window
    qt_main_window = Qt::Application.topLevelWidgets.select {
        |w| w.class.to_s == "lay::MainWindow"
      } [0]
```

Ruby Code 11.10: Ruby Code – RBA interface

```
    # RBA interface of the main window
    rba_main_window = RBA::Application.instance.main_window
```

For a brief introduction into QtRuby see KDE TechBase Ruby.

11.8 What can be done and what can't

Following examples for what can be done with RBA:

- Customizing the menu, i.e. redefining the keyboard shortcuts or rearranging the menu
- Customizing the layer view list, managing custom stipple pattern
- Automation of tasks like loading of layouts, doing screen shots etc.
- Generating layouts dynamically, i.e. for annotation of other layout or visualization purposes
- Linking **KLayout** to other applications or databases for example
- Adding custom browsers using the HTML browser dialog (see BrowserDialog documentation)
- Scanning the layout database (i.e. for marker shapes) and performing actions on the results
- Handling properties on shape level (adding and removing)
- Controlling rulers and markers (query, remove and create)
- Combining RBA with qtruby4 (a Ruby wrapper for Qt) to implement custom dialogs etc.
- Generating layout files (there is a "write" function to write a layout to a file).

And here comes an example for what can't be done with RBA currently:

- Responding to mouse clicks in the canvas (since there is no API for this yet).

11.9 More information

The basic source for more information is the RBA reference documentation. For a deeper understanding of the API, a look at the RBA examples given in chapter 12 might be helpful.

Documentation for older API versions are provided on **KLayout**'s Home Page:
Version 0.20, Version 0.19, Version 0.18, Version 0.17, Version 0.16.

Chapter 12

RBA Examples

This chapter contains some example scripts that hopefully are instructive and may serve as starting point for own experiments.

Content

12.1 Using the HTML browser dialog I: A location browser

The code for this example can be found here: browser.rb.
See chapter 11: RBA Introduction, for a description of how to run that script.

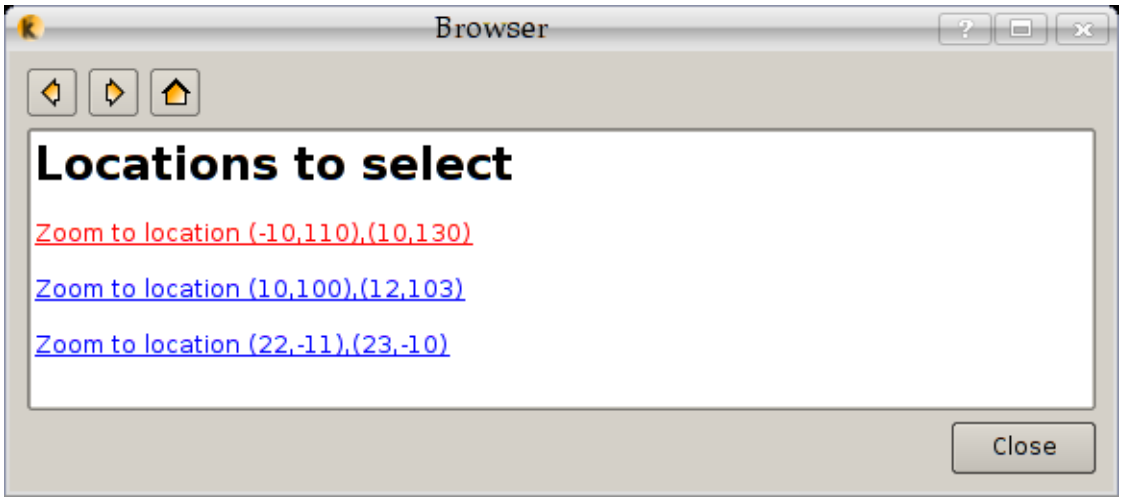

Figure 12.1. RBA Example 1 – Using the HTML browser dialog I – A location browser.

The HTML browser dialog is very handy to implement simple UI's based on HTML code and a client/server scheme. This setup is similar to that of the HTTP client/server pair. The BrowserDialog object acts as a HTML browser and a BrowserSource object can be used to deliver the HTML code for that browser.

More specific, each link with the "int:" scheme that the HTML browser encounters is resolved not by loading the appropriate resource but by asking the BrowserSource object to deliver the data for that URL. This scheme can be used to build user interfaces in the same way that a web application would implement a simple user interface.

In addition to simply delivering data, the BrowserSource object may perform actions on the **KLayout** API, such as zooming to a certain location, opening files, etc. This enables a new class of applications based on HTML and direct interaction with the application core.

The example given here employs this technique to implement a simple location browser: given a set of three locations, the user can browse to one of these locations by clicking the link. To try this application, load a layout and select the Browser item in the toolbar.

12.2 Using the HTML browser dialog II: A screen-shot gallery

The code for this example can be found here: sreenshots.rb.
See chapter 11: RBA Introduction, for a description of how to run that script.

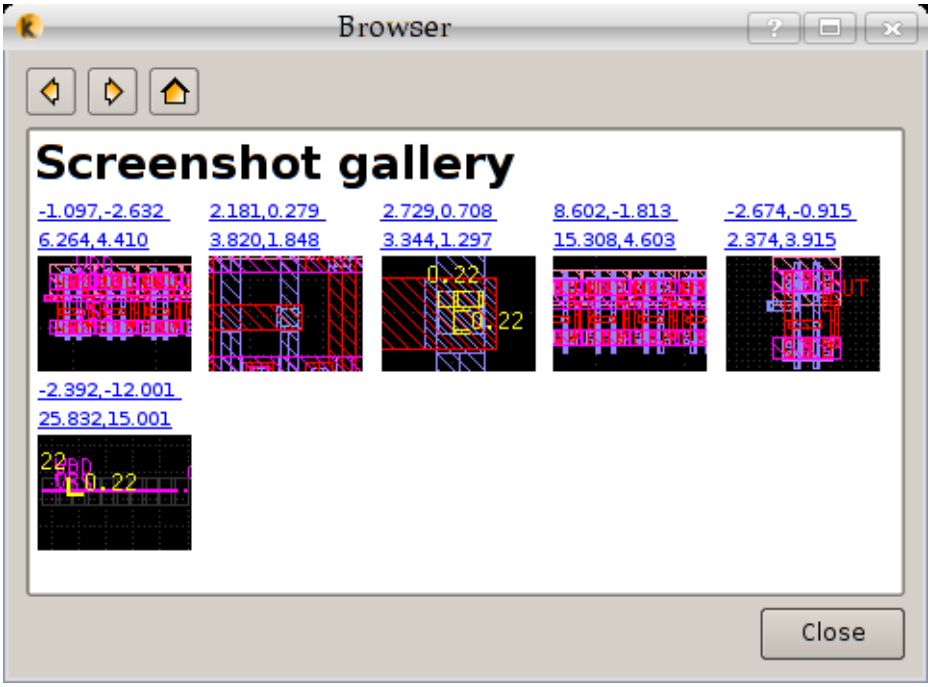

Figure 12.2. RBA Example 2 – Using the HTML browser dialog II – A screen-shot gallery

This example employs the HTML browser dialog to implement a simple screen-shot gallery: by clicking on the Add screenshot item in the toolbar, a screen-shot is taken and placed in the HTML browser window. Each screen-shot will be represented by a thumbnail image and a screen-size image. The browser will display the thumbnails together with a link that will put the viewer to the original location. By clicking on the thumbnail image, the enlarged version is shown in the browser window.

12.3 Dynamic database manipulation: A "Sokoban" implementation

The code for this example can be found here: sokoban.rb.
See RBA Introduction, for a description of how to run that script.

This toy application dynamically changes the database to realize a game arena. As a trial application, it implements one level of the famous "Sokoban" game.

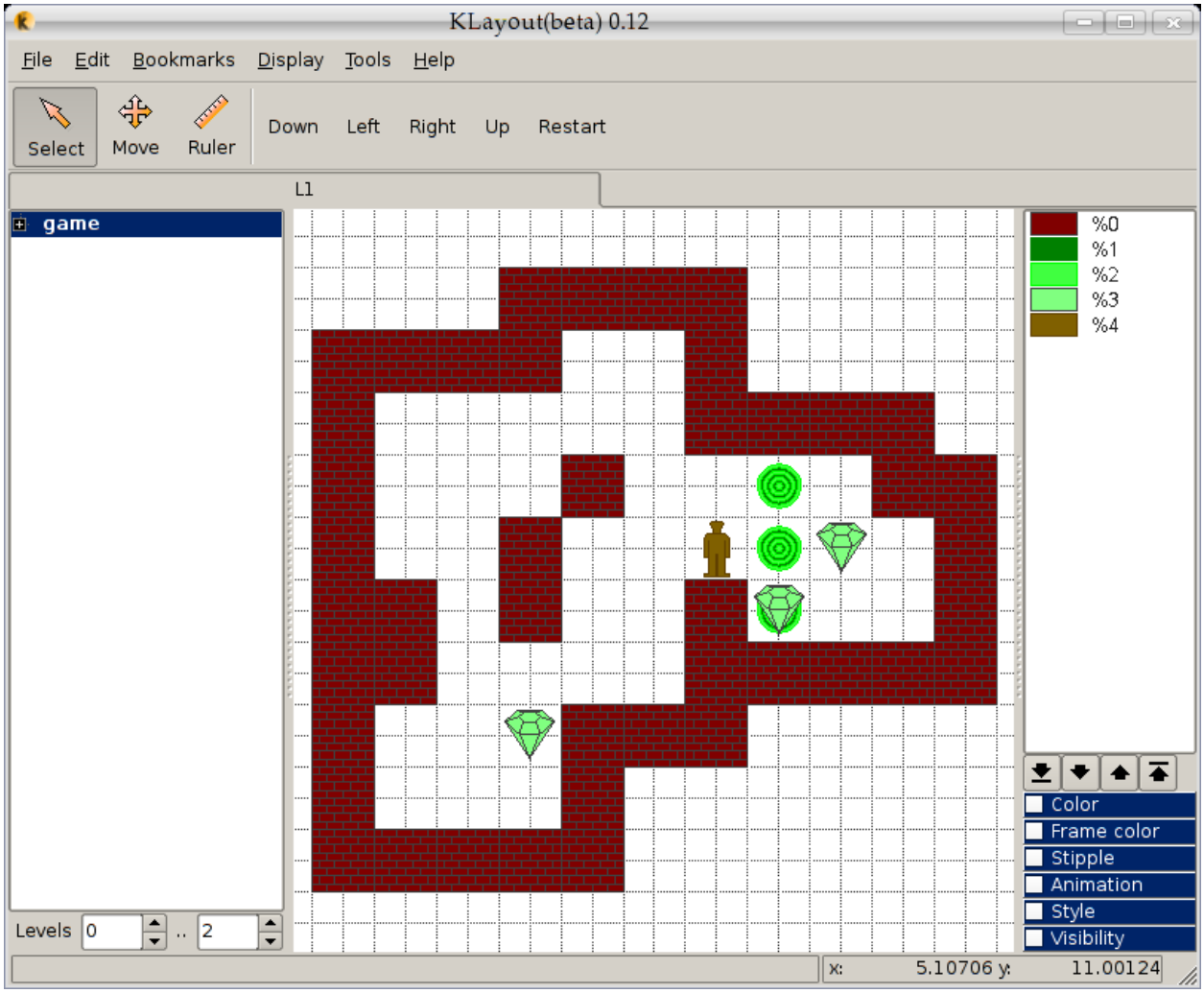

Figure 12.3. RBA Example 3 – Dynamic database manipulation – A "Sokoban" implementation

12.4 Creating layouts I: The Koch curve

The code for this example can be found here: fractal.rb.
See RBA Introduction, for a description of how to run that script.

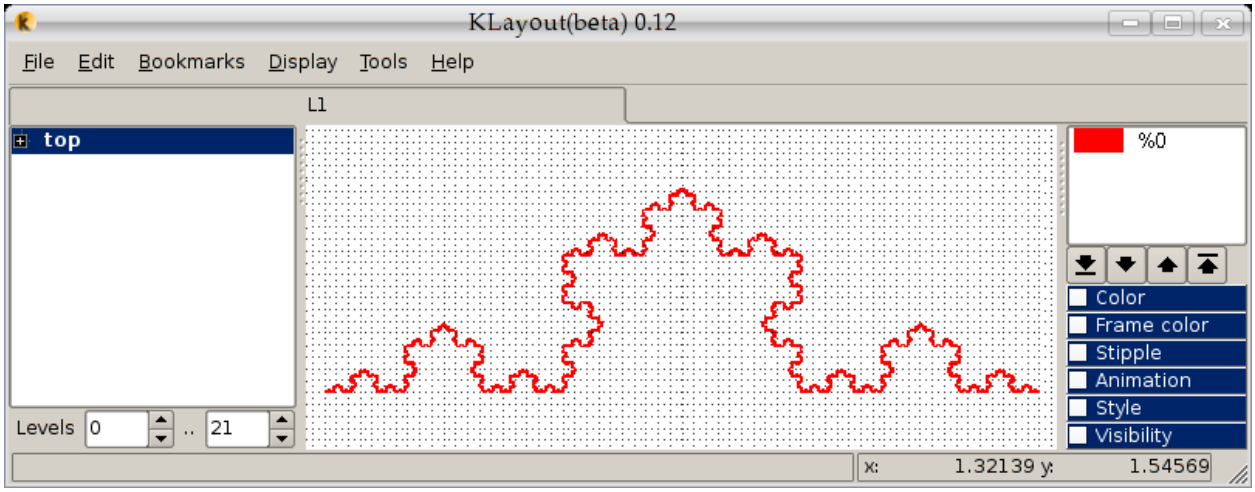

Figure 12.4. RBA Example 4 – Creating layouts I – The Koch curve.

This application creates a Koch curve which is constructed by the recursive application of a generation recipe. In our case, this recipe is implemented by instantiating cells. An exact implementation would require a cell to call itself, but this is not allowed in this frame-word. Instead, a set of up to 20 cells is created with each cell calling the successive one in the same fashion.

When zooming deeply into the curve, the viewer gets pretty slow which is a consequence of the performance de-rating of the underlying quad tree when the quads get really small. However, since this application is a pretty artificial one, I hope that this is not a serious imperfection ....

12.5 Creating layouts II: Data visualization

The code for this example can be found here: datamap.rb.
See RBA Introduction, for a description of how to run that script.

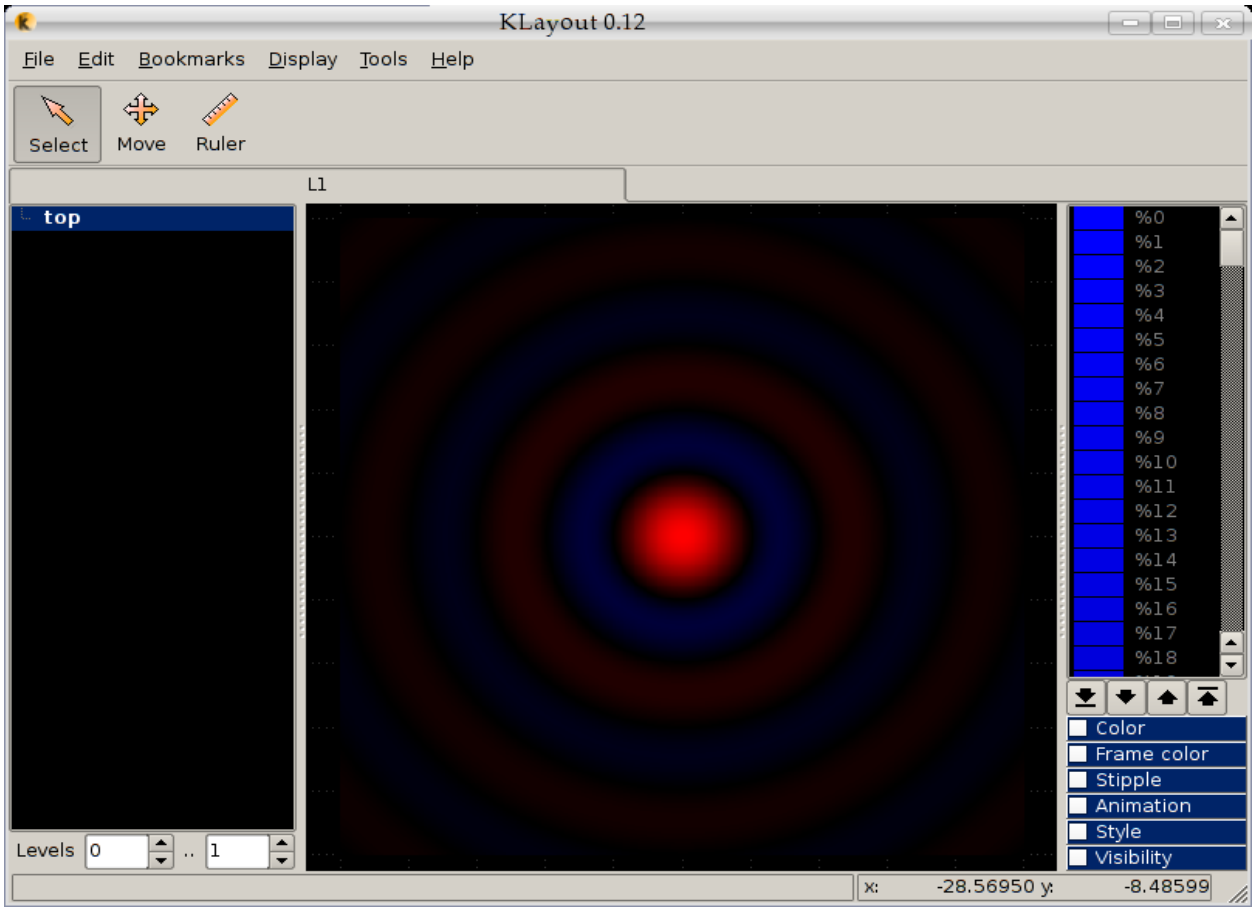

Figure 12.5. RBA Example 5 – Creating layouts II – Data visualization.

This application creates a 2-dimensional function plot by employing differently colored layers to display the pixel of the data map. 256 Layers are created representing values from -1.0 to 1.0 of the function "sin(r)/r". The function is evaluated on the 500 x 500 grid, each grid point is assigned a value, the value is mapped to a layer and a box is created to represent the pixel.

12.6 Menus: Dumping the menu structure

The code for this example can be found here: dump_menu.rb.
See RBA Introduction, for a description of how to run that script.

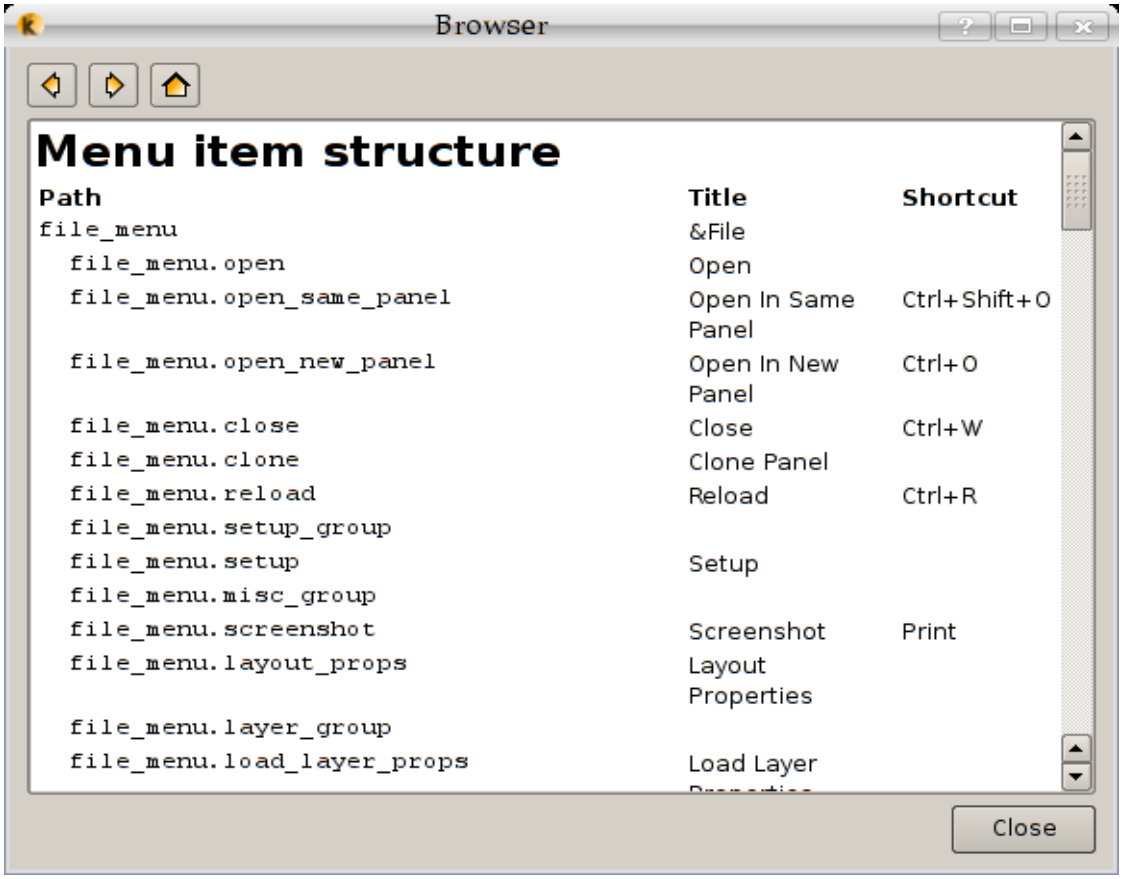

Figure 12.6. RBA Example 6 – Menus – Dumping the menu structure.

This application dumps the menu structure into a HTML browser window. Beyond acting as an example, this script is quite useful to visualize the menu structure and to determine insert points when installing new items.

12.7 Editing: Hierarchical propagation

The code for this example can be found here: flatten.rb.
See RBA Introduction, for a description of how to run that script.

This application provides two new toolbar entries bound to keys F7 and F8. The first function brings up all selected shapes and instances to the current cell level and removes them from their original cell. This makes sense only if the selection contains objects from sub-cells (hence not in "top level only" selection mode). The second function brings up such objects one level in hierarchy. Both functions just bring up objects along the selection path, not into all instances of the selected cell. They are very similar to the function Edit > Selection > Move up in hierarchy menu.

The new functions can only be used in "Edit" mode and require version 0.16 or later.

This code demonstrates in particular:

- How to use the selection set of objects.
- How to modify geometrical objects (transform, erase, copy).
- How to implement undo/redo support, which is pretty simple using the LayoutView's transaction and commit methods.

12.8 Using QtRuby I: Adding a custom dialog

The code for this example can be found here: qtrubydialog.rb.

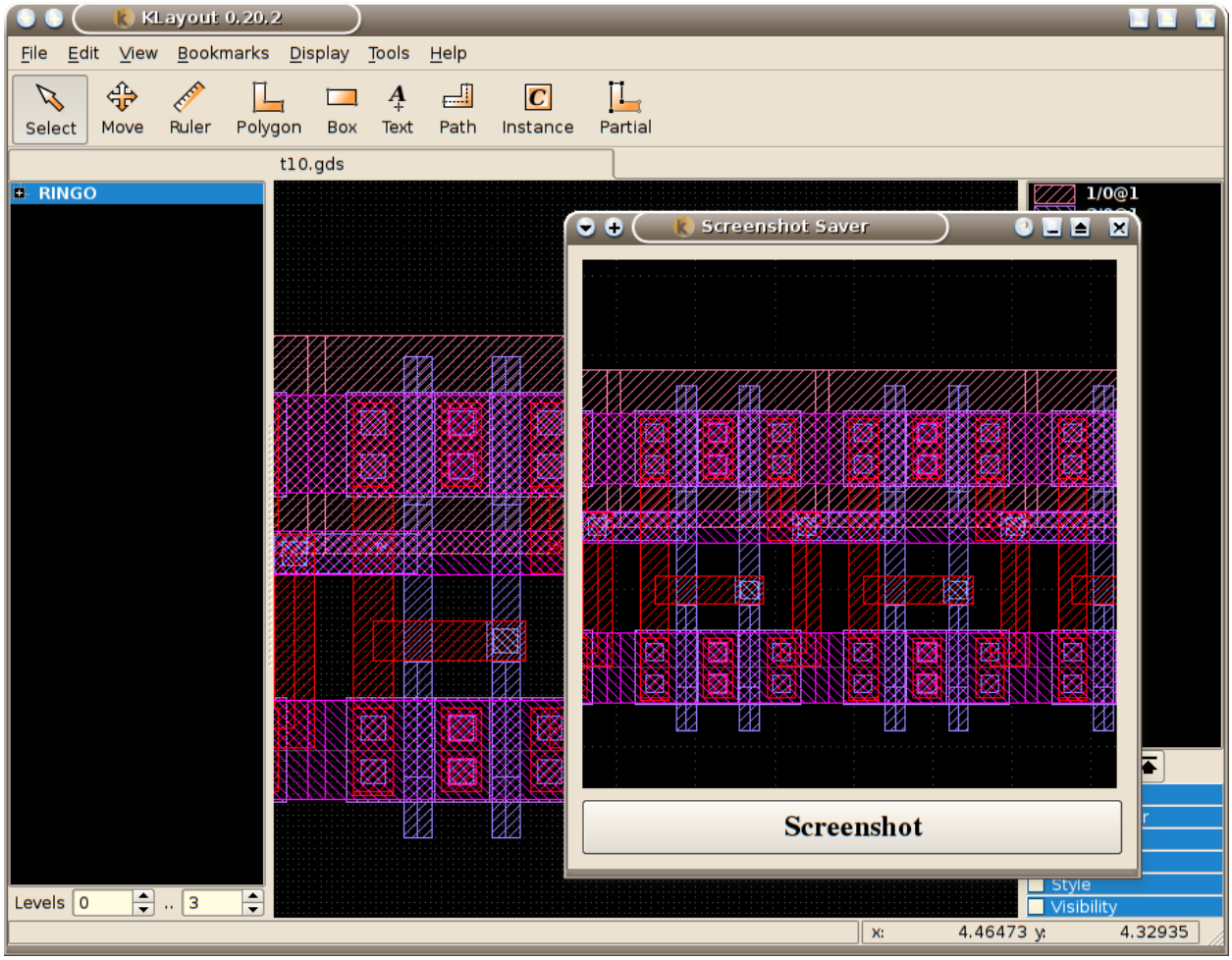

Figure 12.7. RBA Example 8 – Using QtRuby I – Adding a custom dialog.

For this script, it is important that it is run **KLayout** with the "-r" option, i.e.

Console Input 12.1: **KLayout** Command Line Input – Basics

```
klayout -r qtrubyserver.rb
```

The script will add a new dialog to **KLayout** which is opened when **KLayout** starts. It offers a Screenshot button which will take a screen-shot and display it in a label above the button.

This script demonstrates the basic technique of mixing **KLayout** objects with RBA objects. Although both live in different object spaces (RBA is built on a different basis that QtRuby), both share the same Qt object below. For that reason, QtRuby shares the event loop with **KLayout** and can access and even modify **KLayout**ś Qt widget hierarchy.

In particular, this line of code demonstrates how to obtain **KLayout**'s MainWindow widget:

Ruby Code 12.2: QtRuby interface of the main window

```
1   # QtRuby interface of the main window
2   qt_main_window = Qt::Application.topLevelWidgets.select {
```

```
        |w| w.class.to_s == "lay::MainWindow"
    } [0]
```

12.9 Using QtRuby II: Transforming KLayout into a HTTP server.

The code for this example can be found here: qtrubyserver.rb.

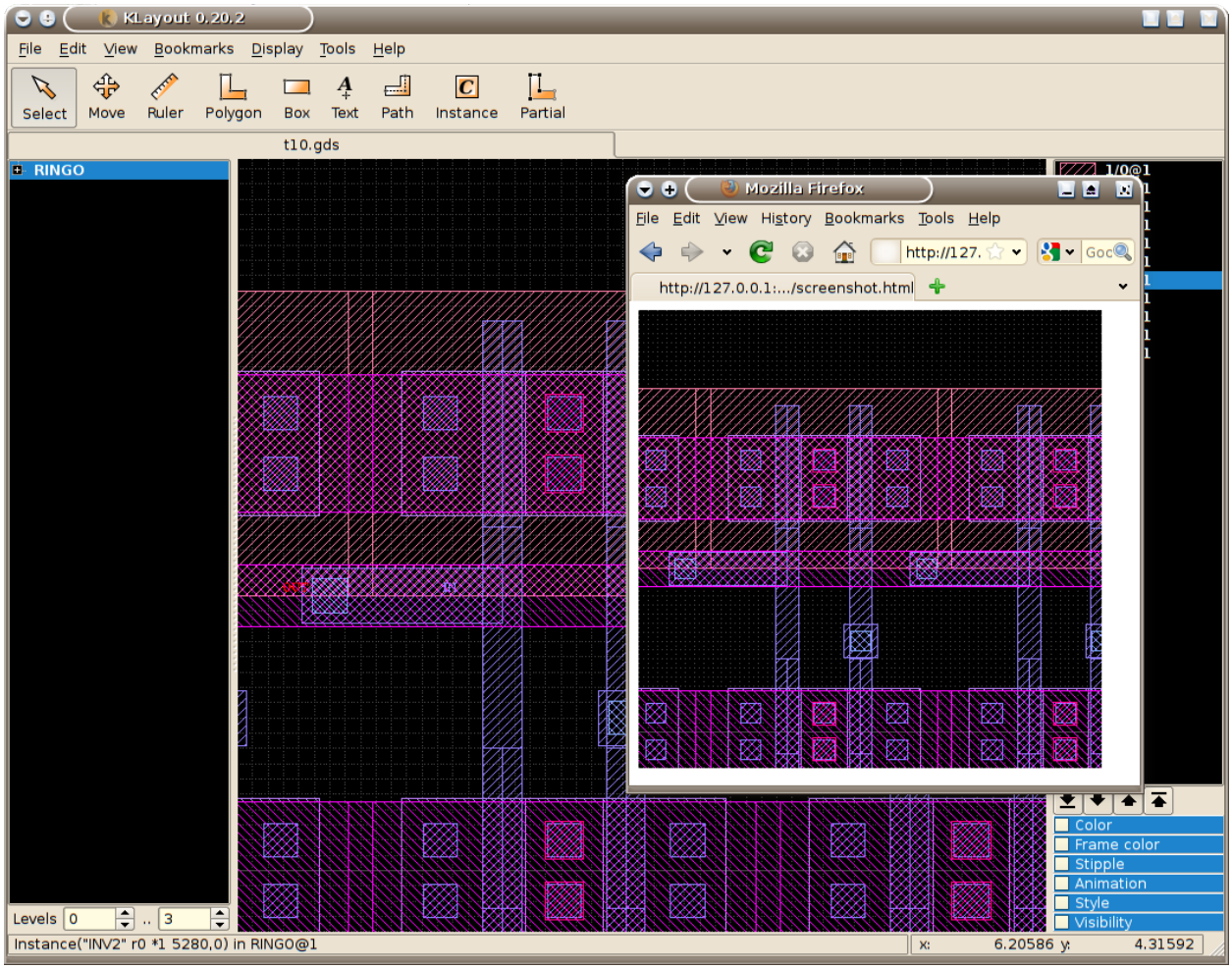

Figure 12.8. RBA Example 9 – Using QtRuby II – Transforming **KLayout** into a HTTP server

For this script, it is important that it is run **KLayout** with the "-r" option, i.e.

Console Input 12.3: **KLayout** Command Line Input – QtRuby Server

```
klayout -r qtrubyserver.rb
```

The script will open a TCP socket on port 8081 and listen to it while **KLayout** runs. In this example, the script will respond to incoming connections and implements and rather simple version of the HTTP protocol. If a browser is used on the local host to open this URL:

Console Input 12.4: Dialog Input – Transformation

```
http://127.0.0.1:8081/screenshot.html
```

Our simple server will respond with a HTML page containing a single image which shows a snapshot of the current screen. For a remote connection, 127.0.0.1 can of course be replaced by the IP address of the host running **KLayout**. Please note, that to run the example, you need to disable the proxy if your browser is configured to use one.

This script demonstrates the cooperation of QtRuby and **KLayout** which share the same event loop: The TcpServer object lives in the context of the application and can control the application through RBA objects. This principle opens a wide field of applications where **KLayout** is remotely controlled by external processes and over the network.

Chapter 13

RBA Reference

A comprehensive documentation of the ruby based automation API.

Class overview in alphabetic principle of arrangement

AbstractMenu	The abstract menu class.
Action	This class implements an event handler for a menu event.
ActionBase	An action.
Annotation	This class implements an "annotation object".
Application	The application object.
ArgType	The description of a type (argument or return value).
Box	A box class.
BrowserDialog	The HTML browser dialog.
BrowserSource	The BrowserDialog source for "int" URL's.
Cell	The cell object.
CellInstArray	A single or array cell instance.
CellMapping	A cell mapping derived from two hierarchies.
CellView	A "cell view" reference.
Class	The interface to the declarations of classes and methods.
CplxTrans	A complex transformation.
DBox	A box class.
DCplxTrans	A complex transformation.
DEdge	An edge class.
DPath	An path class.
DPoint	A point class with double (floating-point) coordinates.
DPolygon	A polygon class.
DSimplePolygon	A simple polygon class.
DText	A text object.
DTrans	A simple transformation.
DoubleValue	Encapsulate a floating point value.
Edge	An edge class.
EdgeProcessor	The edge processor (boolean, sizing, merge).
FileDialog	Various methods to request a file name.
ICplxTrans	A complex transformation.
Image	An image to be stored as a layout annotation.
ImageDataMapping	A structure describing the data mapping of an image object.
InputDialog	Various methods to open a dialog requesting data entry.
InstElement	An element in an instantiation path.

Instance	An instance proxy.
IntValue	Encapsulate an integer value.
LayerInfo	A structure encapsulating the layer properties.
LayerMap	An object representing an arbitrary mapping of physical to logical layers.
LayerProperties	The layer properties structure.
LayerPropertiesIterator	Flat layer iterator.
LayerPropertiesNode	A layer properties node structure.
Layout	The layout object.
LayoutView	The view object presenting one or more layout objects.
LoadLayoutOptions	Layout reader options.
MainWindow	The main application window and central controller object.
Manager	A transaction manager class.
Marker	The floating-point coordinate marker object.
MessageBox	Various methods to display message boxes.
Method	The interface to a method declaration.
ObjectInstPath	A class describing a selected shape or instance.
Observer	This class implements an event handler for use with 'observer' interfaces.
ObserverBase	The "Observer" base class.
ParentInstArray	A parent instance.
Path	An path class.
Point	An integer point class.
Polygon	A polygon class.
RdbCategory	The report database category.
RdbCell	A report database cell representation.
RdbItem	A RDB item.
RdbItemValue	A RDB value object.
RdbReference	A cell reference.
RecursiveShapeIterator	An iterator delivering shapes that touch or overlap the given region recursively.
ReportDatabase	The report database object.
SaveLayoutOptions	Options for saving layouts.
Shape	A shape proxy.
ShapeProcessor	The shape processor (boolean, sizing, merge on shapes).
Shapes	A collection of shapes.
SimplePolygon	A simple polygon class.
StringListValue	Encapsulate a string list.
StringValue	Encapsulate a string value.
Text	A text object.
Trans	A simple transformation.

13.1 Class AbstractMenu (version 0.21) The abstract menu class.

The abstract menu is a class that stores a main menu and several pop-up menus in a generic form such that they can be manipulated and converted into GUI objects.

Each item can be associated with an Action, which delivers a title, enabled/disable state etc. The Action is either provided when new entries are inserted or created upon initialization.

The abstract menu class provides methods to manipulate the menu structure (the state of the menu items, their title and shortcut key is provided and manipulated through the Action object).

Menu items and sub menus are referred to by a "path". The path is a string with this interpretation:

""	is the root
"[<path>.]<name>"	is an element of the sub menu given by <path>. If <path> is omitted, this refers to an element in the root.
"[<path>.]end"	refers to the item past the last item of the sub menu given by <path> or root.
"[<path>.]begin"	refers to the first item of the sub menu given by <path> or root.
"[<path>.]#<n>"	refers to the nth item of the sub menu given by <path> or root (n is an integer number).

Menu items can be put into groups. The path strings of each group can be obtained with the group method. An item is put into a group by appending **":<group-name>"** to the item's name. This specification can be used several times.

Detached menus (i.e. for use in context menus) can be created as virtual top-level sub menus with a name of the form **"@<name>"**. A special detached menu is **"@toolbar"** which describes all elements placed into the toolbar.

Method Overview

action	Get the reference to an Action object associated with the given path.
items	Get the sub items for a given sub menu.
is_menu	Query if an item is a menu item.
is_separator	Query if an item is a separator.
is_valid	Query if a path is a valid one.
insert_item	Insert a new item before the item given by the path.
insert_separator	Insert a new separator before the item given by the path.
insert_menu	Insert a new sub menu before the item given by the path.
delete_item	Delete the item given by the path.
group	Get the group members.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.1.1 [const] ActionBase action(path) Get the reference to an Action object associated with the given path.

Input:	path	The path to the item. This must be a valid path.
Return:	ref	A reference to an Action object associated with this path.

13.1.2 delete_item(path)
Delete the item given by the path.

Input:	path	The path to the item to delete.
Return:		No return.

13.1.3 destroy
Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.1.4 [const] boolean destroyed
Tell, if the object was destroyed.

Return:	true	The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
	false	The object wasn't destroyed.

13.1.5 [const] string[] group(group)
Get the group members.

Input:	group	
Return:	string[]	A vector of all members (by path) of the group.

13.1.6 insert_item(path, name, ActionBase action)
Insert a new item before the one given by the path.

The Action object passed as the third parameter references the handler which both implements the action to perform and the menu item's appearance such as title, icon and keyboard shortcut.

Input:	path	The path to the item as string to insert the new item before it.
	name	The name of the new item to insert.
	action	The Action object to insert.

13.1.7 insert_menu(path, name, title)
Insert a new sub menu before the item given by the path.

The title string optionally encodes the key shortcut and icon resource in the form:
<text>["("<shortcut>")"]["<"<icon-resource>">"].

Input:	path	The path to the item before which to insert the sub menu.
	name	The name of the sub menu to insert
	title	The title of the sub menu to insert.

13.1.8 insert_separator(path, name)
Insert a new separator before the item given by the path.

Input:	path	The path to the item as string to insert the separator before it.
	name	The name of the separator as string to insert.

13.1.9 [const] boolean is_menu(path)
Query if an item is a menu.

Input:	path	The path to the item.
Return:	true	The path is valid.
	false	The path is not valid or is not a menu item.

13.1.10 [const] boolean is_separator(path)
Query if an item is a separator.

This method has been introduced in version 0.19.

Input:	path	The path to the item.
Return:	true	The path is valid.
	false	The path is not valid or is not a menu item.

13.1.11 [const] boolean is_valid(path)
Query if a path is a valid one.

Input:	path	The path to check.
Return:	true	The path is valid.
	false	The path is not valid or is not a menu item.

13.1.12 [const] string[] items(path)
Get the sub items for a given sub menu.

Input:	path	The path to the sub menu.
Return:	string[]	Empty vector if the path is not valid or the item does not have children.
	path	The path string for the child item.
	path(1)...path(n)	A vector path string for the child items.

13.2 Class Action (version 0.21)
The event handler for menu events.

This class allows to re-implement the "triggered" handler to receive menu events. The Actionclass is derived from class ActionBase and inherits all it's methods.

Method Overview

triggered	This method is called if the menu item is selected.
on_triggered	This event is called if the menu item is selected.
title=	Set the title.
title	Get the title.
shortcut=	Set the keyboard shortcut.
shortcut	Get the keyboard shortcut.
is_checkable?	"is_checkable" attribute.
is_checked?	"is_checked" attribute.
is_enabled?	"is_enabled" attribute.
is_visible?	"is_visible" attribute.
checkable=	Make the item(s) check-able or not.
enabled=	Enable or disable the action.
visible=	Show or hide.
checked=	Check or uncheck.
icon=	Set the icon to the given picture.
icon_text=	Set the icon's text.
icon_text	Get the icon's text.
trigger	Trigger the action programmatic-ally.
assign	Assign the contents of another object to self.
dup	Creates a copy of self..
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.2.1 assign(Action other)
Assign the contents of another object to self.

This method assigns the contents of another object to self. This is a deep copy that does not only copy the reference but the actual content.

13.2.2 checkable=(boolean)
Make the item(s) check able or not.

Input:	true	Make the item check able.
	false	Make the item not check able.

13.2.3 checked=(boolean)
Check or unchecked

Input:	true	Make the item checked.
	false	Make the item unchecked.

13.2.4 destroyExplicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.2.5 [const] boolean destroyed
Tell, if the object was destroyed.

Return: true — The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
false — The object is still alive.

13.2.6 [const] Action dup
Creates a copy of self.

Return: Action The copy of self.

13.2.7 enabled=(boolean)
Enable or disable the action.

Return: true — Enable the item.
false — Disable the item.

13.2.8 icon=(file)
Set the icon to the given picture.

Input: file — The image file to load as icon for the menu item. Passing an empty string will reset the icon.

13.2.9 [const] icon_text
Get the icon's text.

Input: icon_text — The current icon text as string.

13.2.10 icon_text=(icon_text)
Set the icon's text.

Input: icon_text — The icon text as string to be set below the icon. If no icon text is given the normal text will be used for the icon. Passing an empty string will reset the icon's text.

13.2.11 [const] boolean is_checkable?
"is_checkable" attribute.

Return: true — The item is check able.
false — The item is not check able.

13.2.12 [const] boolean is_checked?
"is_checked" attribute.

Return:	true	The item is checked.
	false	The item is unchecked.

13.2.13 [const] boolean is_enabled?
"is_enabled" attribute.

Return:	true	The item is enabled.
	false	The item is disabled.

13.2.14 [const] boolean is_visible?
"is_visible" attribute.

Return:	true	The item is visible.
	false	The item is invisible.

13.2.15 [event] on_triggered
This event is called if the menu item is selected.

This event has been introduced in version 0.21.

13.2.16 [const] string shortcut
Get the keyboard shortcut.

Return:	shortcut	The keyboard shortcut as a string.

13.2.17 shortcut=(shortcut)
Set the keyboard shortcut.

Input:	shortcut	The keyboard shortcut as string (i.e. 'Ctrl+C').

13.2.18 [const] string title
Get the title.

Return:	title	The current title as string.

13.2.19 title=(title)
Set the title.

Input:	title	The title to set as string.

13.2.20 trigger

Trigger the action programmatically.

13.2.21 triggered

This method is called if the menu item is selected.

13.2.22 visible=(boolean)

Show or hide.

Input:	true	Make the item visible.
	false	Make the item invisible.

13.3 Class ActionBase (version 0.21) An action.

Actions act as a generalization of menu entries. The action provides the appearance of a menu entry such as title, key shortcut etc. and dispatches the menu events. The action can be manipulated to change to appearance of a menu entry and can be attached an observer that receives the events when the menu item is selected.

Multiple action objects can in fact refer to the same action internally, in which case the information and event handler is copied between the incarnations.

Method Overview

triggered	This method is called if the menu item is selected.
on_triggered	This event is called if the menu item is selected.
title=	Set the title.
title	Get the title.
shortcut=	Set the keyboard shortcut.
shortcut	Get the keyboard shortcut.
is_checkable?	"is_checkable" attribute.
is_checked?	"is_checked" attribute.
is_enabled?	"is_enabled" attribute.
is_visible?	"is_visible" attribute.
checkable=	Make the item(s) check-able or not.
enabled=	Enable or disable the action.
visible=	Show or hide.
checked=	Check or uncheck.
icon=	Set the icon to the given picture.
icon_text=	Set the icon's text.
icon_text	Get the icon's text.
trigger	Trigger the action programmatic-ally.
assign	Assign the contents of another object to self.
dup	Creates a copy of self..
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.3.1 assign(ActionBase other) Assign the contents of another object to self.

This method assigns the contents of another object to self. This is a deep copy that does not only copy the reference but the actual content.

13.3.2 checkable=(boolean) Make the item(s) check able or not.

Input:	true	Make the item check able.
	false	Make the item not check able.

13.3.3 checked=(boolean)
Check or unchecked

Input:

true	Make the item checked.
false	Make the item unchecked.

13.3.4 destroyExplicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.3.5 [const] boolean destroyed
Tell, if the object was destroyed.

Return:

true	The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
false	The object is still alive.

13.3.6 [const] ActionBase dup
Creates a copy of self.

Return: ActionBase The copy of self.

13.3.7 enabled=(boolean)
Enable or disable the action.

Return:

true	Enable the item.
false	Disable the item.

13.3.8 icon=(file)
Set the icon to the given picture.

Input:

file	The image file to load as icon for the menu item. Passing an empty string will reset the icon.

13.3.9 [const] icon_text
Get the icon's text.

Input:

icon_text	The current icon text as string.

13.3.10 icon_text=(icon_text)
Set the icon's text.

Input:

icon_text	The icon text as string to be set below the icon. If no icon text is given the normal text will be used for the icon. Passing an empty string will reset the icon's text.

13.3.11 [const] boolean is_checkable?
"is_checkable" attribute.

Return:	true	The item is check able.
	false	The item is not check able.

13.3.12 [const] boolean is_checked?
"is_checked" attribute.

Return:	true	The item is checked.
	false	The item is unchecked.

13.3.13 [const] boolean is_enabled?
"is_enabled" attribute.

Return:	true	The item is enabled.
	false	The item is disabled.

13.3.14 [const] boolean is_visible?
"is_visible" attribute.

Return:	true	The item is visible.
	false	The item is invisible.

13.3.15 [event] on_triggered
This event is called if the menu item is selected.

This event has been introduced in version 0.21.

13.3.16 [const] string shortcut
Get the keyboard shortcut.

Return:	shortcut	The keyboard shortcut as a string.

13.3.17 shortcut=(shortcut)
Set the keyboard shortcut.

Input:	shortcut	The keyboard shortcut as string (i.e. 'Ctrl+C').

13.3.18 [const] string title
Get the title.

Return:	title	The current title as string.

13.3.19 title=(title)
Set the title.

Input:	title	The title to set as string.

13.3.20 trigger

Trigger the action programmatically.

13.3.21 triggered

This method is called if the menu item is selected.

13.3.22 visible=(boolean)

Show or hide.

Input:	true	Make the item visible.
	false	Make the item invisible.

13.4 Class Annotation (version 0.21) The annotation object.

This class implements an "annotation object".

Method Overview

new	Create a new ruler or marker with the default attributes.
p1	Get the first point of the ruler or marker.
p2	Get the second point of the ruler or marker.
p1=	Set the first point of the ruler or marker.
p2=	Set the second point of the ruler or marker.
box	Get the bounding box of the object (not including text).
transformed	Transform the ruler or marker with the given simple transformation.
transformed_cplx	Transform the ruler or marker with the given complex transformation.
transformed_cplx	Transform the ruler or marker with the given complex transformation.
fmt=	Set the format used for the label.
fmt	Returns the format used for the label.
fmt_x=	Set the format used for the x-axis label.
fmt_x	Returns the format used for the x-axis label.
fmt_y=	Set the format used for the y-axis label.
fmt_y	Returns the format used for the y-axis label.
style=	Set the style used for drawing the annotation object.
style	Returns the style of the annotation object.
style_...	Various style_... codes used by the style method.
style_ruler	style_ruler code.
style_arrow_end	style_arrow_end code.
style_arrow_start	style_arrow_start code.
style_arrow_both	style_arrow_both code.
style_line	style_line code.
outline=	Set the outline style used for drawing the annotation object.
outline	Returns the outline style of the annotation object.
outline_...	Various outline_... codes used by the outline method.
outline_diag	outline_diag code.
outline_xy	outline_xy code.
outline_diag_xy	outline_diag_xy.
outline_yx	outline_yx code.
outline_diag_yx	outline_diag_yx code.
outline_box	outline_box code.
snap=	Set the "snap to objects" attribute.
snap?	Return the "snap to objects" attribute.
angle_constraint=	Set the angle constraint attribute.
angle_constraint	Return the angle constraint attribute.
angle_...	Various angle_... codes used by the angle_constraint method.
angle_any	angle_any code.
angle_diagonal	angle_diagonal code.
angle_ortho	angle_ortho code.
angle_horizontal	angle_horizontal code.
angle_vertical	angle_vertical code.
angle_global	angle_global code.
text_x	Return the formatted text for the x-axis label.
text_y	Return the formatted text for the y-axis label.
text	Return the formatted text for the main label.

to_s	Returns the string representation of the ruler.
==	Equality operator.
!=	Inequality operator.
assign	Assign the contents of another object to self.
dup	Creates a copy of self.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.4.1 [const] boolean !=Annotation
Inequality operator.

Return:	true	The two types are unequal.
	false	The two types are equal.

13.4.2 [const] boolean ==Annotation
Equality operator.

Return:	true	The two types are equal.
	false	The two types are unequal.

13.4.3 [const] integer angle_constraint
Return the angle constraint attribute.

See angle_constraint= method for more detailed description.

13.4.4 angle_constraint=(flag)
Set the angle constraint attribute.

Input:	flag	The angle constraint attribute. This attribute controls if an angle constraint is applied when moving one of the ruler's points. The various angle_... values can be used for this purpose.

13.4.5 [static] integer angle_...
Various angle_... code used by the angle_constraint method.

13.4.5.1 [static] integer angle_any – angle_any code.

13.4.5.2 [static] integer angle_diagonal – angle_diagonal code.

13.4.5.3 [static] integer angle_global – angle_global code.

This code will tell the ruler or marker to use the angle constraint defined globally.

13.4.5.4 [static] integer angle_horizontal – angle_horizontal code.

13.4.5.5 [static] integer angle_ortho – angle_ortho code.

13.4.5.6 [static] integer angle_vertical – angle_vertical code.

13.4.6 assign(Annotation other)
Assign the contents of another object to self.

This method assigns the contents of another object to self. This is a deep copy that does not only copy the reference but the actual content.

13.4.7 [const] DBox box
Get the bounding box of the object (not including text).

Return: The bounding box

13.4.8 destroy
Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.4.9 destroyed
Tell, if the object was destroyed.

Return: true The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
false The object is still alive.

13.4.10 [const] Annotation dup
Creates a copy of self.

Return: Annotation The copy of self.

13.4.11 [const] string fmt
Get the format used for the label.

Return: format The format string.

13.4.12 fmt=(format)
Set the format used for the label.

Input: format The format string.

13.4.13 [const] string fmt_x
Get the format used for the x-axis label.

Return: format The format string.

13.4.14 fmt_x=(format)
Set the format used for the x-axis label.

X-axis labels are only used for styles that have a horizontal component.

Input: format

13.4.15 [const] string fmt_y
Get the format used for the y-axis label.

Return: format The format string.

13.4.16 fmt_y=(format)
Set the format used for the y-axis label.

Y-axis labels are only used for styles that have a vertical component.

Input: format

13.4.17 [static] Annotation new
Create a new ruler or marker with the default attributes.

13.4.18 [const] integer outline
Get the outline style of the annotation object.

Return: style The outline style as integer.

13.4.19 outline=(outline)
Set the outline style used for drawing the annotation object.

Input: outline The outline style used for drawing the annotation object. The **outline_...** values can be used for defining the annotation object's outline. The outline style determines what components are drawn.

13.4.20 [static] integer outline_...
Various outline_... code used by the angle_constraint method.

13.4.20.1 [static] integer outline_box – outline_box code.

13.4.20.2 [static] integer outline_diag – outline_diag code.

13.4.20.3 [static] integer outline_diag_xy – outline_diag_xy code.

13.4.20.4 [static] integer outline_diag_yx – outline_diag_yx code.

13.4.20.5 [static] integer outline_xy – outline_xy code.

13.4.20.6 [static] integer outline_yx – outline_yx code.

13.4.21 [const] const ref p1
Get the first point of the ruler or marker.

The points of the ruler or marker are always given in micron units in floating-point coordinates.

Return: point The first point.

13.4.22 p1=(DPoint point)
Set the first point of the ruler or marker.

The points of the ruler or marker are always given in micron units in floating-point coordinates.

Input: point The first point.

13.4.23 [const] const ref p2
Get the first point of the ruler or marker.

The points of the ruler or marker are always given in micron units in floating-point coordinates.

Return: point The second point.

13.4.24 p2=(DPoint point)
Set the first point of the ruler or marker.

The points of the ruler or marker are always given in micron units in floating-point coordinates.

Input: point The second point.

13.4.25 snap=(flag)
Set the "snap to objects" attribute.

Input: true The ruler or marker snaps to other objects when moved.
false The ruler or marker moves without any snap.

13.4.26 [const] boolean snap?
Get the "snap to objects" attribute.

Return: true|false The 'snap to objects' attribute status.

13.4.27 [const] integer style
Get the style of the annotation object.

Return: style The style of the annotation object as integer.

13.4.28 style=(style)
Set the style used for drawing the annotation object.

Input: style The style used for drawing the annotation object. The various **style_...** values can be used for defining the annotation object's style. The style determines if ticks or arrows are drawn.

13.4.29 [static] integer style_...

Various style_... code used by the angle_constraint method.

13.4.29.1 [static] integer style_arrow_both – style_arrow_both code.

13.4.29.2 [static] integer style_arrow_end – style_arrow_end code.

13.4.29.3 [static] integer style_arrow_start – style_arrow_start code.

13.4.29.4 [static] integer style_line – style_line code.

13.4.29.5 [static] integer style_ruler – style_ruler code.

13.4.30 [const] string text

Get the formatted text for the main label.

Return: string The formatted text for the main label.

13.4.31 [const] string text_x

Get the formatted text for the x-axis label.

Return: string The formatted text for the x-axis label.

13.4.32 [const] string text_y

Get the formatted text for the y-axis label.

Return: string The formatted text for the y-axis label.

13.4.33 [const] string to_s

Get the string representation of the ruler.

This method was introduced in version 0.19.

Return: string The string representation of the ruler.

13.4.34 [const] Annotation transformed(DTrans t)

Transform the ruler or marker with the given simple transformation.

Input: t The simple transformation to apply.
Return: Annotation The transformed object.

13.4.35 [const] Annotation transformed_cplx(DCplxTrans t)

Transform the ruler or marker with the given complex transformation.

Input: t The complex transformation to apply.
Return: Annotation The transformed object.

13.4.36 [const] Annotation transformed_cplx(ICplxTrans t)

Transform the ruler or marker with the given complex transformation.

Input:	t	The complex transformation to apply.
Return:	Annotation	The transformed object (in this case an integer coordinate object).

13.5 Class Application (version 0.21) The application object.

The application object is the main port from which to access all the internals of the application, in particular the main window.

Method Overview

instance	Return the singleton instance of the application.
version	Return the application's version string.
inst_path	Return the application's installation path (where the executable is located).
write_config	Write configuration to a file.
read_config	Read the configuration from a file.
get_config_names	Query all valid configuration parameter names.
get_config	Query the value of a valid configuration parameter.
set_config	Set a configuration parameter with the given name to the given value.
is_editable?	Return true if the application is in editable mode.
main_window	Return a reference to the main window.
exec	Execute the application's main loop.
process_events	Process pending events.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.5.1 destroy Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

Return: singleton — The singleton instance of the application. Does the same as instance, if entered in RBA console.

Console Input 13.1:

```
> RBA::Application.instance.destroy
#<RBA::Application:0x7f39c58f67a8>
```

13.5.2 [const] boolean destroyed Tell, if the object was destroyed.

Return: true — The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.

false — The object is still alive

Console Input 13.2:

```
> RBA::Application.instance.destroyed
false
```

13.5.3 integer exec
Execute the application's main loop.

This method must be called in order to execute the application in the main script if a script is provided.

Return:

Comment: Returns "-1" if entered in RBA Console. What's the meaning? Are there other return codes?

Console Input 13.3: Call exec from RBA Console

```
> RBA::Application.instance.exec
-1
```

13.5.4 [const] string get_config(name)
Query a configuration parameter.

This method returns the **value** of the given configuration parameter **name**. If the parameter is not known, an exception will be thrown. Use get_config_names to obtain a list of all configuration parameter names available.

Configuration parameters are always stored as strings. The actual format of this string is specific to the configuration parameter. The values delivered by this method correspond to the values stored in the configuration file.

Input:	name	The name as string of the configuration parameter whose value shall be obtained.
Return:	value	The value of the parameter.

Console Input 13.4: Query valid configuration parameter

```
> RBA::Application.instance.get_config("grid-micron")
10
```

Console Input 13.5: Query invalid configuration parameter

```
RBA::Application.instance.get_config("grid-micro")
```

13.5.5 [const] string[ ] get_config_names
Query the configuration parameter names.

This method returns a list of valid configuration parameter. **Comment: For better reading and probably handling the names should be listed with usual delimiter (comma, space ?).** These names can be used to get and set configuration parameter values.

Return:	string[]	A vector string containing all valid configuration parameter names.

Console Input 13.6: Query the configuration parameter names

```
> RBA::Application.instance.get_config_names
grid-micronsynchronized-viewsdefault-gridsdbumrutechnologiesreader-enable-text-o\
bjectsreader-enable-propertiesshow-navigatornavigator-show-all-hier-levelsnaviga\
tor-show-imagesshow-toolbarshow-layer-toolboxshow-hierarchy-panelshow-layer-pane\
lwindow-statewindow-geometrykey-bindingstip-window-hiddendigits-microndigits-dbu\
reader-layer-mapreader-create-other-layersreader-enable-text-objectsreader-enabl\
e-propertiespcb-import-specstream-import-specedit-modedefault-layer-propertiesde\
fault-add-other-layerslayers-always-show-ldlayers-always-show-layout-indextest-s\
hapes-in-viewflat-cell-listcell-list-sortinghide-empty-layersmin-inst-label-size\
inst-label-fontinst-label-transforminst-colorinst-visibletext-colortext-visiblet\
ext-lazy-renderingshow-propertiesapply-text-transdefault-text-sizetext-fontsel-c\
olorsel-line-widthsel-vertex-sizesel-dither-patternsel-halosel-transient-modebac\
kground-colorcontext-colorcontext-dimmingcontext-hollowchild-context-colorchild-\
context-dimmingchild-context-hollowchild-context-enabledabstract-mode-widthabstr\
act-mode-enabledfit-new-cellfull-hierarchy-new-cellinitial-hier-depthclear-ruler\
-new-cellmouse-wheel-modepan-distanceabsolute-unitsdbu-unitsdrawing-workersdrop-\
small-cellsdrop-small-cells-conditiondrop-small-cells-valuedraw-array-border-ins\
tancesbitmap-oversamplingcolor-palettestipple-palettestipple-offsetno-stipplegri\
d-colorgrid-style0grid-style1grid-style2grid-visiblegrid-show-rulerrulersruler-s\
nap-rangeruler-colorruler-haloruler-snap-moderuler-obj-snapruler-grid-snapruler-\
templatescurrent-ruler-templateedit-text-stringedit-text-sizeedit-text-halignedi\
t-text-valignedit-path-widthedit-path-ext-typeedit-path-ext-var-beginedit-path-e\
xt-var-endedit-inst-cell-nameedit-inst-angleedit-inst-mirroredit-inst-arrayedit-\
inst-scaleedit-inst-rowsedit-inst-row_xedit-inst-row_yedit-inst-columnsedit-inst\
-column_xedit-inst-column_yedit-inst-place-originedit-max-shapes-of-instancesedi\
t-show-shapes-of-instancesedit-top-level-selectionedit-gridedit-snap-to-objectse\
dit-move-angle-modeedit-connect-angle-modeoasis-compressiongds2-box-record-modeg\
ds2-allow-big-recordsgds2-allow-multi-xy-boundariesgds2-multi-xy-recordsgds2-max\
-vertex-countgds2-max-cellname-lengthgds2-libnamecif-wire-modecif-dbudxf-dbudxf-\
unitdxf-polyline-modedxf-circle-pointsdxf-polygon-modeshb-context-cellshb-contex\
t-modeshb-window-modeshb-window-stateshb-window-dimshb-max-inst-countshb-max-sha\
pe-countgds2-multi-xy-recordsgds2-max-vertex-countgds2-max-cellname-lengthgds2-l\
ibnamecib-context-cellcib-context-modecib-window-modecib-window-statecib-window-\
dimcib-max-inst-countrdb-context-moderdb-window-moderdb-window-staterdb-window-d\
imrdb-max-marker-countrdb-marker-colorrdb-marker-line-widthrdb-marker-vertex-siz\
erdb-marker-halordb-marker-dither-patternnt-window-modent-window-dimnt-max-shape\
s-highlightednt-marker-colornt-marker-line-widthnt-marker-vertex-sizent-marker-h\
alont-marker-dither-patternnt-marker-intensity
```

13.5.6 [const] string inst_path
Query the application's installation path (where the executable is located).

This method has been added in version 0.18.

Return: inst_path — The application's installation path or the value of environment variable $KLAYOUT_PATH, if set.

Console Input 13.7: Query the application's installation path

```
> RBA::Application.instance.inst_path
/home/peter/.klayout
```

13.5.7 [static] ref Application instance
Return the singleton instance of the application.

There is exactly one instance of the application. This instance can be obtained with this method.

Return: singleton — Returns singleton instance of the application.

Console Input 13.8: Return the singleton instance of the application

```
> RBA::Application.instance
#<RBA::Application:0x7f39c58f9e08>
```

13.5.8 [const] boolean is_editable?
Query the edit mode of the application.

Return: true — Edit mode.
false — Viewer mode.

Console Input 13.9:

```
> RBA::Application.instance.is_editable?
true
```

13.5.9 [const] ref MainWindow main_window
Query a reference of the main window.

Return: singleton — Returns an object reference to the main window object.

Console Input 13.10: Query a reference of the main window

```
> RBA::Application.instance.main_window
#<RBA::MainWindow:0x7f39c591e500>
```

13.5.10 process_events
Process pending events.

This method processes pending events and dispatches them internally. Calling this method periodically during a long operation keeps the application "alive".

Console Input 13.11:

```
> RBA::Application.instance.process_events
```

13.5.11 boolean read_config(file_name)
Read the configuration from a file.

This method slightly does nothing, if the config file does not exist. If it does and an error occurred, the error message is printed on stderr. In both cases, false is returned.

Return: true — Config read from given file.
false — Config not read from given file.

Console Input 13.12: file *klayout-configuration* exists and is readable

```
> RBA::Application.instance.read_config("klayout-configuration")
true
```

Console Input 13.13: file *klayout-config* does not exists

```
> RBA::Application.instance.read_config("klayout-config")
false
```

Console Input 13.14: file *klayout-configuration* exists, but is not readable

```
> RBA::Application.instance.read_config("klayout-configuration")
Ruby error: '(eval):0:in `read_config': Problem reading config file klayout-conf\
iguration: XML parser error: unexpected end of file in line 1, column 1' (Runti\
meError)
  (eval)
  (eval):0
```

13.5.12 set_config(name, value)
Set a configuration parameter with the given name to the given value.

This method sets the configuration parameter with the given **name** to the given **value**. Values can only be strings. Numerical values have to be converted into strings first. The actual format of the value depends on the configuration parameter. The name must be one of the names returned by get_config_names. There is no return in any case, even if the name of the configuration parameter is misspelled.

Input: name — The name as string of the configuration parameter to be set.
value — The value to which the configuration parameter to be set.

Console Input 13.15: Set a configuration parameter with the given name to the given value

```
> RBA::Application.instance.set_config("grid-micron","10")
```

13.5.13 [const] version
Query the application's version string.

Return: version — Returns the application's version string.

Console Input 13.16: Query the application's version string

```
> RBA::Application.instance.version
KLayout 0.21.14
```

13.5.14 boolean write_config(file_name)
Write configuration to a file.

If the configuration file cannot be written, **false** is returned but no exception is thrown.

Return: true — Config successfully written to given file.
false — Write config to given file fails.

Console Input 13.17: file *klayout-configuration* does not exists, or exists and is write able

```
> RBA::Application.instance.write_config("klayout-configuration")
true
```

Console Input 13.18: file *klayout-configuration* is set to read only

```
> RBA::Application.instance.write_config("klayout-configuration")
false
```

13.6 Class ArgType (version 0.21) The description of a type.

The description of a type (argument or return value).

Method Overview

type	Get the basic type.
t_...	Various t_... constants.
t_void	Type void.
t_bool	Type boolean.
t_int	Type integer.
t_uint	Type unsigned integer.
t_long	Type long integer.
t_ulong	Type unsigned long integer.
t_longlong	Type long long integer.
t_double	Type floating point.
t_string_ccptr	Type string ??.
t_string	Type string.
t_var	Type variable.
t_object_ref	Type object reference.
t_object_cref	Type object constant reference.
t_object_new	Type object new.
t_object	Type object.
is_vector?	Query if the type is a vector of the basic type.
is_ref?	Query if the type is a reference to the given type.
is_iter?	Query if the return value is an iterator rendering the given type.
cls	Specifies the class for t_object...types.
to_s	Convert to a string.
==	Equality of two types.
!=	Inequality of two types.
assign	Assign the contents of another object to self.
dup	Creates a copy of self.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.6.1 [const] boolean !=(ArgType) Inequality test of two types.

Return:

true	The two types are unequal.
false	The two types are equal.

13.6.2 [const] boolean ==(ArgType) Equality test of two types.

Return:

true	The two types are equal.
false	The two types are unequal.

13.6.3 assign(ArgType other)
Assign the contents of another object to self.

This method assigns the contents of another object to self. This is a deep copy that does not only copy the reference but the actual content.

Input: other The other object.

13.6.4 [const] const ref Class cls
Specifies the class for t_object_... types.

13.6.5 destroy
Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.6.6 [const] boolean destroyed
Tell, if the object was destroyed.

Return: true The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
false The object is still alive.

13.6.7 [const] ArgType dup
Creates a copy of self.

Return: ArgType The copy of self.

13.6.8 [const] boolean is_iter?
Query if the return value is an iterator rendering the given type (Return value only).

Return: true The return value is an iterator rendering the given type. (Return value only.)
false The return value is no iterator.

13.6.9 [const] boolean is_ref?
Query if the type is a reference to the given type.

Return: true The type is a reference to the given object.
false The type is not a reference to the given object.

13.6.10 [const] boolean is_vector?
Query if the type is a vector of the basic type.

Return: true The type is a vector of the basic type.
false The type is not a vector.

13.6.11 [const] integer type

Return the basic type (see various t_... constants).

13.6.12 [const] integer t_...

Various t_... constants).

13.6.12.1 [static] integer t_bool – Type boolean constant.

13.6.12.2 [static] integer t_double – Type floating point constant.

13.6.12.3 [static] integer t_int – Type integer constant.

13.6.12.4 [static] integer t_long – Type long integer constant.

13.6.12.5 [static] integer t_longlong – Type long long integer constant.

13.6.12.6 [static] integer t_object – Type object constant.

13.6.12.7 [static] integer t_object_cref – Type object constant reference constant.

13.6.12.8 [static] integer t_object_new – Type object new constant.

13.6.12.9 [static] integer t_object_ref – Type object reference constant.

13.6.12.10 [static] integer t_string – Type string constant.

13.6.12.11 [static] integer t_string_ccptr – Type string constant. Comment: ????

13.6.12.12 [static] integer t_uint – Type unsigned integer constant.

13.6.12.13 [static] integer t_ulong – Type unsigned long integer constant.

13.6.12.14 [static] integer t_var – Type variable constant.

13.6.12.15 [static] integer t_void – Type void constant.

13.6.13 [static] string to_s

Convert to a string constant.

Return: string The constant converted to a string.

13.7 Class Box (version 0.21)
A box class with integer coordinates.

This object represents a box (a rectangular shape).

The notation is: p1 is the lower left point (x1, y1), p2 the upper right one (x2, y2), compare with fig. 13.1.

A box can be empty. An empty box represents no area (not even a point).

A box can be a point or a single line. In this case, the area is zero but the box still can overlap other boxes.

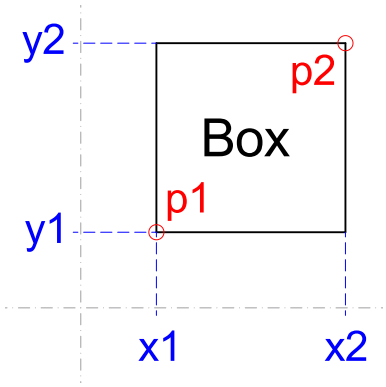

Figure 13.1. Box notation.

Method Overview

from_dbox	Construct an integer box from a floating-point coordinate box.
new	Default constructor: creates an empty (invalid) box.
new	Constructor with four coordinates.
new	Constructor with two points.
p1	Get the lower left point of the box.
p2	Get the upper right point of the box.
center	Get the center of the box.
left	Get the left coordinate of the box.
right	Get the right coordinate of the box.
bottom	Get the bottom coordinate of the box.
top	Get the top coordinate of the box.
width	Get the width of the box.
height	Get the height of the box.
left=	Set the left coordinate of the box.
right=	Set the right coordinate of the box.
bottom=	Set the bottom coordinate of the box.
top=	Set the top coordinate of the box.
p1=	Set the lower left point of the box.
p2=	Set the upper right point of the box.
contains?	Test if a point is inside the box.
empty?	Test if this box is of type empty box.
inside?	Test if this box is inside the argument box.
touches?	Test if this box touches the argument box.
overlaps?	Test if this box overlaps the argument box.
area	Compute the box area
is_point?	Test if the box is a single point
+	Join a box with a point.
+	Joining of two boxes.

&	Intersection of two boxes.
*****	Convolve two boxes.
move	Moves the box by a certain distance.
moved	Get the box moved by a certain distance.
enlarge	Enlarges the box by a certain amount.
enlarged	Get the box enlarged by a certain amount.
transformed	Transform the box with the given simple transformation
transformed_cplx	Transform the box with the given complex transformation
transformed_cplx	Transform the box with the given complex transformation
<	Less operator.
==	Equality operator.
!=	Inequality operator.
to_s	Convert to a string.
assign	Assign the contents of another object to self.
dup	Creates a copy of self.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.7.1 [const] boolean !=(Box box)
Inequality test of two boxes.

Test if this box and the given box are not equal.

Input:	box	The given box.
Return:	true	This and the given box are unequal.
	false	This and the given box are equal.

13.7.2 Box &(Box box)
Intersection of two boxes.

The intersection of two boxes is the largest box common to both boxes. The intersection may be empty if both boxes to not touch. If the boxes do not overlap but touch the result may be a single line or point with an area of zero. Overwrites this box with the result.

Input:	box	The box to take the intersection with.
Return:	Box	The intersection box.

13.7.3 Box *(Box box)
Convolve two boxes.

The * operator convolve the first box with the one given as the second argument. The box resulting from "convolution" is the outer boundary of the union set formed by placing the second box at every point of the first. In other words, the returned box of (p1,p2)*(q1,q2) is (p1+q1,p2+q2).

Input:	box	The given box.
Return:	Box	The intersection box.

13.7.4 Con Box +(Point point)
Join a box with a point.

The + operator joins a point with the box. The resulting box will enclose both the original box and the point.

Input:	point	The point to join with this box.
Return:	Box	The box joined with the point.

13.7.5 Box +(Box box)
Joining of two boxes.

The + operator joins the first box with the one given as the second argument. Joining constructs a box that encloses both boxes given. Empty boxes are neutral: they do not change another box when joining. Overwrites this box with the result.

Input:	box	The box to join with this box.
Return:	Box	The joined box.

13.7.6 [const] boolean <(Box box)
Less operator.

Input:	box	This box.
Return:	true	This box is 'less' with respect to first and second point (in this order).
	false	This box is 'greater'.

13.7.7 [const] boolean ==(Box box)
Equality operator.

Input:	box	This box.
Return:	true	This box and the given box are equal.
	false	This box and the given box are unequal.

13.7.8 [const] double area
Compute the box area.

Return:	double integer	The box area, or
	0	the box is empty.

13.7.9 assign(Box other)
Assign the contents of another object to self.

This method assigns the contents of another object to self. This is a deep copy that does not only copy the reference but the actual content.

Input:	other	The contents of another object.

13.7.10 bottom=(y1)
Set the bottom coordinate of the box.

Input:	y1	The bottom coordinate of the box.

13.7.11 [const] y1 bottom
Query the bottom coordinate of the box.

Return:	y1	The bottom coordinate of the box.

13.7.12 [const] Point center
Query the center of the box.

Return: Point — The center coordinate of the box.

13.7.13 [const] boolean contains?(Point point)
Tests if a point is inside the box.

Input: point — The coordinate to be tested.
Return: true — The point is placed inside the box or on the box contour.
false — The point is placed completely outside the box.

13.7.14 destroy
Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.7.15 [const] boolean destroyed
Tell, if the object was destroyed.

Return: true — The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
false — The object is still alive.

13.7.16 [const] Box dup
Creates a copy of self.

Return: Box — The copy of self.

13.7.17 [const] boolean empty?
Test if the box is of type empty box.

An empty box may be created with the default constructor for example. Such a box is neutral when combining it with other boxes and renders empty boxes if used in box intersections and false in geometrical relationship tests.

Return: true — The box is empty.
false — The box is not empty.

13.7.18 ref Box enlarge(Point enlargement)
Enlarges the box by a certain amount.

Enlarges the box by x and y value specified in the vector passed. Positive values with grow the box, negative ones will shrink the box. The result may be an empty box if the box disappears. The amount specifies the grow or shrink per edge. The width and height will change by twice the amount. Does not check for coordinate overflows.

Input: enlargement — The grow or shrink amount in x and y direction.
Return: ref — A reference to the enlarged box.

13.7.19 [const] Box enlarged(Point enlargement)
Get the box enlarged by a certain amount.

Enlarges the box by **x** and **y** value specified in the vector passed. Positive values will grow the box, negative ones will shrink the box. The result may be an empty box if the box disappears. The amount specifies the grow or shrink per edge. The width and height will change by twice the amount. Does not modify this box. Does not check for coordinate overflows.

Input: enlargement — The grow or shrink amount in x and y direction.
Return: Box — The enlarged box.

13.7.20 [static] Box from_dbox(DBox double_box)
Construct an integer box from a floating-point coordinate box.

Create a integer coordinate box from a floating-point coordinate box.

Input: double_box — The floating-point coordinate box.
Return: Box — The integer coordinate box.

13.7.21 [const] height height
Query the height of the box.

Return: height — The height of the box, where the equation $height = y2 - y1$ is valid.

13.7.22 [const] boolean inside?(Box box)
Test if this box is inside the argument box.

Input: box — The given box.
Return: true — This box is inside the given box, i.e. the box intersection renders this box.
false — This box is not inside the given box.

13.7.23 [const] boolean is_point?
Test if the box is a single point.

Return: true — The box is a single point.
false — The box is not a single point.

13.7.24 left=(x1)
Set the left coordinate of the box.

Input: x1 — The left coordinate of the box.

13.7.25 [const] x1 left
Query the left coordinate of the box.

Return: x1 — The left coordinate of the box.

13.7.26 ref Box move(Point distance)
Moves the box by a certain distance.

Moves the box by a given offset and returns the moved box. Does not check for coordinate overflows.

Input:	distance	The offset to move the box.
Return:	ref	A reference to this box.

13.7.27 [const] Box moved(Point distance)
Get the box moved by a certain distance.

Moves the box by a given offset and returns the moved box. Does not modify this box. Does not check for coordinate overflows.

Input:	distance	The offset to move the box.
Return:	Box	The moved box.

13.7.28 [static] Box new
Default constructor: creates an empty (invalid) box.

Return:	Box	The new empty box.

13.7.29 [static] Box new(left, bottom, right, top)
Constructor with four coordinates.

Synonym for [static] Box new_lbrt(left, bottom, right, top)

Four coordinates are given to create a new box. If the coordinates are not provided in the correct order (i.e. right <left), these are swapped.

Input:	left, bottom, right, top	Four coordinates given to create a new box, where left equals to x1, bottom to y1, right to x2 and top to y2.
Return:	Box	The new box.

13.7.30 [static] Box new(Point lower_left, Point upper_right)
Box constructor with two points.

Synonym for [static] Box new_pp(Point lower_left, Point upper_right).

Two points are given to create a new box. If the coordinates are not provided in the correct order (i.e. right <left), these are swapped.

Input:	lower_left, upper_right	Two points given to create a new box.
Return:	Box	The new box.

13.7.31 [const] boolean overlaps?(Box box)
Test if this box overlaps the argument box.

Input:	box	The argument box.
Return:	true	The intersection box of this box with the argument box exists and has a non-vanishing area.
	false	The intersection box of this box with the argument box does not exists or has a vanishing area.

13.7.32 [const] ref Point p1

Query the lower left point of the box.

Return: lower_left The lower left point of the box, where lower_left equals to x1, y1.

13.7.33 p1=(Point lower_left)

Set the lower left point of the box.

Input: lower_left The lower left point of the box, where lower_left equals to x1, y1.

13.7.34 [const] ref Point p2

Query the upper right point of the box.

Return: upper_right The upper right point of the box, where upper_right equals to x2, y2.

13.7.35 p2=(Point upper_right)

Set the upper right point of the box.

Input: upper_right The upper right point of the box, where upper_right equals to x2, y2.

13.7.36 [const] x2 right

Query the right coordinate of the box.

Return: x2 The right coordinate of the box.

13.7.37 right=(x2)

Set the right coordinate of the box.

Input: x2 The right coordinate of the box.

13.7.38 [const] string to_s

Convert a value to a string.

Return: string The converted value as string.

13.7.39 [const] y2 top

Query the top coordinate of the box.

Return: y2 The top coordinate of the box.

13.7.40 top=(y2)

Set the top coordinate of the box.

Input: y2 The top coordinate of the box.

13.7.41 [const] boolean touches?(Box box)
Test if this box touches the argument box.

Input:	box	The argument box
Return:	true	This box has at least one point common with the argument box.
	false	This box has none point common with the argument box.

13.7.42 [const] Box transformed(Trans t)
Transform the box with the given simple transformation.

Input:	t	The simple transformation to apply.
Return:	Box	The transformed box.

13.7.43 [const] DBox transformed_cplx(CplxTrans t)
Transform the box with the given complex transformation.

Input:	t	The complex transformation to apply.
Return:	DBox	The transformed box (a DBox now).

13.7.44 [const] Box transformed_cplx(ICplxTrans t)
Transform the box with the given complex transformation.

This method has been introduced in version 0.18.

Input:	t	The complex transformation to apply.
Return:	Box	The transformed box (in this case an integer coordinate box).

13.7.45 [const] integer width
Query the width of the box.

Return:	width	The width of the box, where width equals to x2 - x1.

13.8 Class BrowserDialog (version 0.21) The HTML browser dialog.

The HTML browser dialog, see section 12.1: Using the HTML browser dialog I: A location browser and section 12.2: Using the HTML browser dialog II: A screen-shot gallery, Using the HTML Browser Dialog I and II, respectively. The HTML browser displays HTML code in a browser panel. It receives the code by retrieving it from a given URL.

URL's with the special scheme "int" are retrieved from a BrowserSource object. This will act as a kind of server for these URL's.

Method Overview

hide	Hide the HTML browser window.
show	Show the HTML browser window in a non-modal way.
exec	Execute the HTML browser dialog as a modal window.
load	Load the given URL into the browser dialog.
set_source	Connect to a source object.
set_size	Set the size of the dialog window.
set_caption	Set the caption of the window.
reload	Reload the current page.
set_home	Set the browser's initial and current URL which is selected if the "home" location is chosen.
closed	Callback when the dialog is closed.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.8.1 closed
Callback when the dialog is closed.

This callback can be reimplemented to implement cleanup functionality when the dialog is closed.

13.8.2 destroy
Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.8.3 [const] boolean destroyed
Tell, if the object was destroyed.

Return: true — The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.

false — The object is still alive.

13.8.4 integer exec

Execute the HTML browser dialog as a modal window.

13.8.5 hide

Hide the HTML browser window.

13.8.6 load(string)

Load the given URL into the browser dialog.

Input: string The given URL.

13.8.7 reload

Reload the current page.

13.8.8 set_caption(caption)

Set the caption of the window.

Input: caption The caption of the window.

13.8.9 set_home(home_url)

Set the browser's initial and current URL which is selected if the "home" location is chosen.

Input: home_url The browser's initial and current URL.

13.8.10 set_size(width, height)

Set the size of the dialog window.

Input: width, height The dialog window width and height as integer.

13.8.11 set_source(ref BrowserSource source)

Connect to a source object.

Input: source The source object.

Caution: **This will use the object as the source but not hold a reference to that object. In order not to loose the source object (i.e. in RBA), a separate reference is required.**

13.8.12 show

Show the HTML browser window in a non-modal way.

13.9 Class BrowserSource (version 0.21) The BrowserDialog source for "int" URL's.

The BrowserDialog source for "int" URL's, see the examples given in section 12.1: Using the HTML browser dialog I: A location browser and section 12.2: Using the HTML browser dialog II: A screen-shot gallery.

The source object basically acts as a "server" for special URL's using "int" as the scheme. Classes that want to implement such functionality must derive from BrowserSource and re-implement the get method. This method is supposed to deliver a HTML page for the given URL.

Alternatively to implementing this functionality, a source object may be instantiated using the new_html constructor. This will create a source object that simply displays the given string as the initial and only page.

Method Overview

new_html	Construct a BrowserSource object with a default HTML string.
get	Get the HTML code for a given "int" URL.
assign	Assign the contents of another object to self.
dup	Creates a copy of self.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.9.1 assign(BrowserSource other) Assign the contents of another object to self.

This method assigns the contents of another object to self. This is a deep copy that does not only copy the reference but the actual content.

13.9.2 destroy Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.9.3 [const] boolean destroyed Tell, if the object was destroyed.

Return:	true	The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
	false	The object is still alive.

13.9.4 [const] BrowserSource dup Creates a copy of self.

Return:	BrowserSource	The copy of self.

13.9.5 string get(url)
Get the HTML code for a given "int" URL.

Input:	url	The HTML code for a given "int" URL.
Return:	empty	The browser will not be set to a new location. This allows to implement any functionality behind such links.
	content	The content of this string is displayed in the HTML browser page.

13.9.6 [static] BrowserSource new_html(string)
Construct a BrowserSource object with a default HTML string.

The default HTML string is sent when no specific implementation is provided.

Input:	string	The default HTML string.

13.10 Class Cell (version 0.21) The cell object.

A cell object consists of a set of shape containers (called layers), a set of child cell instances and auxiliary information such as the parent instance list. A cell is identified through an index given to the cell upon instantiation. Cell instances refer to single instances or array instances. Both are encapsulated in the same object, the CellInstArray object. In the simple case, this object refers to a single instance. In the general case, this object may refer to a regular array of cell instances as well.

Starting from version 0.16, the child_inst and erase_inst methods are no longer available since they were using index addressing which is no longer supported. Instead, instances are now addressed with the Instance reference objects.

Method Overview

shapes	Return the shapes list of the given layer.
clear_shapes	Clear all shapes in the cell.
clear_insts	Clear the instance list.
erase	Erase the instance given by the Instance object.
swap	Swap the layers given.
move	Move the shapes from the source to the target layer.
copy	Copy the shapes from the source to the target layer.
clear	Clear the shapes on the given layer.
replace_prop_id	Replace (or install) the properties of a cell.
transform	Transform the instance given by the instance with the given transformation.
transform	Transform the instance given by the instance with the given complex transformation.
replace	Replace a cell instance (array) with a different one.
replace	Replace a cell instance (array) with a different one with properties.
insert	Insert a cell instance given by another reference.
insert	Insert a cell instance (array).
insert	Insert a cell instance (array) with properties.
cell_index	The cell index accessor method.
child_instances	Number of child instances.
caller_cells	Return a list of all caller cells.
called_cells	Return a list of all called cells.
bbox	Retrieve the bounding box of the cell.
bbox_per_layer	Retrieve the per-layer bounding box of the cell.
each_overlapping_inst	Region query for the instances in “overlapping” mode.
each_touching_inst	Region query for the instances in “touching” mode.
each_child_cell	Iterate over all child cells.
child_cells	Report the number of child cells.
each_inst	Iterate over all child instances (which may actually be instance arrays).
each_parent_inst	Iterate over the parent instance list (which may actually be instance arrays).
parent_cells	Report the number of parent cells.
each_parent_cell	Iterate over all parent cells.
is_top?	Tell if the cell is a top-level cell.
is_leaf?	Tell if the cell is a leaf cell.
is_valid?	Test if the given Instance object is still pointing to a valid object.
each_shape	Iterate all shapes of a given layer.
each_shape	Iterate all shapes of a given layer.
each_touching_shape	Iterate all shapes of a given layer that touch the given box.
each_touching_shape	Iterate all shapes of a given layer that touch the given box.
each_overlapping_shape	Iterate all shapes of a given layer that overlap the given box.

each_overlapping_shape	Iterate all shapes of a given layer that overlap the given box.
hierarchy_levels	Return the number of hierarchy levels below (expensive).
is_empty?	Returns a value indicating whether the cell is empty.
is_ghost_cell?	Returns a value indicating whether the cell is a "ghost cell".
ghost_cell=	Sets the "ghost cell" flag.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.10.1 [const] const ref Box bbox
Retrieve the bounding box of the cell.

Return: Box — The bounding box of the cell.

13.10.2 [const] const ref Box bbox_per_layer(unsigned int layer_index)
Retrieve the per-layer bounding box of the cell.

Return: Box — The bounding box of the cell considering only the given layer.

13.10.3 [const] integer[] called_cells
Return a list of all called cells.

This method determines all cells which are called either directly or indirectly by the cell.

This method has been introduced in version 0.19.

Return: integer[] — A list of cell indices.

13.10.4 [const] integer[] caller_cells
Return a list of all caller cells.

This method determines all cells which call this cell either directly or indirectly.

This method has been introduced in version 0.19.

Return: integer[] — A list of cell indices.

13.10.5 [const] integer cell_index
The cell index accessor method.

Return: unsigned int — The cell index of the cell.

13.10.6 [const] integer child_cells
Report the number of child cells.

Return: integer — The number of child cells (not child instances!).
CAUTION: This method is SLOW!

13.10.7 [const] integer child_instances
Number of child instances.

Return: integer — Returns the number of cell instances.

13.10.8 clear(integer)
Clear the shapes on the given layer.

Input: integer — The layer index.

13.10.9 clear_insts
Clear the instance list.

13.10.10 clear_shapes
Clear all shapes in the cell.

13.10.11 copy(src, dest)
Copy the shapes from the source to the target layer.

The target layer is not overwritten. Instead, the shapes are added to the shapes of the target layer. If source and target layer are identical, this method does nothing.

This method has been introduced in version 0.19.

Input: src — The layer index of the source layer.
dest — The layer index of the destination layer.

13.10.12 destroy
Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.10.13 [const] boolean destroyed
Tell, if the object was destroyed.

Return: true — The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
false — The object is still alive.

13.10.14 yield integer each_child_cell
Iterate over all child cells.

Return: integer — The child cell indices, not every instance.

13.10.15 yield Instance each_inst
Iterate over all child instances (which may actually be instance arrays).

Starting with version 0.15, this iterator delivers Instance objects, rather than CellInstArray objects.

Return: Instance — The delivered objects as yield.

13.10.16 yield Instance each_overlapping_inst(Box b)
Region query for the instances in "overlapping" mode.

This will iterate over all child cell instances overlapping with the given region box.

Starting with version 0.15, this iterator delivers Instance objects, rather than CellInstArray objects.

Input:	b	The given region box.
Return:	Instance	The delivered objects as yield.

13.10.17 yield Shape each_overlapping_shape(integer Box b)
Iterate all shapes of a given layer that overlap the given box.

This call is equivalent to each_overlapping_shape(layer_index,box,RBA::Shapes::s_all). This convenience method has been introduced in version 0.16.

Input:	b	The region to query the shapes.
	integer	The layer on which to run the query.
Return:	Shape	The delivered objects as yield.

13.10.18 [const] yield Shape each_overlapping_shape(layer_index, Box box, flags)
Iterate all shapes of a given layer that overlap the given box.

Input:	flags	An "or"-ed combination of the **s_...** constants of the Shape class.
	box	The box by which to query the shapes.
	layer_index	The layer on which to run the query.
Return:	Shape	The delivered objects as yield.

13.10.19 [const] yield integer[] each_parent_cell
Iterate over all parent cells.

This iterator will iterate over the parent cells, just returning their cell index.

Return:	integer[]	The cell indexes.

13.10.20 yield ParentInstArray each_parent_inst
Iterate over the parent instance list (which may actually be instance arrays).

Return:	ParentInstArray	The parent instance list.

13.10.21 [const] yield Shape each_shape(integer)
Iterate all shapes of a given layer.

This call is equivalent to each_shape(layer_index,RBA::Shapes::s_all). This convenience method has been introduced in version 0.16.

Input:	integer	The layer on which to run the query.
Return:	Shape	The delivered objects as yield.

13.10.22 [const] yield Shape each_shape(layer_index, flags)
Iterate all shapes of a given layer.

This call is equivalent to each_shape(layer_index,RBA::Shapes::s_all). This convenience method has been introduced in version 0.16.

Input:	layer_index	The layer on which to run the query.
	flags	An "or"-ed combination of the s_.. constants of the Shapes class.
Return:	Shape	The delivered objects as yield.

13.10.23 yield Instance each_touching_inst(Box b)
Region query for the instances in "touching" mode.

This will iterate over all child cell instances touching the given region b.

Starting with version 0.15, this iterator delivers Instance objects, rather than CellInstArray objects.

Input:	b	The region to query.
Return:	Instance	The delivered objects as yield.

13.10.24 [const] yield Shape each_touching_shape(layer_index, Box b)
Iterate all shapes of a given layer that touch the given box.

This call is equivalent to each_touching_shape(layer_index,box,RBA::Shapes::s_all). This convenience method has been introduced in version 0.16.

Input:	b	The region to query.
	layer_index	The layer on which to run the query.
Return:	Shape	The delivered objects as yield.

13.10.25 [const] yield Shape each_touching_shape(layer_index, Box b, flags)
Iterate all shapes of a given layer that touch the given box.

Input:	flags	An "or"-ed combination of the **s_...** constants of the Shapes class.
	box	The box by which to query the shapes.
	layer_index	The layer on which to run the query.
Return:	Shape	The delivered objects as yield.

13.10.26 erase(Instance inst)
Erase the instance given by the Instance object.

This method has been introduced in version 0.16. It can only be used in editable mode.

Input:	inst	The instance object to be erased..

13.10.27 ghost_cell=(boolean)
Sets the "ghost cell" flag.

See is_ghost_cell? for a description of this property.

This method has been introduced in version 0.20.

Input:	boolean	The "ghost cell" flag.

13.10.28 [const] integer hierarchy_levels
Return the number of hierarchy levels below (expensive).

Return: integer The number of hierarchy levels below.

13.10.29 Instance cell_inst_array(CellInstArray cell_inst_array)
Insert a cell instance (array).

With version 0.16, this method returns an Instance object that represents the new instance. It's use is discouraged in read-only mode, since it invalidates other Instance references.

Input: cell_inst_array The given cell instance (array).
Return: Instance The new instance object.

13.10.30 Instance insert(Instance inst)
Insert a cell instance given by another reference.

This method allows to copy instances taken from a reference (an Instance object). It has been added in version 0.16.

Input: inst The instant object to be inserted.
Return: Instance The new instance object.

13.10.31 Instance insert(CellInstArray cell_inst_array)
Insert a cell instance (array).

With version 0.16, this method returns an Instance object that represents the new instance. It's use is discouraged in read-only mode, since it invalidates other Instance references.

Input: cell_inst_array The given cell instance (array).
Return: Instance The new instance object.

13.10.32 Instance insert(CellInstArray cell_inst_array, property_id)
Insert a cell instance (array) with properties.

The property Id must be obtained from the Layout object's property_id method which associates a property set with a property Id. With version 0.16, this method returns an Instance object that represents the new instance. It's use is discouraged in read-only mode, since it invalidates other Instance references.

Input: cell_inst_array The given cell instance (array).
property_id The property set Id.
Return: Instance The new instance object.

13.10.33 [const] boolean is_empty?
Returns a value indicating whether the cell is empty.

An empty cell is a cell not containing instances nor any shapes.

This method has been introduced in version 0.20.

Return: true The cell is empty.
false The cell is not empty.

13.10.34 [const] boolean is_ghost_cell?
Returns a value indicating whether the cell is a "ghost cell".

The ghost cell flag is used by the GDS reader for example to indicate that the cell is not located inside the file. Upon writing the reader can determine whether to write the cell or not. To satisfy the references inside the layout, a dummy cell is created in this case which has the "ghost cell" flag set to true.

This method has been introduced in version 0.20.

Return: true — The cell is a "ghost cell".
false — The cell is no "ghost cell".

13.10.35 [const] boolean is_leaf?
Tell if the cell is a leaf cell.

A cell is a leaf cell if there are no child instantiations.

Return: true — The cell is a leaf cell.
false — The cell is not a leaf cell.

13.10.36 [const] boolean is_top?
Tell if the cell is a top-level cell.

A cell is a top-level cell if there are no parent instantiations.

Return: true — The cell is a top-level cell.
false — The cell is not a top-level cell.

13.10.37 [const] boolean is_valid?(Instance inst)
Test if the given Instance object is still pointing to a valid object.

This method has been introduced in version 0.16.

Return: true — Another instance has been inserted already that occupies the original instances position.
false — The instance represented by the given reference has been deleted.

13.10.38 move(src, dest)
Move the shapes from the source to the target layer.

The target layer is not overwritten. Instead, the shapes are added to the shapes of the target layer.

This method has been introduced in version 0.19.

Input: src — The layer index of the source layer.
dest — The layer index of the destination layer.

13.10.39 [const] integer parent_cells
Report the number of parent cells.

Return: integer — The number of parent cells (cells which reference to this cell).

13.10.40 Instance replace(classInstance inst, CellInstArray cell_inst_array)
Replace a cell instance (array) with a different one.

This method has been introduced in version 0.16. It can only be used in editable mode. The instance given by the instance object (first argument) is replaced by the given instance (second argument). The new object will not have any properties.

Input:	inst	The instance object to be replaced.
	cell_inst_array	The given cell instance (array).
Return:	Instance	The new instance object without any properties.

13.10.41 Instance replace(Instance inst, CellInstArray cell_inst_array, property_id)
Replace a cell instance (array) with a different one with properties.

This method has been introduced in version 0.16. It can only be used in editable mode. The instance given by the instance object (first argument) is replaced by the given instance (second argument) with the given properties Id. The property Id must be obtained from the Layout object's property_id method which associates a property set with a property Id. The new object will not have any properties.

Input:	inst	The instance object to be replaced.
	cell_inst_array	The given cell instance (array).
	property_id	The property set Id.
Return:	Instance	The new instance object.

13.10.42 Instance replace_prop_id(Instance inst, unsigned int property_id)
Replace (or install) the properties of a cell.

This method has been introduced in version 0.16. It can only be used in editable mode. Changes the properties Id of the given instance or install a properties Id on that instance if it does not have one yet. The property Id must be obtained from the Layout object'sproperty_id method which associates a property set with a property Id.

Input:	inst	The instance object to be replaced or installed.
	property_id	The property set Id.
Return:	Instance	The new instance object.

13.10.43 ref Shapes shapes(integer)
Return the shapes list of the given layer.

This method allows to access the shapes list on a certain layer. If the layer does not exist yet, it is created.

Input:	integer	The layer index of the shapes list to retrieve.
Return:	ref	A reference to the shapes list.

13.10.44 swap(layer_index1, layer_index2)
Swap the layers given.

Input:	layer_index1	The first layer index.
	layer_index2	The second layer index.

13.10.45 Instance transform(Instance inst, Trans t)
Transform the instance given by the instance with the given transformation.

This method has been introduced in version 0.16. The original instance may be deleted and re-inserted by this method. Therefore, a new reference is returned. It is permitted in editable mode only.

Input:	inst	The instance to be transformed.
	t	The simple transformation to be performed.
Return:	Instance	A reference (an Instance object) to the new instance.

13.10.46 Instance transform(Instance inst, CplxTrans t)
Transform the instance given by the instance with the given complex transformation.

This method has been introduced in version 0.16. The original instance may be deleted and re-inserted by this method. Therefore, a new reference is returned. It is permitted in editable mode only.

Input:	inst	The instance to be transformed.
	t	The complex transformation to be performed.
Return:	Instance	A reference (an Instance object) to the new instance.

13.11 Class CellInstArray (version 0.21) A single or array cell instance.

This object represents either single or array cell instances. A cell instance array is a regular array, described by two displacement vectors (a, b) and the instance count along that axes (na, nb).

In addition, this object represents either instances with simple transformations or instances with complex transformations. The latter includes magnified instances and instances rotated by an arbitrary angle.

Method Overview

new	Default constructor.
new	Create a single cell instance.
new	Create a single cell instance with a complex transformation.
new	Create a single cell instance.
new	Create a single cell instance with a complex transformation.
bbox	The bounding box of the array.
bbox_per_layer	The bounding box of the array with respect to one layer.
size	The number of single instances in the array.
cell_index	Get the cell index of the cell instantiated.
cplx_trans	Get the complex transformation of the first instance in the array.
trans	Get the transformation of the first instance in the array.
invert	Invert an array reference.
transformed	Returns the transformed cell instance.
transformed	Returns the transformed cell instance (complex transformation).
transformed	Returns the transformed cell instance (complex transformation).
transform	Transform the cell instance with the given transformation.
transform	Transform the cell instance with the given complex transformation.
transform	Transform the cell instance with the given complex transformation.
<	Less operator.
==	Compare operator for equality.
!=	Compare operator for inequality.
is_complex?	Test, if the array is a complex array.
is_regular_array?	Test, if this instance is a regular array.
a	Return the displacement vector for the 'a' axis.
b	Return the displacement vector for the 'b' axis.
na	Return the number of instances in the 'a' axis.
nb	Return the number of instances in the 'b' axis.
assign	Assign the contents of another object to self.
dup	Creates a copy of self.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.11.1 [const] boolean !=(CellInstArray inst) Compare operator for inequality.

Input:	inst	This instance.
Return:	true	This instance and the given instance are unequal.
	false	This instance and the given instance are equal.

13.11.2 [const] boolean <(CellInstArray inst)
Less operator.

Input:	inst	This instance.
Return:	true	This instance is 'less' than the given instance.
	false	This instance is 'greater' than the given instance.

13.11.3 [const] boolean ==(CellInstArray inst)
Compare operator for equality.

Input:	inst	This instance.
Return:	true	This instance and the given instance are equal.
	false	This instance and the given instance are unequal.

13.11.4 [const] Point a
Return the displacement vector for the 'a' axis.

Return:	Return the displacement vector for the 'a' axis.

13.11.5 assign(CellInstArray other)
Assign the contents of another object to self.

This method assigns the contents of another object to self. This is a deep copy that does not only copy the reference but the actual content.

13.11.6 [const] Point b
Return the displacement vector for the 'b' axis.

Return:	Return the displacement vector for the 'b' axis.

13.11.7 [const] Box bbox(Layout layout)
The bounding box of the array.

The bounding box incorporates all instances that the array represents. It needs the layout object to access the actual cell from the cell index.

13.11.8 [const] Box bbox_per_layer(Layout layout, layer_index)
The bounding box of the array with respect to one layer.

The bounding box incorporates all instances that the array represents. It needs the layout object to access the actual cell from the cell index.

13.11.9 [const] integer cell_index
Get the cell index of the cell instantiated.

13.11.10 [const] CplxTrans cplx_trans
Get the complex transformation of the first instance in the array.

This method is always valid compared to trans, since simple transformations can be expressed as complex transformations as well.

13.11.11 destroy
Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.11.12 [const] boolean destroyed
Tell, if the object was destroyed.

Return: true — The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
false — The object is still alive.

13.11.13 [const] CellInstArray dup
Creates a copy of self.

Return: CellInstArray — The copy of self.

13.11.14 invert
Invert an array reference.

The inverted array reference describes in which transformations the parent cell is seen from the current cell.

13.11.15 [const] boolean is_complex?
Test, if the array is a complex array.

Return: true — The array represents complex instances (that is, with magnification and arbitrary rotation angles).
false — The array represents simple instances.

13.11.16 [const] boolean is_regular_array?
Test, if this instance is a regular array.

Return: true — The array represents regular instances (that is, without magnification and arbitrary rotation angles).
false — The array represents simple instances.

13.11.17 [const] long na
Return the number of instances in the 'a' axis.

Return: long — The number of instances in the 'a' axis.

13.11.18 [const] long nb
Return the number of instances in the 'b' axis.

Return: long — The number of instances in the 'b' axis.

13.11.19 [static] CellInstArray new
Default constructor.

13.11.20 [static] CellInstArray new(cell_index, Trans t)
Create a single cell instance.

A synonym of: [static] CellInstArray new_inst(cell_index, Trans t).

Input:	cell_index	The cell to instantiate.
	t	The complex transformation by which to instantiate the cell.
Return:	CellInstArray	The newly created cell instance array.

13.11.21 [static] CellInstArray new(cell_index, CplxTrans t)
Create a single cell instance with a complex transformation.

A synonym of: [static] CellInstArray new_inst_cplx(cell_index, CplxTrans t).

Input:	cell_index	The cell to instantiate.
	t	The complex transformation by which to instantiate the cell.
Return:	CellInstArray	The newly created cell instance array.

13.11.22 [static] CellInstArray new(cell_index, Trans t,Point a, Point b, na, nb)
Create a single cell instance.

A synonym of: [static] CellInstArray new_inst_array(cell_index, Trans t,Point a, Point b, na, nb).

Input:	cell_index	The cell to instantiate.
	t	The complex transformation by which to instantiate the cell.
	a	The displacement vector of the array in the 'a' axis.
	b	The displacement vector of the array in the 'b' axis.
	na	The number of placements in the 'a' axis.
	nb	The number of placements in the 'b' axis.
Return:	CellInstArray	The newly created cell instance array.

13.11.23 [static] CellInstArray new(cell_index, CplxTrans t, Point b, Point b, na,nb)
Create a single cell instance with a complex transformation.

A synonym of: [static] CellInstArray new_inst_array_cplx(cell_index, CplxTrans t, Point b, Point b, na,nb).

Input:	cell_index	The cell to instantiate.
	t	The complex transformation by which to instantiate the cell.
	a	The displacement vector of the array in the 'a' axis.
	b	The displacement vector of the array in the 'b' axis.
	na	The number of placements in the 'a' axis.
	nb	The number of placements in the 'b' axis.
Return:	CellInstArray	The newly created cell instance array.

13.11.24 [const] integer size
The number of single instances in the array.

If the instance represents a single instance, the count is 1. Otherwise it is na*nb.

13.11.25 [const] const refTrans trans

Get the transformation of the first instance in the array.

The transformation returned is only valid if the array does not represent a complex transformation array.

13.11.26 transform(Trans t)

Transform the cell instance with the given transformation.

This method has been introduced in version 0.20.

13.11.27 transform(CplxTrans t)

Transform the cell instance with the given complex transformation.

This method has been introduced in version 0.20.

13.11.28 transform(ICplxTrans t)

Transform the cell instance with the given complex transformation.

This method has been introduced in version 0.20.

13.11.29 [const] CellInstArray transformed(Trans t)

Returns the transformed cell instance.

This method has been introduced in version 0.20.

13.11.30 [const] CellInstArray transformed(CplxTrans t)

Returns the transformed cell instance (complex transformation).

This method has been introduced in version 0.20.

13.11.31 [const] CellInstArray transformed(ICplxTrans t)

Returns the transformed cell instance (complex transformation).

This method has been introduced in version 0.20.

13.12 Class CellMapping (version 0.21) A cell mapping derived from two hierarchies.

A cell mapping is an association of cells in two layouts forming pairs of cells, i.e. on cell corresponds to another cell in the other layout. Correspondency is defined by exact identity of both flat instantiations in the given staring cell. Therefore, when a cell is mapped to another cell, shapes can be transferred from one cell to another while effectively rendering the same flat geometry (in the context of the given starting cells).

A cell might not be mapped to another cell which basically means that there is no corresponding cell. In this case, flattening to the next mapped cell is an option to transfer geometries despite the missing mapping.

A cell mapping is created by instantiating a cell mapping object. Pass two layouts and two starting cells to specify which cell trees to map.

Method Overview

new	Create a new cell mapping.
has_mapping?	Determine if a layout_b cell has a mapping to a layout_a cell.
cell_mapping	Determine cell mapping to a layout_b cell to the corresponding layout_a cell.
assign	Assign the contents of another object to self.
dup	Creates a copy of self.
destroy	Explicitly destroy the object
destroyed	Tell, if the object was destroyed

13.12.1 assign(CellMapping other) Assign the contents of another object to self.

This method assigns the contents of another object to self. This is a deep copy that does not only copy the reference but the actual content.

13.12.2 [const] cell_index_a cell_mapping(cell_index_b) Determine cell mapping to a layout_b cell to the corresponding layout_a cell.

Input: cell_index_b The index of the cell in layout_b whose mapping is requested.
Return: cell_index_a The cell index in layout_a.

13.12.3 destroy Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.12.4 [const] boolean destroyed Tell, if the object was destroyed.

Return: true The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
false The object is still alive.

13.12.5 [const] CellMapping dup

Creates a copy of self.

Return: CellMapping The copy of self.

13.12.6 [const] boolean has_mapping?(cell_index_b)

Determine if a cell layout_b has a mapping to a layout_a cell.

Input: cell_index_b The index of the cell in layout_b whose mapping is requested.
Return: true The cell has a mapping.
false The cell has no mapping.

13.12.7 [static] CellMapping new(Layout layout_a, cell_index_a, Layout layout_b, cell_index_b)

Create a new cell mapping.

The cell mapping is created for cells below cell_a and cell_b in the respective layouts.

13.13 Class CellView (version 0.21) A "cell view" reference.

A cell view reference points to a certain cell within a certain layout. The layout pointer can be nil, indicating that it is invalid. Also, the cell view describes a cell within that layout. The cell is addressed by a cell index or a cell object reference.

The cell is not only identified by it's index or object but as well by the path leading to that cell. This path describes how to find the cell in the context of it's parent cells.

The path is in fact composed in two ways: once in an unspecific fashion, just describing which parent cells are used. The target of this path is called the context cell. It is accessible by the ctx_cell_index or ctx_cell methods.

Additionally the path may further identify a certain instance of a certain sub-cell in the context cell. This is done through a set of InstElement objects. The target of this context path is the actual cell addressed by the cell view. This target cell is accessible by the cell_index or cell methods. In the viewer, the target cell is shown in the context of the context cell. The hierarchy levels are counted from the context cell, which is on level 0. If the context path is empty, the context cell is identical with the target cell.

Method Overview

==	Equality: compares the cell the view points to, not the path.
is_valid?	Test if the view points to a valid cell.
set_path	Set the unspecific part of the path explicitly.
set_context_path	Set the context path explicitly.
set_cell	Set the path to the given cell.
set_cell_name	Set the cell by name.
reset_cell	Reset the cell.
ctx_cell_index	Get the context cell's index.
ctx_cell	Get the reference to the context cell currently addressed.
cell_index	Get the target cell's index.
cell	Get the reference to the target cell currently addressed.
filename	Get file name associated with the layout behind the cell view.
name	Get the unique name associated with the layout behind the cell view.
path	Get the cell's unspecific part of the path leading to the context cell.
context_path	Get the cell's context path.
layout	Get the reference to the layout object addressed by this view.
assign	Assign the contents of another object to self.
dup	Creates a copy of self.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.13.1 [const] boolean ==(CellView other) Equality test compares the cell the view points to, not the path.

13.13.2 assign(CellView other) Assign the contents of another object to self.

This method assigns the contents of another object to self. This is a deep copy that does not only copy the reference but the actual content.

13.13.3 [const] ref Cellcell
Get the reference to the target cell currently addressed.

13.13.4 [const] integer cell_index
Get the target cell's index.

13.13.5 [const] InstElement[] context_path
Get the cell's context path.

The context path leads from the context cell to the target cell in a specific fashion, i.e. describing each instance in detail, not just be cell indices. If the context and target cell are identical, the context path is empty.

13.13.6 [const] ref Cell ctx_cell
Get the reference to the context cell currently addressed.

13.13.7 [const] integer ctx_cell_index
Get the context cell's index.

13.13.8 destroy
Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.13.9 [const] boolean destroyed
Tell, if the object was destroyed.

Return:	true	The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
	false	The object is still alive.

13.13.10 [const] CellView dup
Creates a copy of self.

Return:	CellView	The copy of self.

13.13.11 [const] string filename
Get the file name associated with the layout behind the cell view.

Return:	string	The file name associated with the layout.

13.13.12 [const] boolean is_valid?
Test if the view points to a valid cell.

Return:	true	The view points to a valid cell.
	false	The view points to an invalid cell.

13.13.13 [const] ref Layout layout
Get the reference to the layout object addressed by this view.

Return: ref The reference to the layout.

13.13.14 [const] string name
Get the unique name associated with the layout behind the cell view.

Return: string The unique name associated with the layout.

13.13.15 [const] integer[] path
Get the cell's unspecific part of the path leading to the context cell.

Return: integer[] The cell's unspecific part of the path leading to the context cell.

13.13.16 reset_cell
Reset the cell.

The cell view will become invalid. The layout object will still be attached to the cellview.

13.13.17 set_cell(integer)
Set the path to the given cell.

This method will construct any path to this cell, not a particular one. It will clear the context path and update the context and target cell.

13.13.18 set_cell_name(string)
Set the cell by name.

If the name is not a valid one, the cell view will become invalid. This method will construct any path to this cell, not a particular one. It will clear the context path and update the context and target cell.

13.13.19 set_context_path(InstElement path[])
Set the context path explicitly.

This method assumes that the unspecific part of the path is established already and that the context path starts from the context cell.

13.13.20 set_path(integerpath[])
Set the unspecific part of the path explicitly.

Setting the unspecific part of the path will clear the context path component and update the context and target cell.

13.14 Class Class (version 0.21)
The interface to the declarations of classes and methods.

Method Overview

each_class	Iterate over all classes.
each_method	Iterate over all methods of this class.
name	The name of the class.
can_copy	True if the class offers assignment.
doc	The documentation string for this class.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.14.1 [const] boolean can_copy
True if the class offers assignment.

Return: true — The class offers assignment.
false — The class offers no assignment.

13.14.2 destroy
Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.14.3 [const] boolean destroyed
Tell, if the object was destroyed.

Return: true — The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
false — The object is still alive.

13.14.4 [const] string doc
The documentation string for this class.

Return: string — The documentation string.

13.14.5 [static] yield const ref Class each_class
Iterate over all classes.

Return: yield — An array of references to all methods of all classes.

13.14.6 [static] yield ref each_method
Iterate over all methods of this class.

Return: yield — An array of references to all methods of this class.

13.14.7 [const] string name
The name of the class.

Return: string The name of the class.

13.15 Class CplxTrans (version 0.21) A complex transformation.

A complex transformation provides magnification, mirroring at the x-axis, rotation by an arbitrary angle and a displacement. This version can transform integer-coordinate objects into floating-point coordinate objects, which is the generic and exact case.

Method Overview

from_dtrans	Conversion constructor from an floating-point transformation.
new	Creates a unit transformation.
new	Conversion constructor from a fix-point transformation.
new	Constructor from a magnification.
new	Constructor from a simple transformation and a magnification.
new	Constructor from a simple transformation alone.
new	The standard constructor using magnification, angle, mirror flag and displacement.
inverted	Inversion.
invert	In-place inversion.
ctrans	The transformation of a distance.
trans	The transformation of a point.
*****	Multiplication (concatenation) of transformations.
<	A sorting criterion.
==	Equality test.
!=	Inequality test.
to_s	String conversion.
disp	Gets the displacement.
disp=	Sets the displacement.
rot	Returns the respective rotation code if possible.
is_mirror?	Gets the mirror flag.
mirror=	Sets the mirror flag.
is_unity?	Test, whether this is a unit transformation.
is_ortho?	Test, if the transformation is an orthogonal transformation.
s_trans	Extract the simple transformation part.
angle	Gets the angle.
angle=	Sets the angle.
mag	Gets the magnification.
is_mag?	Test, if the transformation is a magnifying one.
mag=	Sets the magnification.
m_*/r_*	Various angle/mirror codes for the named transformation.
r0	"unrotated" transformation.
r90	"rotated by 90 degree counterclockwise" transformation.
r180	"rotated by 180 degree counterclockwise" transformation.
r270	"rotated by 270 degree counterclockwise" transformation.
m0	"mirrored at the x-axis" transformation.
m45	"mirrored at the 45 degree axis" transformation.
m90	"mirrored at the y (90 degree) axis" transformation.
m135	"mirrored at the 135 degree axis" transformation.
assign	Assign the contents of another object to self.
dup	Creates a copy of self.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.15.1 [const] boolean !=(CplxTrans)
Inequality test.

Input:	CplxTrans text	The object to compare against.
Return:	true	This object and the given one are not equal.
	false	???.

13.15.2 [const] CplxTrans *(CplxTrans t)
Multiplication (concatenation) of transformations.

The * operator returns self*t ("t is applied before this transformation").

Input:	t	The transformation to apply before.
Return:	CplxTrans	The modified transformation.

13.15.3 [const] boolean <(CplxTrans)
A sorting criterion.

Input:	e	The object to compare against.
Return:	true	The object is 'less' than the other.
	false	??.

13.15.4 [const] boolean ==(CplxTrans)
Equality test.

Input:	e	The object to compare against.
Return:	true	Equality.
	false	??.

13.15.5 [const] double angle
Gets the angle.

To check, if the transformation represents a rotation by a angle that is a multiple of 90 degree, use this predicate.

Return:	double	The rotation angle this transformation provides in degree units (0..360 deg).

13.15.6 angle=(double a)
Sets the angle.

Input:	a	The new angle.

13.15.7 assign(CplxTrans other)
Assign the contents of another object to self.

This method assigns the contents of another object to self. This is a deep copy that does not only copy the reference but the actual content.

13.15.8 [const] double ctrans(d)
The transformation of a distance.

The ctransethod transforms the given distance: $e = t(d)$. For the simple transformations, there is no magnification and no modification of the distance therefore.

Input: d — The distance to transform.
Return: double — The transformed distance.

13.15.9 destroy
Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.15.10 [const] boolean destroyed
Tell, if the object was destroyed.

Return: true — The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
false — The object is still alive.

13.15.11 [const] u DPoint disp
Gets the displacement.

Return: u — The displacement.

13.15.12 disp=(DPoint u)
Sets the displacement.

Input: u — The new displacement.

13.15.13 [const] CplxTrans dup
Creates a copy of self.

Return: CplxTrans — The copy of self.

13.15.14 [static] CplxTrans from_dtrans(CplxTrans dbl_trans)
Conversion constructor from an floating-point transformation.

13.15.15 CplxTrans invert
In-place inversion.

Inverts the transformation and replaces this transformation by the inverted one.

Return: CplxTrans — The inverted transformation.

13.15.16 [const] CplxTrans inverted
Inversion.

Return: CplxTrans — The inverted transformation.

13.15.17 [const] boolean is_mag?
Test, if the transformation is a magnifying one.

This is the recommended test for checking if the transformation represents a magnification.

Return: true — The transformation is a magnifying.
false — ???.

13.15.18 [const] boolean is_mirror?
Gets the mirror flag.

Return: true — The transformation is composed of a mirroring at the x-axis followed by a rotation by the angle given by the angle property.
false — ???.

13.15.19 is_ortho?
Test, if the transformation is an orthogonal transformation.

Return: true — The rotation is by a multiple of 90 degree.
false — The rotation is not orthogonal.

13.15.20 [const] boolean is_unity?
Test, whether this is a unit transformation.

Return: true — A unit transformation.
false — Any other transformation.

13.15.21 [static] integer m_*/r_*
Various angle/mirror codes for the named transformation.

13.15.21.1 [static] integer m0 – "mirrored at the x-axis".

Return: integer — The angle/mirror code for this transformation.

13.15.21.2 [static] integer m135 – "mirrored at the 135 degree axis"

Return: integer — The angle/mirror code for this transformation.

13.15.21.3 [static] integer m45 – "mirrored at the 45 degree axis".

Return: integer — The angle/mirror code for this transformation.

13.15.21.4 [static] integer m90 – "mirrored at the 90 degree axis".

Return: integer — The angle/mirror code for this transformation.

13.15.21.5 [static] integer r0 – "unrotated".

Return: integer — The angle/mirror code for this transformation.

13.15.21.6 [static] integer r180 – "rotated by 180 degree counterclockwise".

Return: integer The angle/mirror code for this transformation.

13.15.21.7 [static] integer r270 – "rotated by 270 degree counterclockwise".

Return: integer The angle/mirror code for this transformation.

13.15.21.8 [static] integer r90 – "rotated by 90 degree counterclockwise".

Return: integer The angle/mirror code for this transformation.

13.15.22 [const] double mag
Gets the magnification.

Return: integer The angle/mirror code for this transformation.

13.15.23 mag=(double m)
Sets the magnification.

Input: double m The new magnification.

13.15.24 mirror=(boolean)
Sets the mirror flag.

"mirroring" describes a reflection at the x-axis which is included in the transformation prior to rotation.

Input: boolean The new mirror flag.

13.15.25 [static] CplxTrans new
Creates a unit transformation.

13.15.26 [static] CplxTrans new(f)
Conversion constructor from a fix-point transformation.

A synonym of: [static] CplxTrans new_f(f).

This constructor will create a transformation with a fix point transformation but no displacement.

Input: f The rotation/mirror code (r0 .. m135 constants).

13.15.27 [static] CplxTrans new(double m)
Constructor from a magnification.

A synonym of: [static] CplxTrans new_m(double m).

Creates a magnifying transformation without displacement and rotation given the magnification m.

Input: double m The magnification.

13.15.28 [static] CplxTrans new(Trans t, double m)
Constructor from a simple transformation and a magnification.

A synonym of: [static] CplxTrans new_tm(Trans t, double m).

Input:	t	The transformation.
	double m	The magnification.
Return:	CplxTrans	The resulting complex transformation from a simple transformation and a magnification.

13.15.29 [static] CplxTrans new(Trans t)
Constructor from a simple transformation alone.

A synonym of: [static] CplxTrans new_t(Trans t).

Input:	t	The transformation.
Return:	CplxTrans	The resulting complex transformation from a simple transformation and a magnification of 1.0.

13.15.30 [static] CplxTrans new(double m, double r, boolean, DPoint u)
The standard constructor using magnification, angle, mirror flag and displacement.

A synonym of: [static] CplxTrans new_mrmu(double m, double r, boolean, DPoint u).

The sequence of operations is: magnification, mirroring at x axis, rotation, application of displacement.

Input:	double m	The magnification.
	double r	The rotation angle in units of degree.
	boolean	True, if mirrored at x axis.
	u	The displacement.

13.15.31 [const] integer rot
Returns the respective rotation code if possible.

If this transformation is orthogonal (is_ortho = true), then this method will return the corresponding fix point transformation, not taking into account magnification and displacement. Otherwise, the result reflects the quadrant the rotation goes into with the guarantee to reproduce the correct quadrant in the exact case.

13.15.32 [const] Trans s_trans
Extract the simple transformation part.

The simple transformation part does not reflect magnification not arbitrary angles. On the angle contribution up to a multiple of 90 degree is reflected.

13.15.33 [const] string to_s
String conversion.

Return:	string	The resulting string.

13.15.34 [const] DPoint trans(Point p)
The transformation of a point.

The "trans" method transforms the given point $q = t(p)$.

Input:	p	The point to transform.
Return:	DPoint	The transformed point.

13.16 Class DBox (version 0.21) A box class with double (floating-point) coordinates.

This object represents a box (a rectangular shape).

The notation is: p1 is the lower left point (x1, y1), p2 the upper right one (x2, y2), compare with fig. 13.2.

A box can be empty. An empty box represents no area (not even a point).

A box can be a point or a single line. In this case, the area is zero but the box still can overlap other boxes.

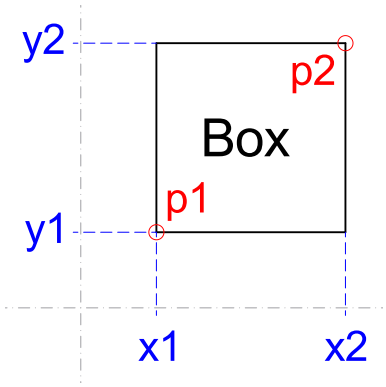

Figure 13.2. Box notation.

Method Overview

from_ibox	Construct a floating-point coordinate box from an integer coordinate box.
new	Default constructor: creates an empty (invalid) box.
new	Constructor with four coordinates.
new	Constructor with two points.
p1	Get the lower left point of the box.
p2	Get the upper right point of the box.
center	Get the center of the box.
left	Get the left coordinate of the box.
right	Get the right coordinate of the box.
bottom	Get the bottom coordinate of the box.
top	Get the top coordinate of the box.
width	Get the width of the box.
height	Get the height of the box.
left=	Set the left coordinate of the box.
right=	Set the right coordinate of the box.
bottom=	Set the bottom coordinate of the box.
top=	Set the top coordinate of the box.
p1=	Set the lower left point of the box.
p2=	Set the upper right point of the box.
contains?	Test if a point is inside the box.
empty?	Test if this box is of type empty box.
inside?	Test if this box is inside the argument box.
touches?	Test if this box touches the argument box.
overlaps?	Test if this box overlaps the argument box.
area	Compute the box area
is_point?	Test if the box is a single point
+	Join a box with a point.
+	Joining of two boxes.

&	Intersection of two boxes.
*****	Convolve two boxes.
move	Moves the box by a certain distance.
moved	Get the box moved by a certain distance.
enlarge	Enlarges the box by a certain amount.
enlarged	Get the box enlarged by a certain amount.
transformed	Transform the box with the given simple transformation
transformed_cplx	Transform the box with the given complex transformation
<	Less operator.
==	Equality operator.
!=	Inequality operator.
to_s	Convert to a string.
assign	Assign the contents of another object to self.
dup	Creates a copy of self.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.16.1 [const] boolean !=(DBox box)
Inequality test of two boxes.

Test if this box and the given box are not equal.

Input:	box	The given box.
Return:	true	This and the given box are unequal.
	false	This and the given box are equal.

13.16.2 DBox &(DBox box)
Intersection of two boxes.

The intersection of two boxes is the largest box common to both boxes. The intersection may be empty if both boxes to not touch. If the boxes do not overlap but touch the result may be a single line or point with an area of zero. Overwrites this box with the result.

Input:	box	The box to take the intersection with.
Return:	DBox	The intersection box.

13.16.3 DBox *(DBox box)
Convolve two boxes.

The * operator convolve the first box with the one given as the second argument. The box resulting from 'convolution' is the outer boundary of the union set formed by placing the second box at every point of the first. In other words, the returned box of (p1,p2)*(q1,q2) is (p1+q1,p2+q2).

Input:	box	The given box.
Return:	DBox	The intersection box.

13.16.4 [const] DBox +(DPoint point)
Join a box with a point.

The + operator joins a point with the box. The resulting box will enclose both the original box and the point.

Input:	point	The point to join with this box.
Return:	DBox	The box joined with the point.

13.16.5 DBox +(DBox box)
Joining of two boxes.

The + operator joins the first box with the one given as the second argument. Joining constructs a box that encloses both boxes given. Empty boxes are neutral: they do not change another box when joining. Overwrites this box with the result.

Input:	box	The box to join with this box.
Return:	DBox	The joined box.

13.16.6 [const] boolean <(DBox box)
Less operator.

Input:	box	This box.
Return:	true	This box is 'less' with respect to first and second point (in this order).
	false	This box is 'greater'.

13.16.7 [const] boolean ==(DBox box)
Equality operator.

Input:	box	This box.
Return:	true	This box and the given box are equal.
	false	This box and the given box are unequal.

13.16.8 [const] double area
Compute the box area.

Return:	double integer	The box area, or
	0	the box is empty.

13.16.9 assign(DBox other)
Assign the contents of another object to self.

This method assigns the contents of another object to self. This is a deep copy that does not only copy the reference but the actual content.

Input:	other	The contents of another object.

13.16.10 bottom=(y1)
Set the bottom coordinate of the box.

Input:	y1	The bottom coordinate of the box.

13.16.11 [const] y1 bottom
Query the bottom coordinate of the box.

Return:	y1	The bottom coordinate of the box.

13.16.12 [const] DPoint center
Query the center of the box.

Return: DPoint — The center coordinate of the box.

13.16.13 [const] boolean contains?(DPoint point)
Tests if a point is inside the box.

Input: point — The coordinate to be tested.
Return: true — The point is placed inside the box or on the box contour.
false — The point is placed completely outside the box.

13.16.14 destroy
Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.16.15 [const] boolean destroyed
Tell, if the object was destroyed.

Return: true — The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
false — The object is still alive.

13.16.16 [const] DBox dup
Creates a copy of self.

Return: DBox — The copy of self.

13.16.17 [const]boolean empty?
Test if the box is of type empty box.

An empty box may be created with the default constructor for example. Such a box is neutral when combining it with other boxes and renders empty boxes if used in box intersections and false in geometrical relationship tests.

Return: true — The box is empty.
false — The box is not empty.

13.16.18 ref DBox enlarge(DPoint enlargement)
Enlarges the box by a certain amount.

Enlarges the box by x and y value specified in the vector passed. Positive values with grow the box, negative ones will shrink the box. The result may be an empty box if the box disappears. The amount specifies the grow or shrink per edge. The width and height will change by twice the amount. Does not check for coordinate overflows.

Input: enlargement — The grow or shrink amount in x and y direction.
Return: ref — A reference to the enlarged box.

13.16.19 [const] DBox enlarged(DPoint enlargement)
Get the box enlarged by a certain amount.

Enlarges the box by **x** and **y** value specified in the vector passed. Positive values will grow the box, negative ones will shrink the box. The result may be an empty box if the box disappears. The amount specifies the grow or shrink per edge. The width and height will change by twice the amount. Does not modify this box. Does not check for coordinate overflows.

Input: enlargement — The grow or shrink amount in x and y direction.
Return: DBox — The enlarged box.

13.16.20 [static] DBox from_ibox(Box int_box)
Construct a floating-point coordinate box from an integer coordinate box.

Create a floating-point coordinate box from an integer coordinate box.

Input: int_box — The floating-point coordinate box.
Return: DBox — The integer coordinate box.

13.16.21 [const] height height
Query the height of the box.

Return: height — The height of the box as double integer, where the equation $height = y2 - y1$ is valid.

13.16.22 [const] boolean inside?(DBox box)
Test if this box is inside the argument box.

Input: box — The given box.
Return: true — This box is inside the given box, i.e. the box intersection renders this box.
false — This box is not inside the given box.

13.16.23 [const] boolean is_point?
Test if the box is a single point.

Return: true — The box is a single point.
false — The box is not a single point.

13.16.24 left=(x1)
Set the left coordinate of the box.

Input: x1 — The left coordinate of the box.

13.16.25 [const] x1 left
Query the left coordinate of the box.

Return: x1 — The left coordinate of the box as double integer.

13.16.26 ref DBox move(DPoint distance)
Moves the box by a certain distance.

Moves the box by a given offset and returns the moved box. Does not check for coordinate overflows.

Input:	distance	The offset to move the box.
Return:	ref	A reference to this box.

13.16.27 [const] DBox moved(DPoint distance)
Get the box moved by a certain distance.

Moves the box by a given offset and returns the moved box. Does not modify this box. Does not check for coordinate overflows.

Input:	distance	The offset to move the box.
Return:	DBox	The moved box.

13.16.28 [static] DBox new
Default constructor: creates an empty (invalid) box.

Return:	DBox	The new empty box.

13.16.29 [static] DBox new(left, bottom, right, top)
Constructor with four coordinates.

Synonym for [static] DBox new-lbrt(left, bottom, right, top)

Four coordinates are given to create a new box. If the coordinates are not provided in the correct order (i.e. right <left), these are swapped.

Input:	left, bottom, right, top	Four coordinates given to create a new box, where left equals to x1, bottom to y1, right to x2 and top to y2.
Return:	DBox	The new box.

13.16.30 [static] DBox new(DPoint lower_left, DPoint upper_right)
Box constructor with two points.

Synonym for [static] DBox new_pp(DPoint lower_left, DPoint upper_right).

Two points are given to create a new box. If the coordinates are not provided in the correct order (i.e. right <left), these are swapped.

Input:	lower_left, upper_right	Two points given to create a new box.
Return:	DBox	The new box.

13.16.31 [const] boolean overlaps?(DBox box)
Test if this box overlaps the argument box.

Input:	box	The argument box.
Return:	true	The intersection box of this box with the argument box exists and has a non-vanishing area.
	false	The intersection box of this box with the argument box does not exists or has a vanishing area.

13.16.32 [const] ref DPoint p1
Query the lower left point of the box.

Return: lower_left The lower left point of the box, where lower_left equals to x1, y1.

13.16.33 p1=(DPoint lower_left)
Set the lower left point of the box.

Input: lower_left The lower left point of the box, where lower_left equals to x1, y1.

13.16.34 [const] ref DPoint p2
Query the upper right point of the box.

Return: upper_right The upper right point of the box, where upper_right equals to x2, y2.

13.16.35 p2=(DPoint upper_right)
Set the upper right point of the box.

Input: upper_right The upper right point of the box, where upper_right equals to x2, y2.

13.16.36 [const] x2 right
Query the right coordinate of the box.

Return: x2 The right coordinate of the box as double integer.

13.16.37 right=(x2)
Set the right coordinate of the box.

Input: x2 The right coordinate of the box.

13.16.38 [const] string to_s
Convert a value to a string.

Return: string The converted value as string.

13.16.39 [const]y2 top
Query the top coordinate of the box.

Return: y2 The top coordinate of the box as double integer.

13.16.40 top=(y2)
Set the top coordinate of the box.

Input: y2 The top coordinate of the box.

13.16.41 [const] boolean touches?(DBox box)

Test if this box touches the argument box.

Input:	box	The argument box
Return:	true	This box has at least one point common with the argument box.
	false	This box has none point common with the argument box.

13.16.42 [const] DBox transformed(DTrans t)

Transform the box with the given simple transformation.

Input:	t	The simple transformation to apply.
Return:	Box	The transformed box.

13.16.43 [const] DBox transformed_cplx(DCplxTrans t)

Transform the box with the given complex transformation.

Input:	t	The complex transformation to apply.
Return:	DBox	The transformed box (a DBox now).

13.16.44 [const] width width

Query the width of the box.

Return:	width	The width of the box as double integer, where width equals to x2 - x1.

13.17 Class DCplxTrans (version 0.21) A complex transformation.

A complex transformation provides magnification, mirroring at the x-axis, rotation by an arbitrary angle and a displacement. This version can transform integer-coordinate objects into floating-point coordinate objects, which is the generic and exact case.

Method Overview

from_itrans	Conversion constructor from an integer coordinate transformation.
new	Creates a unit transformation.
new	Conversion constructor from a fix-point transformation.
new	Constructor from a magnification.
new	Constructor from a simple transformation and a magnification.
new	Constructor from a simple transformation alone.
new	The standard constructor using magnification, angle, mirror flag and displacement.
inverted	Inversion.
invert	In-place inversion.
ctrans	The transformation of a distance.
trans	The transformation of a point.
*****	Multiplication (concatenation) of transformations.
<	A sorting criterion.
==	Equality test.
!=	Inequality test.
to_s	String conversion.
disp	Gets the displacement.
disp=	Sets the displacement.
rot	Returns the respective rotation code if possible.
is_mirror?	Gets the mirror flag.
mirror=	Sets the mirror flag.
is_unity?	Test, whether this is a unit transformation.
is_ortho?	Test, if the transformation is an orthogonal transformation.
s_trans	Extract the simple transformation part.
angle	Gets the angle.
angle=	Sets the angle.
mag	Gets the magnification.
is_mag?	Test, if the transformation is a magnifying one.
mag=	Sets the magnification.
m_*/r_*	Various angle/mirror codes for the named transformation.
r0	"unrotated" transformation.
r90	"rotated by 90 degree counterclockwise" transformation.
r180	"rotated by 180 degree counterclockwise" transformation.
r270	"rotated by 270 degree counterclockwise" transformation.
m0	"mirrored at the x-axis" transformation.
m45	"mirrored at the 45 degree axis" transformation.
m90	"mirrored at the y (90 degree) axis" transformation.
m135	"mirrored at the 135 degree axis" transformation.
assign	Assign the contents of another object to self.
dup	Creates a copy of self.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.17.1 [const] boolean !=(DCplxTrans)
Inequality test.

Input:	DTrans text	The object to compare against.
Return:	true	This object and the given one are not equal.
	false	???.

13.17.2 [const] DCplxTrans *(DCplxTrans t)
Multiplication (concatenation) of transformations.

The * operator returns self*t ("t is applied before this transformation").

Input:	t	The transformation to apply before.
Return:	DCplxTrans	The modified transformation.

13.17.3 [const] boolean <(DCplxTrans)
A sorting criterion.

Input:	e	The object to compare against.
Return:	true	The object is 'less' than the other.
	false	??.

13.17.4 [const] boolean ==(DCplxTrans)
Equality test.

Input:	e	The object to compare against.
Return:	true	Equality.
	false	??.

13.17.5 [const] double angle
Gets the angle.

To check, if the transformation represents a rotation by an angle that is a multiple of 90 degree, use this predicate.

Return:	double	The rotation angle this transformation provides in degree units (0..360 deg).

13.17.6 angle=(double)
Sets the angle.

Input:	double	The new angle.

13.17.7 assign(DCplxTrans other)
Assign the contents of another object to self.

This method assigns the contents of another object to self. This is a deep copy that does not only copy the reference but the actual content.

13.17.8 [const] double ctrans(d)
The transformation of a distance.

The ctransethod transforms the given distance: $e = t(d)$. For the simple transformations, there is no magnification and no modification of the distance therefore.

Input: d — The distance to transform as double integer.
Return: double — The transformed distance.

13.17.9 destroy
Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.17.10 [const] boolean destroyed
Tell, if the object was destroyed.

Return: true — The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
false — The object is still alive.

13.17.11 [const] const ref DPoint disp
Gets the displacement.

13.17.12 disp=(DPoint u)
Sets the displacement.

Input: u — The new displacement.

13.17.13 [const] DCplxTrans dup
Creates a copy of self.

Return: DCplxTrans — The copy of self.

13.17.14 [static] DCplxTrans from_itrans(DCplxTrans dbl_trans)
Conversion constructor from an floating-point transformation.

13.17.15 DCplxTrans invert
In-place inversion.

Inverts the transformation and replaces this transformation by the inverted one.

Return: DCplxTrans — The inverted transformation.

13.17.16 [const] DCplxTrans inverted
Inversion.

Return: DCplxTrans — The inverted transformation.

13.17.17 [const] boolean is_mag?
Test, if the transformation is a magnifying one.

This is the recommended test for checking if the transformation represents a magnification.

13.17.18 [const] boolean is_mirror?
Gets the mirror flag.

Return: true — The transformation is composed of a mirroring at the x-axis followed by a rotation by the angle given by the angle property.
false — ???.

13.17.19 [const] boolean is_ortho?
Test, if the transformation is an orthogonal transformation.

Return: true — The rotation is by a multiple of 90 degree.
false — The rotation is not orthogonal.

13.17.20 [const] boolean is_unity?
Test, whether this is a unit transformation.

Return: true — A unit transformation.
false — An other transformation.

13.17.21 [static] integer m_*/r_*
Various angle/mirror codes for the named transformation.

13.17.21.1 [static] integer m0 – "mirrored at the x-axis".

Return: integer — The angle/mirror code for this transformation.

13.17.21.2 [static] integer m135 – "mirrored at the 135 degree axis".

Return: integer — The angle/mirror code for this transformation.

13.17.21.3 [static] integer m45 – "mirrored at the 45 degree axis".

Return: integer — The angle/mirror code for this transformation.

13.17.21.4 [static] integer m90 – "mirrored at the 90 degree axis".

Return: integer — The angle/mirror code for this transformation.

13.17.21.5 [static] integer r0 – "unrotated".

Return: integer — The angle/mirror code for this transformation.

13.17.21.6 [static] integer r180 – "rotated by 180 degree counterclockwise".

Return: integer The angle/mirror code for this transformation.

13.17.21.7 [static] integer r270 – "rotated by 270 degree counterclockwise".

Return: integer The angle/mirror code for this transformation.

13.17.21.8 [static] integer r90 – "rotated by 90 degree counterclockwise".

Return: integer The angle/mirror code for this transformation.

13.17.22 [const] double mag
Gets the magnification.

Return: integer The angle/mirror code for this transformation.

13.17.23 mag=(double m)
Sets the magnification.

Input: m The new magnification.

13.17.24 mirror=(boolean)
Sets the mirror flag.

"mirroring" describes a reflection at the x-axis which is included in the transformation prior to rotation.

Input: boolean The new mirror flag.

13.17.25 [static] DCplxTrans new
Creates a unit transformation.

13.17.26 [static] DCplxTrans new(f)
Conversion constructor from a fix-point transformation.

A synonym of: [static] DCplxTrans new_f(f).

This constructor will create a transformation with a fixpoint transformation but no displacement.

Input: f The rotation/mirror code (r0 .. m135 constants).

13.17.27 [static] DCplxTrans new(double m)
Constructor from a magnification.

A synonym of: [static] DCplxTrans new_m(double m).

Creates a magnifying transformation without displacement and rotation given the magnification m.

Input: double m The magnification.

13.17.28 [static] DCplxTrans new(Trans t, double m)
Constructor from a simple transformation and a magnification.

A synonym of: [static] DCplxTrans new_tm(Trans t, double m).

Creates a magnifying transformation from a simple transformation and a magnification.

13.17.29 [static] DCplxTrans new(Trans t)
Constructor from a simple transformation alone.

A synonym of: [static] DCplxTrans new_t(Trans t).

Creates a magnifying transformation from a simple transformation and a magnification of 1.0.

13.17.30 [static] DCplxTrans new(double m, double r, boolean, DPoint u)
The standard constructor using magnification, angle, mirror flag and displacement.

A synonym of: [static] DCplxTrans new_mrmu(double m, double r, boolean, DPoint u).

The sequence of operations is: magnification, mirroring at x axis, rotation, application of displacement.

Input:	double m	The magnification.
	double r	The rotation angle in units of degree.
	boolean	True, if mirrored at x axis.
	u	The displacement.

13.17.31 [const] integer rot
Returns the respective rotation code if possible.

If this transformation is orthogonal (is_ortho () == true), then this method will return the corresponding fix-point transformation, not taking into account magnification and displacement. If the transformation is not orthogonal, the result reflects the quadrant the rotation goes into with the guarantee to reproduce the correct quadrant in the exact case.

13.17.32 [const] DTrans s_trans
Extract the simple transformation part.

The simple transformation part does not reflect magnification not arbitrary angles. On the angle contribution up to a multiple of 90 degree is reflected.

13.17.33 [const] string to_s
String conversion.

Return:	string	The resulting string.

13.17.34 [const] DPoint trans(Point p)
The transformation of a point.

The trans method transforms the given point. q = t(p).

Input:	p	The point to transform.
Return:	DPoint	The transformed point.

13.18 Class DEdge (version 0.21)
An edge class with double (floating-point) coordinates.

An edge is a connection between points, usually participating in a larger context such as a polygon. An edge has a defined direction (from p1 to p2).

Method Overview

from_iedge	Construct a floating-point coordinate edge from an integer coordinate edge
new	Default constructor: creates a degenerated edge 0,0 to 0,0.
new	Constructor with two coordinates given as single values.
new	Constructor with two points.
<	Less operator.
==	Equality test.
!=	Inequality test.
moved	Returns the moved edge.
enlarged	Returns the enlarged edge.
transformed	Transform the edge.
transformed_cplx	Transform the edge.
move	Moves the edge.
enlarge	Enlarges the edge.
p1	The first point.
p2	The second point.
dx	The horizontal extend of the edge.
dy	The vertical extend of the edge.
x1	Shortcut for p1.x.
y1	Shortcut for p1.y.
x2	Shortcut for p2.x.
y2	Shortcut for p2.y.
dx_abs	The absolute value of the horizontal extend of the edge.
dy_abs	The vertical extend of the edge.
bbox	Return the bounding box of the edge.
is_degenerate?	Test for degenerated edge.
length	The length of the edge.
sq_length	The square of the length of the edge.
ortho_length	The orthogonal length of the edge (“manhattan-length”).
to_s	Convert to a string.
is_parallel?	Test for being parallel.
contains?	Test whether a point is on an edge.
contains_excl?	Test whether a point is on an edge excluding the endpoints.
coincident?	Coincidence check.
intersect?	Intersection test.
intersection_point	Returns the intersection point of two edges.
distance	Distance between the edge and a point.
side_of	Indicates at which side the point is located relative to the edge.
distance_abs	Absolute distance between the edge and a point.
swap_points	Swap the points of the edge.
crossed_by?	Check, if an edge is cut by a line (given by an edge).
crossing_point	Returns the crossing point on two edges.
assign	Assign the contents of another object to self.
dup	Creates a copy of self.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.18.1 [const] boolean !=(DEdge e)
Inequality test.

Input:	e	The object to compare against.
Return:	true	Inequality.
	false	???.

13.18.2 [const] boolean <(DEdge e)
Less operator.

Input:	e	The object to compare against.
Return:	true	The edge is 'less' than the other edge with respect to first and second point.
	false	???.

13.18.3 [const] boolean ==(DEdge e)
Equality test.

Input:	e	The object to compare against.
Return:	true	Equality.
	false	???.

13.18.4 assign(DEdge other)
Assign the contents of another object to self.

This method assigns the contents of another object to self. This is a deep copy that does not only copy the reference but the actual content.

13.18.5 [const] DBox bbox
Return the bounding box of the edge.

Return:	DBox	The bounding box of the edge.

13.18.6 [const] boolean coincident?(DEdge e)
Coincidence check.

Checks whether a edge is coincident with another edge. Coincidence is defined by being parallel and that at least one point of one edge is on the other edge.

Input:	e	The edge to test with.
Return:	true	The edges are coincident.
	false	???.

13.18.7 [const] boolean contains?(DPoint p)
Test whether a point is on an edge.

A point is on a edge if it is on (or at least closer than a grid point to) the edge.

Input:	p	The point to test with the edge.
Return:	true	The is on the edge.
	false	???.

13.18.8 [const] boolean contains_excl?(DPoint p)
Test whether a point is on an edge excluding the endpoints.

A point is on a edge if it is on (or at least closer than a grid point to) the edge.

Input:	p	The point to test with the edge.
Return:	true	The is on the edge but not equal p1 or p2.
	false	???.

13.18.9 [const] boolean crossed_by?(DEdge e)
Check, if an edge is cut by a line (given by an edge).

This method returns true if p1 is in one semispace while p2 is in the other, or one of them is on the line through the edge "e".

Input:	e	The edge representing the line that the edge must be crossing.
Return:	true	The line crosses the edge.
	false	???.

13.18.10 [const] DPoint crossing_point(DEdge e)
Returns the crossing point on two edges.

This method delivers the point where the given edge (self) crosses the line given by the edge in argument "e" If self does not cross this line, the result is undefined. See crossed_by? for a description of the crossing predicate.

This method has been introduced in version 0.19.

Input:	e	The edge representing the line that self must be crossing.
Return:	DPoint	The point where self crosses the line given by "e".

13.18.11 destroy
Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.18.12 [const] boolean destroyed
Tell, if the object was destroyed.

Return:	true	The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
	false	The object is still alive.

13.18.13 [const] double distance(DPoint p)
Distance between the edge and a point.

Returns the distance between the edge and the point. The distance is measured by projecting the point onto the line through the edge. If the edge is degenerated, the distance is not defined.

Input:	p	The point to test.
Return:	-1	The point is "left" of the edge.
	0	The point is on the edge.
	1	The point is "right" of the edge.

13.18.14 [const] double distance_abs(DPoint p)
Absolute distance between the edge and a point.

Input:	p	The point to test.
Return:	unsigned integer	The distance as unsigned double integer.

13.18.15 [const] DEdge dup
Creates a copy of self.

Return:	DEdge	The copy of self.

13.18.16 [const] double dx
The horizontal extend of the edge.

13.18.17 [const] double dx_abs
The absolute value of the horizontal extend of the edge.

13.18.18 [const] double dy
The vertical extend of the edge.

13.18.19 [const] double dy_abs
The vertical extend of the edge.

13.18.20 ref DEdge enlarge(DPoint p)
Enlarges the edge.

Enlarges the edge by the given distance and returns the enlarged edge. The edge is overwritten.

Input:	p	The distance to move the edge points.
Return:	ref	Reference to the enlarged edge.

13.18.21 [const] DEdge enlarged(DPoint p)
Returns the enlarged edge.

Enlarges the edge by the given offset and returns the moved edge. The edge is not modified. Enlargement means that the first point is shifted by -p, the second by p.

Input:	p	The distance to enlarge the edge points.
Return:	DEdge	The enlarged edge.

13.18.22 [static] DEdge from_iedge(Edge int_edge)
Construct a floating-point coordinate edge from an integer coordinate edge.

Create a floating-point coordinate edge from an integer edge.

Input:	int_edge	A integer coordinate edge.
Return:	DEdge	The resulting floating-point coordinate edge.

13.18.23 [const] boolean intersect?(DEdge e)
Intersection test.

Input:	e	The edge to test.
Return:	true	The edges intersect. Two edges intersect if they share at least one point. If the edges coincide, they also intersect. For degenerated edges, the intersection is mapped to point containment tests.
	false	The edges does not intersect.

13.18.24 [const] DPoint intersection_point(DEdge e)
Returns the intersection point of two edges.

This method delivers the intersection point. If the edges do not intersect, the result is undefined.

This method has been introduced in version 0.19.

Input:	e	The edge to test.
Return:	DPoint	The point where the edges intersect.

13.18.25 [const] boolean is_degenerate?
Test for degenerated edge.

Return:	true	This edge is degenerated, that means end and start point are identical.
	false	End and start point are different.

13.18.26 [const] boolean is_parallel?(DEdge e)
Test for being parallel.

Input:	e	The edge to test against.
Return:	true	The edges are parallel.
	false	The edges are not parallel.

13.18.27 [const] double length
Get the length of the edge.

Return:	double	The length of the edge.

13.18.28 ref DEdge move(DPoint p)
Moves the edge.

Moves the edge by the given offset and returns the moved edge. The edge is overwritten.

Input:	p	The distance to move the edge.
Return:	ref	Reference to the enlarged edge.

13.18.29 [const] DEdge moved(DPoint p)
Returns the moved edge.

Moves the edge by the given offset and returns the moved edge. The edge is not modified.

Input:	p	The distance to move the edge.
Return:	DEdge	The enlarged edge.

13.18.30 [static] DEdge new
Default constructor: creates a degenerated edge 0,0 to 0,0.

13.18.31 [static] DEdge new(double x1, double y1, double x2, double y2)
Constructor with two coordinates given as single values.

A synonym for:[static] DEdge new_xyxy(double x1, double y1, double x2, double y2).par Four values, denotes two coordinates, are given to create a new edge.

Input:	double x1	The x part of the first coordinate.
	double y1	The y part of the first coordinate.
	double x2	The x part of the second coordinate.
	double y2	The y part of the second coordinate.
Return:	DEdge	The resulting edge.

13.18.32 [static] DEdge new(DPoint p1 DPoint p2)
Constructor with two points.

A synonym for:[static] DEdge new_pp(DPoint p1 DPoint p2).

Two points are given to create a new edge.

Input:	DPoint p1	The first point.
	DPoint p2	The second point.
Return:	DEdge	The resulting edge.

13.18.33 [const] double ortho_length
The orthogonal length of the edge "manhattan-length").

Return:	double	The orthogonal length equals to $abs(dx) + abs(dy)$.

13.18.34 [const] const ref DPointp1
The first point.

13.18.35 [const] const ref DPointp2
The second point.

13.18.36 [const] integer side_of(DPoint p)
Indicates at which side the point is located relative to the edge.

Input:	p	The point to test.
Return:	-1	The point is "left" of the edge.
	0	The point is on the edge.
	1	The point is "right" of the edge.

13.18.37 [const] double sq_length
The square of the length of the edge.

13.18.38 swap_points
Swap the points of the edge.

13.18.39 [const] string to_s
Convert to a string.

Return: string The resulting string.

13.18.40 [const] DEdge transformed(DTrans t)
Transform the edge.

Transforms the edge with the given complex transformation. Does not modify the edge but returns the transformed edge.

Input: t The transformation to apply.
Return: DEdge The transformed edge.

13.18.41 [const] DEdge transformed_cplx(DCplxTrans t)
Transform the edge.

Transforms the edge with the given complex transformation. Does not modify the edge but returns the transformed edge.

Input: t The transformation to apply.
Return: DEdge The transformed edge.

13.18.42 [const] double x1
Shortcut for p1.x.

13.18.43 [const] double x2
Shortcut for p2.x.

13.18.44 [const] double y1
Shortcut for p1.y.

13.18.45 [const] double y2
Shortcut for p2.y.

13.19 Class DPath (version 0.21)
An path class with double (floating-point) coordinates.

A path consists of an sequence of line segments forming the 'spine' of the path and a width. In addition, the starting point can be drawn back by a certain extent (the 'begin extension') and the end point can be pulled forward somewhat (by the 'end extension'). A path may have round ends for special purposes.

Method Overview

new	Default constructor: creates an empty (invalid) path with width 0.
new	Constructor given the points of the path's spine and the width.
new	Constructor given the points of the path's spine, the width and the extensions.
new	Constructor given the points of the path's spine, the width, the extensions and the round end flag.
<	Less operator.
==	Equality test.
!=	Inequality test.
points=	Set the points of the path.
each_point	Get the points that make up the path's spine.
points	Get the number of points.
width=	Set the width.
width	Get the width.
bgn_ext=	Set the begin extension.
bgn_ext	Get the begin extension.
end_ext=	Set the end extension.
end_ext	Get the end extension.
round=	Set the 'round ends' flag.
is_round?	Tell, if the path has round ends.
move	Moves the path.
moved	Returns the moved path.
transformed	Transform the path.
transformed_cplx	Transform the path.
to_s	Convert to a string.
simple_polygon	Convert the path to a simple polygon.
polygon	Convert the path to a polygon.
bbox	Return the bounding box of the path.
from_ipath	Construct a floating-point coordinate path from an integer coordinate one.
assign	Assign the contents of another object to self.
dup	Creates a copy of self.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.19.1 [const] boolean !=(DPath p)
Inequality test.

Input:	p	The object to compare against.
Return:	true	Inequality.
	false	???.

13.19.2 [const] boolean <(DPath p)
Less operator.

This operator is provided to establish some, not necessarily a certain sorting order.

Input:	p	The object to compare against.
Return:	true	The path is less then the argument path.
	false	The path is greater then the argument path.

13.19.3 [const] boolean ==(DPath p)
Equality test.

Input:	p	The object to compare against.
Return:	true	Equality.
	false	???.

13.19.4 assign(DPath other)
Assign the contents of another object to self.

This method assigns the contents of another object to self. This is a deep copy that does not only copy the reference but the actual content.

13.19.5 [const] DBox bbox
Return the bounding box of the path.

Return:	DBox	The bounding box.

13.19.6 [const] double bgn_ext
Get the begin extension.

Return:	double	The begin extension.

13.19.7 bgn_ext=(double)
Set the begin extension.

Input:	double	The begin extension.

13.19.8 destroy
Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.19.9 [const] boolean destroyed
Tell, if the object was destroyed.

Return:	true	The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
	false	The object is still alive.

13.19.10 [const] DPath dup
Creates a copy of self.

Return: DPath The copy of self.

13.19.11 [const] yield DPoint each_point
Get the points that make up the path's spine.

Return: yield The points that make up the path's spine.

13.19.12 [const] double end_ext
Get the end extension.

Return: double The end extension.

13.19.13 end_ext=(double)
Set the end extension.

Input: double The end extension.

13.19.14 [static] DPath from_ipath(Path int_path)
Construct a floating-point coordinate path from an integer coordinate one.

This method has been added in version 0.15.

13.19.15 [const] boolean is_round?
Tell, if the path has round ends.

Return: true The path has round ends.
false The path has other ends.

13.19.16 ref DPath move(DPoint p)
Moves the path.

Moves the path by the given offset and returns the reference of the moved path. The path is overwritten.

Input: p The distance to move the path.
Return: ref The reference of the moved path.

13.19.17 [const] DPath moved(DPoint p)
Returns the moved path.

Moves the path by the given offset and returns the reference of the moved path. The path is not modified.

Input: p The distance to move the path.
Return: DPath The moved path.

13.19.18 [static] DPath new
Default constructor: creates an empty (invalid) path with width 0.

Return: DPath The empty (invalid) path.

13.19.19 [static] DPath new(DPoint pts[], double width)
Constructor given the points of the path's spine and the width.

A synonym for: [static] DPath new_pw(DPoint pts[], double width).

Input:	pts[]	The points forming the spine of the path.
	double width	The width of the path.
Return:	DPath	The resulting path.

13.19.20 [static] DPath new(DPoint pts[], double width, double bgn_ext, double end_ext)
Constructor given the points of the path's spine, the width and the extensions.

A synonym for: [static] DPath new_pwx(DPoint pts[], double width, double bgn_ext, double end_ext).

Input:	pts[]	The points forming the spine of the path.
	double width	The width of the path.
	double bgn_ext	The begin extension of the path.
	double end_ext	The end extension of the path.
Return:	DPath	The resulting path.

13.19.21 [static] DPath new(DPoint pts[], double width, double bgn_ext, double end_ext, boolean round)
Constructor given the points of the path's spine, the width, the extensions and the round end flag.

A synonym for: [static] DPath new_pwxr(DPoint pts[], double width, double bgn_ext, double end_ext, boolean round).

Input:	pts[]	The points forming the spine of the path.
	double width	The width of the path.
	double bgn_ext	The begin extension of the path.
	double end_ext	The end extension of the path.
	boolean round	If this flag is true, the path will get rounded ends.
Return:	DPath	The resulting path.

13.19.22 [const] unsigned points
Get the number of points.

Return:	unsigned	The number of points.

13.19.23 points=(DPoint pts[])
Set the points of the path.

Input:	pts[]	An area of points forming the spine of the path.

13.19.24 [const] DPolygon polygon
Convert the path to a polygon.

The returned polygon is not guaranteed to be non-self overlapping. This may happen if the path overlaps itself or contains very short segments.

Return:	DPolygon	The resulting polygon.

13.19.25 round=(boolean)
Set the "round ends" flag.

Input:	true	"round ends".
	false	Other ends.

13.19.26 [const] DSimplePolygon simple_polygon
Convert the path to a simple polygon.

The returned polygon is not guaranteed to be non-selfoverlapping. This may happen if the path overlaps itself or contains very short segments.

Return: DSimplePolygon The resulting polygon.

13.19.27 [const] string to_s
Convert to a string.

Return: string The resulting string.

13.19.28 [const] DPath transformed(DTrans t)
Transform the path.

Transforms the path with the given transformation. Does not modify the path but returns the transformed path.

Input:	t	The transformation to apply.
Return:	DPath	The transformed path.

13.19.29 [const] DPath transformed_cplx(DCplxTrans t)
Transform the path.

Transforms the path with the given complex transformation. Does not modify the path but returns the transformed path.

Input:	t	The transformation to apply.
Return:	DPath	The transformed path.

13.19.30 [const] double width
Get the width.

Return: double The width of the path.

13.19.31 width=(double)
Set the width.

Input: double The width of the path.

13.20 Class DPoint (version 0.21)
A point class with double (floating-point) coordinates.

Method Overview

from_ipoint	Create a floating-point coordinate point from an integer coordinate point.
new	Default constructor: creates a point at 0,0.
new	Constructor for a point from two coordinate values.
+	Add one point to another.
-	Subtract one point from another.
<	"less" comparison operator.
==	Equality test operator.
!=	Inequality test operator.
x	Accessor to the x coordinate.
y	Accessor to the y coordinate.
x=	Write accessor to the x coordinate.
y=	Write accessor to the y coordinate.
*****	Scaling by some factor.
distance	The euclidean distance to another point.
sq_distance	The square euclidean distance to another point.
to_s	String conversion.
assign	Assign the contents of another object to self.
dup	Creates a copy of self.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.20.1 [const] boolean !=(DPoint p)
Inequality test operator.

Input:	p	The given floating-point coordinate point.
Return:	true	This and the given point are unequal.
	false	This and the given point are equal.

13.20.2 [const] DPoint *(double f)
Scaling by some factor.

Input:	double f	The given floating-point scaling factor.
Return:	DPoint	The scaled floating-point coordinate point.

13.20.3 [const] DPoint +(DPoint p)
Add one point to another.

Add point p to self by adding the coordinates.

Input:	p	The given floating-point coordinate point.
Return:	DPoint	The resulting floating-point coordinate point.

13.20.4 [const] DPoint –(DPoint p)
Subtract one point to another.

Subtract point p from self by subtracting the coordinates.

Input:	p	The given floating-point coordinate point.
Return:	DPoint	The resulting floating-point coordinate point.

13.20.5 [const] boolean <(DPoint p)
"less" comparison operator.

This operator is provided to establish a sorting order.

Input:	p	The given floating-point coordinate point.
Return:	true	This point is 'less'.
	false	This point is 'greater'.

13.20.6 [const] boolean ==(DPoint p)
Equality test operator.

Input:	p	The given floating-point coordinate point.
Return:	true	This point and the given point are equal.
	false	This point and the given point are unequal.

13.20.7 assign(DPoint other)
Assign the contents of another object to self.

This method assigns the contents of another object to self. This is a deep copy that does not only copy the reference but the actual content.

13.20.8 destroy
Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.20.9 [const] boolean destroyed
Tell, if the object was destroyed.

Return:	true	The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
	false	The object is still alive.

13.20.10 [const] double distance(DPoint d)
The euclidean distance to another point.

Input:	d	The other point to compute the distance to.
Return:	double	The euclidean distance.

13.20.11 [const] DPoint dup

Creates a copy of self.

Return: DPoint The copy of self.

13.20.12 [static] DPoint from_ipoint(Point p)

Create a floating-point coordinate point from an integer coordinate point.

Input: p The integer coordinate point.
Return: DPoint The created floating-point coordinate point.

13.20.13 [static] DPoint new

Default constructor: creates a point at 0,0.

Return: DPoint The new floating-point coordinate point at 0,0.

13.20.14 [static] DPoint new(double x, double y)

Constructor for a point from two coordinate values.

Input: double x The floating-point x part of the coordinate.
double y The floating-point y part of the coordinate.
Return: DPoint The new floating-point coordinate point.

13.20.15 [const] double sq_distance(DPoint d)

The square euclidean distance to another point.

Input: d The other point to compute the distance to.
Return: double The square euclidean distance.

13.20.16 [const] string to_s

String conversion.

Return: string The floating-point coordinate point as string.

13.20.17 [const] double x

Accessor to the x part of the coordinate.

Return: integer The x part of the floating-point coordinate point.

13.20.18 x=(double)

Write accessor to the x part of the coordinate.

Input: integer The x part of the floating-point coordinate point.

13.20.19 [const] double y

Accessor to the y part of the coordinate.

Return: integer The y part of the floating-point coordinate point.

13.20.20 y=(double)
Write accessor to the y part of the coordinate.

Input: integer The y part of the floating-point coordinate point.

13.21 Class DPolygon (version 0.21) A polygon class with double (floating-point) coordinates.

A polygon consists of an outer hull and zero to many holes. Each contour consists of several points. The point list is normalized such that the leftmost, lowest point is the first one. The orientation is normalized such that the orientation of the hull contour is clockwise, while the orientation of the holes is counter-clockwise.

It is in no way checked that the contours are not over-lapping. This must be ensured by the user of the object when filling the contours.

Method Overview

new	Default constructor: creates an empty (invalid) polygon.
new	Constructor given the points of the polygon hull.
new	Constructor converting a box to a polygon.
<	Less operator.
==	Equality test.
!=	Inequality test.
hull=	Set the points of the hull of polygon.
assign_hole	Set the points of the given hole of the polygon.
points	Get the total number of points (hull plus holes).
point_hull	Get a specific point of the hull@args p.
point_hole	Get a specific point of a hole@args n,p.
points_hull	Get the number of points of the hull.
points_hole	Get the number of points of the given hole.
insert_hole	Insert a hole with the given points.
each_point_hull	Iterate over the points that make up the hull.
each_point_hole	Iterate over the points that make up the nth hole.
size	Sizing (biasing).
size	Sizing (biasing).
holes	Get the number of holes.
each_edge	Iterate over the edges that make up the polygon.
inside	Test, if the given point is inside the polygon.
compress	Compress the polygon.
move	Moves the polygon.
moved	Returns the moved polygon.
transformed	Transform the polygon.
transformed_cplx	Transform the polygon with a complex transformation.
to_s	Convert to a string.
area	The area of the polygon.
bbox	Return the bounding box of the polygon.
from_ipoly	Construct a floating-point coordinate polygon from an integer coordinate one.
assign	Assign the contents of another object to self.
dup	Creates a copy of self.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.21.1 [const] boolean !=(DPolygon p)
Inequality test.

Input:	p	The object to compare against.
Return:	true	Inequality.
	false	???.

13.21.2 [const] boolean <(DPolygon p)
Less operator.

This operator is provided to establish some, not necessarily a certain sorting order.

Input:	p	The object to compare against.
Return:	true	This polygon is less than the given one.
	false	???.

13.21.3 [const] boolean ==(DPolygon p)
Equality test.

Input:	p	The object to compare against.
Return:	true	The polygons are equal.
	false	???.

13.21.4 [const] double area
The area of the polygon.

The area is correct only if the polygon is not self-overlapping and oriented clockwise.

Return:	double	The area of the polygon.

13.21.5 assign(DPolygon other)
Assign the contents of another object to self.

This method assigns the contents of another object to self. This is a deep copy that does not only copy the reference but the actual content.

13.21.6 assign_hole(unsigned, DPoint p[])
Set the points of the given hole of the polygon.

If the hole index is not valid, this method does nothing.

This method was introduced in version 0.18.

Input:	unsigned	The index of the hole to which the points should be assigned.
	p[]	An array of points to assign to the polygon's hole.

13.21.7 [const] const refDBox bbox
Return the bounding box of the polygon.

13.21.8 compress(boolean)
Compress the polygon.

Removes redundant points from the polygon, such as points being on a line formed by two other points.

Input: true — Additionally removes points if the two adjacent edges form a spike.
false — Basic behavior.

13.21.9 destroy
Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.21.10 [const] boolean destroyed
Tell, if the object was destroyed.

Return: true — The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
false — The object is still alive.

13.21.11 [const] DPolygon dup
Creates a copy of self.

Return: DPolygon — The copy of self.

13.21.12 yield DEdge each_edge
Iterate over the edges that make up the polygon.

Return: yield The array of the edges that make up the polygon.

13.21.13 [const] yield DPoint each_point_hole(unsigned)
Iterate over the points that make up the nth hole.

Input: unsigned — The hole number, which must be equal or less than the number of holes (see holes)

13.21.14 [const] yield DPoint each_point_hull
Iterate over the points that make up the hull.

Return: yield The array of the points that make up the hull.

13.21.15 [static] DPolygon from_ipoly(Polygon int_poly)
Construct a floating-point coordinate polygon from an integer coordinate one.

This method has been added in version 0.15.

13.21.16 [const] unsigned holes
Get the number of holes.

Return: unsigned The number of holes.

13.21.17 hull=(DPoint p[])
Set the points of the hull of polygon.

A synonym for: assign_hull(DPoint p[]).

The 'assign_hull' variant is provided in analogy to 'assign_hole'.

Input: p[] An array of points to assign to the polygon's hull.

13.21.18 insert_hole(DPoint p[])
Insert a hole with the given points.

Input: p[] An array of points to insert as a new hole.

13.21.19 [const] boolean inside(DPoint p)
Test, if the given point is inside the polygon.

This tests works well only if the polygon is not self-overlapping and oriented clockwise.

Input: true The given point is inside the polygon.
false The given point is outside the polygon.

13.21.20 ref DPolygon move(DPoint p)
Moves the polygon.

Moves the polygon by the given offset and returns the reference of the moved polygon. The polygon is overwritten.

Input: p The distance to move the polygon.
Return: ref The reference of the moved polygon.

13.21.21 [const] DPolygon moved(DPoint p)
Returns the moved polygon.

Moves the polygon by the given offset and returns the moved polygon. The polygon is not modified.

Input: p The distance to move the polygon.
Return: DPolygon The moved polygon.

13.21.22 [static] DPolygon new
Default constructor: creates an empty (invalid) polygon.

13.21.23 [static] DPolygon new(DPoint p[])
Constructor given the points of the polygon hull.

A synonym for: [static] DPolygon new_p(DPoint p[]).

Input: p[] An array of points to insert as a new polygon hull.

13.21.24 [static] DPolygon new(DBox box)
Constructor converting a box to a polygon.

A synonym for: [static] DPolygon new_b(DBox box).

Input: box The box to convert to a polygon.

13.21.25 DPoint point_hole(unsigned n, unsigned p)
Get a specific point of a hole@args n,p.

This method was introduced in version 0.18.

Input: unsigned n — The index of the hole to which the points should be assigned.
unsigned p — The index of the point to get.
Return: DPoint — The specific hole point. If the index of the point or of the hole is not valid, a default value is returned.

13.21.26 DPoint point_hull(unsigned p)
Get a specific point of a hull@args p.

This method was introduced in version 0.18.

Input: unsigned p — The index of the point to get.
Return: DPoint — The specific hull point. If the index of the point is not a valid index, a default value is returned.

13.21.27 unsigned points
Get the total number of points (hull plus holes).

This method was introduced in version 0.18.

Return: unsigned The total number of points.

13.21.28 unsigned points_hole(unsigned n)
Get the number of points of the given hole.

The argument gives the index of the hole of which the number of points are requested. The index must be less than the number of holes, see holes.

Input: unsigned n The given hole.
Return: unsigned The number of points.

13.21.29 unsigned points_hull
Get the number of points of the hull.

Return: unsigned The number of points of the hull.

13.21.30 size(double dx, double dy, unsigned mode)
Sizing (biasing).

Shifts the contour outwards (dx,dy>0) or inwards (dx,dy<0). May create invalid (self-overlapping, reverse oriented) contours. The sign of dx and dy should be identical.

Input: double dx — The x value to shift the contour.
double dy — The y value to shift the contour.
0 — Bending angle cutoff occurs at greater than 0 degree.
1 — Bending angle cutoff occurs at greater than 45 degree.
2 — Bending angle cutoff occurs at greater than 90 degree.
3 — Bending angle cutoff occurs at greater than 135 degree.
4 — Bending angle cutoff occurs at greater than approximately 168 degree.
other — Bending angle cutoff occurs at greater than approximately 179 degree.

13.21.31 size(double d, unsigned mode)
Sizing (biasing).

Shifts the contour outwards (d>0) or inwards (d<0). May create invalid (self-overlapping, reverse oriented) contours.

Input:	double d	The distance to shift the contour in x and y direction.
	0	Bending angle cutoff occurs at greater than 0 degree.
	1	Bending angle cutoff occurs at greater than 45 degree.
	2	Bending angle cutoff occurs at greater than 90 degree.
	3	Bending angle cutoff occurs at greater than 135 degree.
	4	Bending angle cutoff occurs at greater than approximately 168 degree.
	other	Bending angle cutoff occurs at greater than approximately 179 degree.

13.21.32 string to_s
Convert to a string.

Return:	string	The resulting string.

13.21.33 [const] DPolygon transformed(DTrans t)
Transform the polygon.

Transforms the polygon with the given transformation. Does not modify the polygon but returns the transformed polygon.

Input:	t	The transformation to apply.
Return:	DPolygon	The transformed polygon.

13.21.34 [const] DPolygon transformed_cplx(DCplxTrans t)
Transform the polygon.

Transforms the polygon with the given transformation. Does not modify the polygon but returns the transformed polygon.

Input:	t	The transformation to apply.
Return:	DPolygon	The transformed polygon.

13.22 Class DSimplePolygon (version 0.21) A polygon class.

A simple polygon consists of an outer hull only. The contour consists of several points. The point list is normalized such that the leftmost, lowest point is the first one. The orientation is normalized such that the orientation of the hull contour is clockwise.

It is in no way checked that the contours are not over-lapping. This must be ensured by the user of the object when filling the contours.

Method Overview

new	Default constructor: creates an empty (invalid) polygon.
new	Constructor given the points of the simple polygon hull.
new	Constructor converting a box to a polygon.
==	Equality test.
!=	Inequality test.
points=	Set the points of the simple polygon.
point	Get a specific point.
points	Get the number of points.
each_point	Iterate over the points that make up the simple polygon.
each_edge	Iterate over the edges that make up the polygon.
inside	Test, if the given point is inside the polygon.
compress	Compress the polygon.
move	Moves the polygon.
moved	Returns the moved polygon.
transformed	Transform the polygon.
transformed_cplx	Transform the polygon with a complex transformation.
to_s	Convert to a string.
area	The area of the polygon.
bbox	Return the bounding box of the polygon.
from_ipoly	Construct a floating-point coordinate polygon from an integer coordinate one.
assign	Assign the contents of another object to self.
dup	Creates a copy of self.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.22.1 [const] boolean !=(DSimplePolygon p) Inequality test.

Input:	p	The object to compare against.
Return:	true	Inequality.
	false	???.

13.22.2 [const] boolean ==(DSimplePolygon p) Equality test.

Input:	p	The object to compare against.
Return:	true	The polygons are equal.
	false	???.

13.22.3 [const] double area
The area of the polygon.

The area is correct only if the polygon is not self-overlapping and oriented clockwise.

Return: double The area of the polygon.

13.22.4 assign(DSimplePolygon other)
Assign the contents of another object to self.

This method assigns the contents of another object to self. This is a deep copy that does not only copy the reference but the actual content.

13.22.5 [const] const refDBox bbox
Return the bounding box of the polygon.

13.22.6 compress(boolean)
Compress the polygon.

Removes redundant points from the polygon, such as points being on a line formed by two other points.

Input: true Additionally removes points if the two adjacent edges form a spike.
false Basic behavior.

13.22.7 destroy
Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.22.8 [const] boolean destroyed
Tell, if the object was destroyed.

Return: true The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
false The object is still alive.

13.22.9 [const] DSimplePolygon dup
Creates a copy of self.

Return: DSimplePolygon The copy of self.

13.22.10 yield DEdge each_edge
Iterate over the edges that make up the simple polygon.

Return: yield The array of the edges that make up the simple polygon.

13.22.11 [const] yield DPoint each_point
Iterate over the points that make up the simple polygon.

Return: yield The array of the points that make up the simple polygon.

13.22.12 [static] DSimplePolygon from_ipoly(SimplePolygon int_poly)
Construct a floating-point coordinate polygon from an integer coordinate one.

This method has been added in version 0.15.

13.22.13 [const] boolean inside(DPoint p)
Test, if the given point is inside the polygon.

This tests works well only if the polygon is not self-overlapping and oriented clockwise.

Input:	true	The given point is inside the polygon.
	false	The given point is outside the polygon.

13.22.14 ref DSimplePolygon move(DPoint p)
Moves the simple polygon.

Moves the simple polygon by the given offset and returns the reference of the moved polygon. The polygon is overwritten.

Input:	p	The distance to move the polygon.
Return:	ref	The reference of the moved polygon.

13.22.15 [const] DSimplePolygon moved(DPoint p)
Returns the moved polygon.

Moves the polygon by the given offset and returns the moved polygon. The polygon is not modified.

Input:	p	The distance to move the polygon.
Return:	DSimplePolygon	The moved polygon.

13.22.16 [static] DSimplePolygon new
Default constructor: creates an empty (invalid) polygon.

13.22.17 [static] DSimplePolygon new(DPoint p[])
Constructor given the points of the simple polygon.

A synonym for: [static] DSimplePolygon new_p(DPoint p[]).

Input:	p[]	An array of points to insert as a new polygon hull.

13.22.18 [static] DSimplePolygon new(DBox box)
Constructor converting a box to a polygon.

A synonym for: [static] DSimplePolygon new_b(DBox box).

Input:	box	The box to convert to a polygon.

13.22.19 DPoint point(unsigned p)
Get a specific point of a contour@args p.

This method was introduced in version 0.18.

Input: unsigned p The index of the point to get.
Return: DPoint The specific contour point. If the index of the point is not a valid index, a default value is returned.

13.22.20 unsigned points
Get the number of points.

Return: unsigned The number of points.

13.22.21 points=(DPoint p[])
Set the points of the simple polygon.

Input: p[] An array of points to assign to the simple polygon.

13.22.22 string to_s
Convert to a string.

Return: string The resulting string.

13.22.23 [const] DSimplePolygon transformed(DTrans t)
Transform the simple polygon.

Transforms the simple polygon with the given transformation. Does not modify the polygon but returns the transformed polygon.

Input: t The transformation to apply.
Return: DSimplePolygon The transformed simple polygon.

13.22.24 [const] DSimplePolygon transformed_cplx(DCplxTrans t)
Transform the simple polygon.

Transforms the simple polygon with the given transformation. Does not modify the polygon but returns the transformed polygon.

Input: t The transformation to apply.
Return: DSimplePolygon The transformed simple polygon.

13.23 Class DText (version 0.21) A text object.

A text object has a point (location), a text, a text transformation, a text size and a font id. Text size and font id are provided to be able to render the text correctly.

Method Overview

from_itext	Construct an floating-point coordinate text object from an integer coordinate text
new	Default constructor.
new	Constructor with string and transformation.
new	Constructor with string, transformation, text height and font.
string=	Assign a text string to this object.
string	Get the text string.
trans=	Assign a transformation (text position and orientation) to this object.
trans	Get the transformation.
size=	Set the text height of this object.
size	Get the text height.
font=	Set the font number.
font	Get the font number.
move	Moves the text by a certain distance.
moved	Returns the text moved by a certain distance.
transformed	Transform the text with the given simple transformation.
transformed_cplx	Transform the text with the given complex transformation.
<	Less operator.
!=	Equality test.
==	Inequality test.
to_s	Convert to a string.
assign	Assign the contents of another object to self.
dup	Creates a copy of self.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.23.1 [const] boolean !=(DText text) Inequality test.

Input:	DText text	The text object and the given text to compare against.
Return:	true	This text object and the given text are not equal.
	false	???.

13.23.2 [const] boolean <(DText t) Less operator.

This operator is provided to establish some, not necessarily a certain sorting order.

Input:	t	The object to compare against.
Return:	true	This object is less than the given one.
	false	???.

13.23.3 [const] boolean ==(DText text)
Equality test.

Input:	DText text	The object and the given text to compare against.
Return:	true	This text object and the given text are not equal.
	false	???.

13.23.4 assign(DText other)
Assign the contents of another object to self.

This method assigns the contents of another object to self. This is a deep copy that does not only copy the reference but the actual content.

13.23.5 destroy
Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.23.6 [const] boolean destroyed
Tell, if the object was destroyed.

Return:	true	The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
	false	The object is still alive.

13.23.7 [const] DText dup
Creates a copy of self.

Return:	DText	The copied text object.

13.23.8 integer font
Get the font number.

Return:	integer	The font number.

13.23.9 font=(integer)
Set the font number.

Input:	integer	The font number.

13.23.10 [static] DText from_itext(Text text)
Construct an floating-point coordinate text object from an integer coordinate text.

Input:	text	Integer coordinate text object.
Return:	DText	Floating-point coordinate text object.

13.23.11 ref DText move(DPoint p)
Moves the text by a certain distance.

Moves the text by a given offset and returns the moved text. Does not check for coordinate overflows.

Input:	p	The distance to move the text.
Return:	ref	The reference to the moved text object.

13.23.12 [const] DText moved(DPoint p)
Returns the text moved by a certain distance.

Moves the text by a given offset and returns the moved text. Does not modify *this. Does not check for coordinate overflows.

Input:	p	The distance to move the text.
Return:	DText	The moved text.

13.23.13 [static] DText new
Default constructor.

Creates a text with unit transformation and empty text.

13.23.14 [static] DText new(string, DTrans t)
Constructor with string and transformation.

A string and a transformation is provided to this constructor. The transformation specifies the location and orientation of the text object. In addition, the text height and font can be specified.

Input:	string	The text string.
	t	The transformation to apply.
Return:	DText	The new text object.

13.23.15 [static] DText new(string, DTrans t, double height, font_id)
Constructor with string, transformation, text height and font number.

A string and a transformation is provided to this constructor. The transformation specifies the location and orientation of the text object. In addition, the text height and font can be specified.

Input:	string	The text string.
	t	The transformation to apply.
	double height	The text height as double integer.
	font_id	The font number as integer.
Return:	DText	The new text object.

13.23.16 [const] double size
Get the text height.

Return:	integer	The font height as double integer.

13.23.17 size=(double)
Set the text height of this object.

Input:	integer	The text height as double integer.

13.23.18 [const] string string

Get the text string.

Return: string The text string.

13.23.19 string=(string)

Assign a text string to this object.

Input: string The text string.

13.23.20 string to_s

Convert to a string.

Return: string The resulting string.

13.23.21 [const] const ref DTrans trans

Get the transformation.

13.23.22 trans=(DTrans t)

Assign a transformation (text position and orientation) to this object.

Input: t The transformation to assign.

13.23.23 [const] DText transformed(DTrans t)

Transform the text with the given simple transformation.

Input: t The transformation to apply.
Return: DText The transformed text object.

13.23.24 [const] DText transformed_cplx(DCplxTrans t)

Transform the text with the given complex transformation.

Input: t The transformation to apply.
Return: DText The transformed text object.

13.24 Class DTrans (version 0.21) A simple transformation.

The simple transformation applies a displacement vector and a simple fix-point transformation. This version acts on double coordinates.

Method Overview

from_itrans	Conversion constructor from an integer transformation.
new	Creates a unit transformation.
new	Conversion constructor from a fix-point transformation.
new	The standard constructor using angle and mirror flag.
new	The standard constructor using angle and mirror flag and two coordinate values for displacement.
new	The standard constructor using a code rather than angle and mirror.
new	The standard constructor using a code rather than angle and mirror and two coordinate values for displacement.
new	The standard constructor using a displacement only.
new	The standard constructor using a displacement given as two coordinates.
inverted	Inversion.
invert	In-place inversion.
ctrans	The transformation of a distance.
trans	The transformation of a point.
*	Multiplication (concatenation) of transformations.
<	A sorting criterion.
==	Equality test.
!=	Inequality test.
to_s	String conversion.
disp	Accessor to the point.
rot	Returns the respective rotation code if possible.
angle	Gets the angle.
is_mirror?	Gets the mirror flag.
angle=	Sets the angle.
disp=	Sets the displacement.
mirror=	Sets the mirror flag.
rot=	Sets the angle/mirror code for the named transformation.
r0	"unrotated".
r90	"rotated by 90 degree counterclockwise".
r180	"rotated by 180 degree counterclockwise".
r270	"rotated by 270 degree counterclockwise".
m0	"mirrored at the x-axis".
m45	"mirrored at the 45 degree axis".
m90	"mirrored at the y (90 degree) axis".
m135	"mirrored at the 135 degree axis".
assign	Assign the contents of another object to self.
dup	Creates a copy of self.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.24.1 [const] boolean !=(DTrans)
Inequality test.

Input:	DTrans text	The object to compare against.
Return:	true	This object and the given one are not equal.
	false	???.

13.24.2 [const] DTrans *(DTrans t)
Multiplication (concatenation) of transformations.

The * operator returns self*t ("t is applied before this transformation").

Input:	t	The transformation to apply before.
Return:	DTrans	The modified transformation.

13.24.3 [const] boolean <(DTrans)
A sorting criterion.

Input:	e	The object to compare against.
Return:	true	The object is 'less' than the other.
	false	??.

13.24.4 [const] boolean ==(DTrans)
Equality test.

Input:	e	The object to compare against.
Return:	true	Equality.
	false	??.

13.24.5 [const] double angle
Gets the angle in units of 90 degree.

This value delivers the rotation component. In addition, a mirroring at the x axis may be applied before if the is_mirror? property is true.

Return:	integer	The rotation angle in units of 90 degree.

13.24.6 angle=(double a)
Sets the angle in units of 90 degree.

This method was introduced in version 0.20.

Input:	a	The new angle.

13.24.7 assign(DTrans other)
Assign the contents of another object to self.

This method assigns the contents of another object to self. This is a deep copy that does not only copy the reference but the actual content.

13.24.8 [const] double ctrans(d)
The transformation of a distance.

The ctransethod transforms the given distance: $e = t(d)$. For the simple transformations, there is no magnification and no modification of the distance therefore.

Input: d The distance to transform.
Return: double The transformed distance.

13.24.9 destroy
Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.24.10 [const]boolean destroyed
Tell, if the object was destroyed.

Return: true The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
false The object is still alive.

13.24.11 [const] const ref DPoint disp
Accessor to the point.

Return: ref The accessor to the point.

13.24.12 disp=(DPoint u)
Sets the displacement.

This method was introduced in version 0.20.

Input: u The new displacement.

13.24.13 [const] DTrans dup
Creates a copy of self.

Return: DTrans The copy of self.

13.24.14 [static] DTrans from_itrans(DTrans int_trans)
Conversion constructor from an integer coordinate transformation.

Input: int_trans The integer coordinate transformation.
Return: DTrans The floating-point coordinate transformation.

13.24.15 DTrans invert
In-place inversion.

Inverts the transformation and replaces this transformation by the inverted one.

Return: DTrans The inverted and replaced transformation.

13.24.16 [const] DTrans inverted
Inversion.

Return: DTrans — The inverted transformation.

13.24.17 [const] boolean is_mirror?
Gets the mirror flag.

Return: true — The transformation is composed of a mirroring at the x-axis followed by a rotation by the angle given by the angle property.
false — ???.

13.24.18 [static] integer m_*/r_*
Various angle/mirror codes for the named transformation.

13.24.18.1 [static] integer m0 – "mirrored at the x-axis".

Return: integer — The angle/mirror code for this transformation.

13.24.18.2 [static] integer m135 – "mirrored at the 135 degree axis".

Return: integer — The angle/mirror code for this transformation.

13.24.18.3 [static] integer m45 – "mirrored at the 45 degree axis".

Return: integer — The angle/mirror code for this transformation.

13.24.18.4 [static] integer m90 – "mirrored at the 90 degree axis".

Return: integer — The angle/mirror code for this transformation.

13.24.18.5 [static] integer r0 – "unrotated".

Return: integer — The angle/mirror code for this transformation.

13.24.18.6 [static] integer r180 – "rotated by 180 degree counterclockwise".

Return: integer — The angle/mirror code for this transformation.

13.24.18.7 [static] integer r270 – "rotated by 270 degree counterclockwise".

Return: integer — The angle/mirror code for this transformation.

13.24.18.8 [static] integer r90 – "rotated by 90 degree counterclockwise".

Return: integer — The angle/mirror code for this transformation.

13.24.19 [const] double mag
Gets the magnification.

Return: integer The angle/mirror code for this transformation.

13.24.20 mirror=(boolean)
Sets the mirror flag.

"mirroring" describes a reflection at the x-axis which is included in the transformation prior to rotation.

This method was introduced in version 0.20.

Input: boolean The new mirror flag.

13.24.21 [static] DTrans new
Creates a unit transformation.

13.24.22 [static] DTrans new(f)
Conversion constructor from a fix-point transformation.

A synonym of: [static] DTrans new_f(f).

This constructor will create a transformation with a fixpoint transformation but no displacement.

Input: f The rotation/mirror code (r0 .. m135 constants).

13.24.23 [static] DTrans new(rot, boolean, ref DPoint u)
The standard constructor using angle and mirror flag.

A synonym of: [static] DTrans new_rmu(rot, boolean, ref DPoint u).

The sequence of operations is: mirroring at x axis, rotation, application of displacement.

Input: rot The rotation in units of 90 degree.
boolean True, if mirrored at x axis.
u The displacement.

13.24.24 [static] DTrans new(f, double x, double y)
The standard constructor using a code rather than angle and mirror and two coordinate values for displacement.

A synonym of: [static] DTrans new_fxy(f, double x, double y).

The sequence of operations is: mirroring at x axis, rotation, application of displacement.

Input: f The rotation/mirror code (r0 .. m135 constants).
double x The horizontal displacement.
double y The vertical displacement.

13.24.25 [static] DTrans new(f, DPoint u)
The standard constructor using a code rather than angle and mirror.

A synonym of: [static] DTrans new_fu(f, DPoint u).

Input: f The rotation/mirror code (r0 .. m135 constants).
u The displacement.

13.24.26 [static] DTrans new(rot, boolean, double x, double y)
The standard constructor using angle and mirror flag and two coordinate values for displacement.

A synonym of: [static] DTrans new_rmxy(rot, boolean, double x, double y).

The sequence of operations is: mirroring at x axis, rotation, application of displacement.

Input:	rot	The rotation in units of 90 degree.
	boolean	True, if mirrored at x axis.
	double x	The horizontal displacement.
	double y	The vertical displacement.

13.24.27 [static] DTrans new(DPoint u)
The standard constructor using a displacement only.

A synonym of: [static] DTrans new_u(DPoint u).

Input:	u	The displacement.

13.24.28 [static] DTrans new(double x, double y)
The standard constructor using a displacement given as two coordinates.

Input:	double x	The horizontal displacement.
	double y	The vertical displacement.

13.24.29 [const] integer rot
Gets the angle/mirror code.

The angle/mirror code is one of the constants r0, r90, r180, r270, m0, m45, m90 and m135. rx is the rotation by an angle of x counter clockwise. mx is the mirroring at the axis given by the angle x (to the x-axis).

13.24.30 rot=(r)
Sets the angle/mirror code.

This method was introduced in version 0.20.

Input:	r	The new angle/rotation code (see rot property).

13.24.31 [const] string to_s
String conversion.

Return:	string	The string representing the object.

13.24.32 [const] DPoint trans(DPoint p)
The transformation of a point.

The trans method transforms the given point. q = t(p).

Input:	p	The point to transform.
Return:	DPoint	The transformed point.

13.25 Class DoubleValue (version 0.21) Encapsulate a floating point value.

This class is provided as a return value of InputDialog::get_double. By using an object rather than a pure value, an object with has_value?= false can be returned indicating that the "Cancel" button was pressed.

Method Overview

has_value?	True, if a value is present.
to_f	Get the actual value (a synonym for value).
value	Get the actual value.
assign	Assign the contents of another object to self.
dup	Creates a copy of self.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.25.1 assign(DoubleValue other) Assign the contents of another object to self.

This method assigns the contents of another object to self. This is a deep copy that does not only copy the reference but the actual content.

13.25.2 destroy Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.25.3 [const] boolean destroyed Tell, if the object was destroyed.

Return:	true	The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
	false	The object is still alive.

13.25.4 [const] DoubleValue dup Creates a copy of self.

Return:	DoubleValue	The copy of self.

13.25.5 [const] boolean has_value? Query weather a value is present.

Return:	true	A value is present.
	false	Indication that the "Cancel" button was pressed.

13.25.6 [const] double to_f Get the actual value (a synonym for value).

Return:	double	The actual value.

13.25.7 [const] double value
Get the actual value.

Return: double The actual value.

13.26 Class Edge (version 0.21)
An edge class with integer coordinates.

An edge is a connection between points, usually participating in a larger context such as a polygon. An edge has a defined direction (from p1 to p2).

Method Overview

from_dedge	Construct an integer coordinate edge from a floating-point coordinate edge
new	Default constructor: creates a degenerated edge 0,0 to 0,0.
new	Constructor with two coordinates given as single values.
new	Constructor with two points.
<	Less operator.
==	Equality test.
!=	Inequality test.
moved	Returns the moved edge.
enlarged	Returns the enlarged edge.
transformed	Transform the edge.
transformed_cplx	Transform the edge.
transformed_cplx	Transform the edge.
move	Moves the edge.
enlarge	Enlarges the edge.
p1	The first point.
p2	The second point.
dx	The horizontal extend of the edge.
dy	The vertical extend of the edge.
x1	Shortcut for p1.x.
y1	Shortcut for p1.y.
x2	Shortcut for p2.x.
y2	Shortcut for p2.y.
dx_abs	The absolute value of the horizontal extend of the edge.
dy_abs	The vertical extend of the edge.
bbox	Return the bounding box of the edge.
is_degenerate?	Test for degenerated edge.
length	The length of the edge.
sq_length	The square of the length of the edge.
ortho_length	The orthogonal length of the edge (“manhattan-length”).
to_s	Convert to a string.
is_parallel?	Test for being parallel.
contains?	Test whether a point is on an edge.
contains_excl?	Test whether a point is on an edge excluding the endpoints.
coincident?	Coincidence check.
intersect?	Intersection test.
intersection_point	Returns the intersection point of two edges.
distance	Distance between the edge and a point.
side_of	Indicates at which side the point is located relative to the edge.
distance_abs	Absolute distance between the edge and a point.
swap_points	Swap the points of the edge.
crossed_by?	Check, if an edge is cut by a line (given by an edge).
crossing_point	Returns the crossing point on two edges.
assign	Assign the contents of another object to self.
dup	Creates a copy of self.
destroy	Explicitly destroy the object.

destroyed	Tell, if the object was destroyed.

13.26.1 [const] boolean !=(Edge e)
Inequality test.

Input:	e	The object to compare against.
Return:	true	Inequality.
	false	???.

13.26.2 [const] boolean <(Edge e)
Less operator.

Input:	e	The object to compare against.
Return:	true	The edge is "less" than the other edge with respect to first and second point.
	false	???.

13.26.3 [const] boolean ==(Edge e)
Equality test.

Input:	e	The object to compare against.
Return:	true	Equality.
	false	???.

13.26.4 assign(Edge other)
Assign the contents of another object to self.

This method assigns the contents of another object to self. This is a deep copy that does not only copy the reference but the actual content.

13.26.5 [const] Box bbox
Return the bounding box of the edge.

Return:	Box	The bounding box of the edge.

13.26.6 [const] boolean coincident?(Edge e)
Coincidence check.

Checks whether a edge is coincident with another edge. Coincidence is defined by being parallel and that at least one point of one edge is on the other edge.

Input:	e	The edge to test with.
Return:	true	The edges are coincident.
	false	???.

13.26.7 [const] boolean contains?(DPoint p)
Test whether a point is on an edge.

A point is on a edge if it is on (or at least closer than a grid point to) the edge.

Input:	p	The point to test with the edge.
Return:	true	The is on the edge.
	false	???.

13.26.8 [const] boolean contains_excl?(DPoint p)
Test whether a point is on an edge excluding the endpoints.

A point is on a edge if it is on (or at least closer than a grid point to) the edge.

Input:	p	The point to test with the edge.
Return:	true	The is on the edge but not equal p1 or p2.
	false	???.

13.26.9 [const] boolean crossed_by?(Edge e)
Check, if an edge is cut by a line (given by an edge).

This method returns true if p1 is in one semispace while p2 is in the other, or one of them is on the line through the edge "e".

Input:	e	The edge representing the line that the edge must be crossing.
Return:	true	The line crosses the edge.
	false	???.

13.26.10 [const] DPoint crossing_point(Edge e)
Returns the crossing point on two edges.

This method delivers the point where the given edge (self) crosses the line given by the edge in argument "e". If self does not cross this line, the result is undefined. See crossed_by? for a description of the crossing predicate.

This method has been introduced in version 0.19.

Input:	e	The edge representing the line that self must be crossing.
Return:	DPoint	The point where self crosses the line given by "e".

13.26.11 destroy
Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.26.12 [const] boolean destroyed
Tell, if the object was destroyed.

Return:	true	The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
	false	The object is still alive.

13.26.13 [const] integer distance(Point p)
Distance between the edge and a point.

Returns the distance between the edge and the point. The distance is measured by projecting the point onto the line through the edge. If the edge is degenerated, the distance is not defined.

Input:	p	The point to test.
Return:	-1	The point is "left" of the edge.
	0	The point is on the edge.
	1	The point is "right" of the edge.

13.26.14 [const] integer distance_abs(Point p)
Absolute distance between the edge and a point.

Input:	p	The point to test.
Return:	integer	The distance as unsigned double integer.

13.26.15 [const] Edge dup
Creates a copy of self.

Return:	Edge	The copy of self.

13.26.16 [const] integer dx
The horizontal extend of the edge.

Return: integer The horizontal extend of the edge.

13.26.17 [const] integer dx_abs
The absolute value of the horizontal extend of the edge.

Return: integer The absolute value of the horizontal extend of the edge.

13.26.18 [const] integer dy
The vertical extend of the edge.

Return: integer The vertical extend of the edge.

13.26.19 [const] integer dy_abs
The absolute value of the vertical extend of the edge.

Return: integer The absolute value of the vertical extend of the edge.

13.26.20 ref Edge enlarge(DPoint p)
Enlarges the edge.

Enlarges the edge by the given distance and returns the enlarged edge. The edge is overwritten.

Input:	p	The distance to move the edge points.
Return:	ref	Reference to the enlarged edge.

13.26.21 [const] Edge enlarged(DPoint p)
Returns the enlarged edge.

Enlarges the edge by the given offset and returns the moved edge. The edge is not modified. Enlargement means that the first point is shifted by -p, the second by p.

Input:	p	The distance to enlarge the edge points.
Return:	Edge	The enlarged edge.

13.26.22 [static] Edge from_dedge(DEdge double_edge)
Construct an integer coordinate edge from a floating-point coordinate edge.

Input:	double_edge	A floating-point coordinate edge.
Return:	Edge	The resulting integer coordinate edge.

13.26.23 [const] boolean intersect?(Edge e)
Intersection test.

Input:	e	The edge to test.
Return:	true	The edges intersect. Two edges intersect if they share at least one point. If the edges coincide, they also intersect. For degenerated edges, the intersection is mapped to point containment tests.
	false	The edges does not share any point.

13.26.24 [const] DPoint intersection_point(Edge e)
Returns the intersection point of two edges.

This method delivers the intersection point. If the edges do not intersect, the result is undefined.

This method has been introduced in version 0.19.

Input:	e	The edge to test.
Return:	DPoint	The point where the edges intersect.

13.26.25 [const] boolean is_degenerate?
Test for degenerated edge.

Return:	true	This edge is degenerated, that means end and start point are identical.
	false	End and start point are different.

13.26.26 [const] boolean is_parallel?(Edge e)
Test for being parallel.

Input:	e	The edge to test against.
Return:	true	The edges are parallel.
	false	The edges are not parallel.

13.26.27 [const] unsigned length
Get the length of the edge.

Return:	unsigned	The length of the edge.

13.26.28 ref Edge move(Point p)
Moves the edge.

Moves the edge by the given offset and returns the moved edge. The edge is overwritten.

Input:	p	The distance to move the edge.
Return:	ref	Reference to the enlarged edge.

13.26.29 [const] Edge moved(DPoint p)
Returns the moved edge.

Moves the edge by the given offset and returns the moved edge. The edge is not modified.

Input:	p	The distance to move the edge.
Return:	Edge	The enlarged edge.

13.26.30 [static] Edge new
Default constructor: creates a degenerated edge 0,0 to 0,0.

13.26.31 [static] Edge new(x1, y1, x2, y2)
Constructor with two coordinates given as single values.

A synonym for:[static] Edge new_xyxy(x1, y1, x2, y2).par Four values, denotes two coordinates, are given to create a new edge.

Input:	x1	The x part of the first coordinate.
	y1	The y part of the first coordinate.
	x2	The x part of the second coordinate.
	y2	The y part of the second coordinate.
Return:	Edge	The resulting edge.

13.26.32 [static] Edge new(Point p1 Point p2)
Constructor with two points.

A synonym for:[static] Edge new_pp(Point p1 Point p2).

Two points are given to create a new edge.

Input:	Point p1	The first point.
	Point p2	The second point.
Return:	Edge	The resulting edge.

13.26.33 [const] unsigned ortho_length
The orthogonal length of the edge ("manhattan-length").

Return:	unsigned	The orthogonal length equals to $abs(dx) + abs(dy)$.

13.26.34 [const] const ref Pointp1
The first point.

13.26.35 [const] const ref Pointp2
The second point.

13.26.36 [const] integer side_of(Point p)
Indicates at which side the point is located relative to the edge.

Input:	p	The point to test.
Return:	-1	The point is "left" of the edge.
	0	The point is on the edge.
	1	The point is "right" of the edge.

13.26.37 [const] long sq_length
The square of the length of the edge.

Return: long The square of the length of the edge.

13.26.38 swap_points
Swap the points of the edge.

13.26.39 [const] string to_s
Convert to a string.

Return: string The resulting string.

13.26.40 [const] Edge transformed(Trans t)
Transform the edge.

Transforms the edge with the given complex transformation. Does not modify the edge but returns the transformed edge.

Input:	t	The transformation to apply.
Return:	Edge	The transformed edge.

13.26.41 [const] Edge transformed_cplx(CplxTrans t)
Transform the edge.

Transforms the edge with the given complex transformation. Does not modify the edge but returns the transformed edge.

Input:	t	The transformation to apply.
Return:	DEdge	The transformed edge.

13.26.42 [const] Edge transformed_cplx(ICplxTrans t)
Transform the edge.

Transforms the edge with the given complex transformation. Does not modify the edge but returns the transformed edge.

This method has been introduced in version 0.18.

Input: t The transformation to apply.
Return: Edge The transformed edge (in this case an integer coordinate edge).

13.26.43 [const] double x1
Shortcut for p1.x.

Return: double The x coordinate value of the first point.

13.26.44 [const] double x2
Shortcut for p2.x.

Return: double The x coordinate value of the second point.

13.26.45 [const] double y1
Shortcut for p1.y.

Return: double The y coordinate value of the first point.

13.26.46 [const] double y2
Shortcut for p2.y.

Return: double The y coordinate value of the second point.

13.27 Class EdgeProcessor (version 0.21) The edge processor (boolean, size, merge).

The edge processor implements the boolean and edge set operations (size, merge). Because the edge processor might allocate resources which can be reused in later operations, it is implemented as an object that can be used several times.

Method Overview

simple_merge_p2e	Merge the given polygons in a simple "non-zero wrap count" fashion
simple_merge_p2p	Merge the given polygons in a simple "non-zero wrap count" fashion into polygons
simple_merge_e2e	Merge the given edges in a simple "non-zero wrap count" fashion
simple_merge_e2p	Merge the given edges in a simple "non-zero wrap count" fashion into polygons
merge_p2e	Merge the given polygons
merge_p2p	Merge the given polygons
size_p2e	Size the given polygons
size_p2p	Size the given polygons into polygons
size_p2e	Size the given polygons (isotropic)
size_p2p	Size the given polygons into polygons (isotropic)
boolean_p2e	Boolean operation for a set of given polygons, creating edges
boolean_p2p	Boolean operation for a set of given polygons, creating polygons
boolean_e2e	Boolean operation for a set of given edges, creating edges
boolean_e2p	Boolean operation for a set of given edges, creating polygons
mode_and	Boolean method's mode value for AND operation
mode_or	Boolean method's mode value for OR operation
mode_xor	Boolean method's mode value for XOR operation
mode_anotb	Boolean method's mode value for A NOT B operation
mode_bnota	Boolean method's mode value for B NOT A operation
assign	Assign the contents of another object to self.
dup	Creates a copy of self.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.27.1 assign(EdgeProcessor other) Assign the contents of another object to self.

This method assigns the contents of another object to self. This is a deep copy that does not only copy the reference but the actual content.

13.27.2 Edge[] boolean_e2e(Edge a[], Edge b[], mode) Boolean operation for a set of given edges, creating edges.

A synonym for: Edge[] boolean(Edge a[], Edge b[], mode).

This method computes the result for the given boolean operation on two sets of edges. The input edges must form closed contours where holes and hulls must be oriented differently. The input edges are processed with a simple non-zero wrap count rule as a whole.

The result is presented as a set of edges forming closed contours. Hulls are oriented clockwise while holes are oriented counter-clockwise.

Prior to version 0.21 this method was called "boolean". It was renamed to avoid ambiguities for empty input arrays. The old version is still available but deprecated.

Input:	a[]	The input edges (first operand).
	b[]	The input edges (second operand).
	mode	The boolean mode (one of the mode_...values).
Return:	Edge[]	The output edges.

13.27.3 Polygon[] boolean_e2p(Edge a[], Edge b[], mode, resolve_holes, min_coherence)
Boolean operation for a set of given edges, creating polygons.

Synonym for: Polygon[] boolean_to_polygon(Edge a[], Edge b[], mode, resolve_holes, min_coherence).

This method computes the result for the given boolean operation on two sets of edges. The input edges must form closed contours where holes and hulls must be oriented differently. The input edges are processed with a simple non-zero wrap count rule as a whole.

This method produces polygons on output and allows to fine-tune the parameters for that purpose.

Prior to version 0.21 this method was called "boolean_to_polygon". Is was renamed to avoid ambiguities for empty input arrays. The old version is still available but deprecated.

Input:	a[]	The input polygon (first operand).
	b[]	The input polygon (second operand).
	mode	The boolean mode (one of the mode_...values).
	resolve_holes	True, if holes should be resolved into the hull.
	min_coherence	True, if touching corners should be resolved into less connected contours.
Return:	Polygon[]	The output polygons.

13.27.4 Edge[] boolean_p2e(Polygon a[],Polygon b[], mode)
Boolean operation for a set of given polygons, creating edges.

A synonym for: Edge[] boolean(Polygon a[],Polygon b[], mode).

This method computes the result for the given boolean operation on two sets of polygons. The result is presented as a set of edges forming closed contours. Hulls are oriented clockwise while holes are oriented counter-clockwise.

This is a convenience method that bundles filling of the edges, processing with a Boolean operator and puts the result into an output vector.

Prior to version 0.21 this method was called "boolean". Is was renamed to avoid ambiguities for empty input arrays. The old version is still available but deprecated.

Input:	a[]	The input polygon (first operand).
	b[]	The input polygon (second operand).
	mode	The boolean mode (one of the mode_...values).
Return:	Edge[]	The output edges.

13.27.5 Polygon[] boolean_p2p(Polygon a[], Polygon b[], mode, resolve_holes, min_coherence)
Boolean operation for a set of given edges, creating polygons.

A synonym for: Polygon[] boolean_to_polygon(Polygon a[], Polygon b[], mode, resolve_holes, min_coherence).

This method computes the result for the given boolean operation on two sets of polygons. This method produces polygons on output and allows to fine-tune the parameters for that purpose.

This is a convenience method that bundles filling of the edges, processing with a Boolean operator and puts the result into an output vector.

Prior to version 0.21 this method was called "boolean_to_polygon". Is was renamed to avoid ambiguities for empty input arrays. The old version is still available but deprecated.

Input:	a[]	The input polygon (first operand).
	b[]	The input polygon (second operand).
	mode	The boolean mode (one of the mode_...values).
	resolve_holes	True, if holes should be resolved into the hull.
	min_coherence	True, if touching corners should be resolved into less connected contours.
Return:	Polygon[]	The output polygons.

13.27.6 destroy
Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.27.7 [const] boolean destroyed
Tell, if the object was destroyed.

Return:	true	The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
	false	The object is still alive.

13.27.8 [const] DText dup
Creates a copy of self.

Return:	EdgeProcessor	The copied text object.

13.27.9 Edge[] merge_p2e(Polygon in[], unsigned min_wc)
Merge the given polygons.

A synonym for: Edge[] merge(Polygon in[], unsigned min_wc).

In contrast to "simple_merge", this merge implementation considers each polygon individually before merging them. Thus self-overlaps are effectively removed before the output is computed and holes are correctly merged with the hull. In addition, this method allows to select areas with a higher wrap count which allows to compute overlaps of polygons on the same layer. Because this method merges the polygons before the overlap is computed, self-overlapping polygons do not contribute to higher wrap count areas.

The result is presented as a set of edges forming closed contours. Hulls are oriented clockwise while holes are oriented counter-clockwise.

Prior to version 0.21 this method was called "merge". Is was renamed to avoid ambiguities for empty input arrays. The old version is still available but deprecated.

Input:	in[]	The input polygons.
	min_wc	The minimum wrap count for output (0: all polygons, 1: at least two overlapping).
Return:	Edge[]	The output edges.

13.27.10 Polygon[] merge_p2p(Polygon in[], unsigned min_wc, resolve_holes, min_coherence)
Merge the given polygons.

A synonym for: Polygon[] merge_to_polygon(Polygon in[], unsigned min_wc, resolve_holes, min_coherence).

In contrast to "simple_merge", this merge implementation considers each polygon individually before merging them. Thus self-overlaps are effectively removed before the output is computed and holes are correctly merged with the hull. In addition, this method allows to select areas with a higher wrap count which allows to compute overlaps of polygons on the same layer. Because this method merges the polygons before the overlap is computed, self-overlapping polygons do not contribute to higher wrap count areas.

This method produces polygons and allows to fine-tune the parameters for that purpose.

Prior to version 0.21 this method was called "merge_to_polygon". Is was renamed to avoid ambiguities for empty input arrays. The old version is still available but deprecated.

Input:	in[]	The input polygons.
	min_wc	The minimum wrap count for output (0: all polygons, 1: at least two overlapping).
	resolve_holes	True, if holes should be resolved into the hull.
	min_coherence	True, if touching corners should be resolved into less connected contours.
Return:	Edge[]	The output edges.

13.27.11 [static] integer mode_and
boolean method's mode value for AND operation.

13.27.12 [static] integer mode_anotb
boolean method's mode value for A NOT B operation.

13.27.13 [static] integer mode_bnota
boolean method's mode value for B NOT A operation.

13.27.14 [static] integer mode_or
boolean method's mode value for OR operation.

13.27.15 [static] integer mode_xor
boolean method's mode value for XOR operation.

13.27.16 Edge[] simple_merge_e2e(Edge in[])
Merge the given edges in a simple "non-zero wrap count" fashion.

A synonym for: Edge[] simple_merge(Edge in[]).

The egdes provided must form valid closed contours. Contours oriented differently "cancel" each other. Overlapping contours are merged when the orientation is the same.

The result is presented as a set of edges forming closed contours. Hulls are oriented clockwise while holes are oriented counter-clockwise.

This is a convenience method that bundles filling of the edges, processing with a SimpleMerge operator and puts the result into an output vector.

Prior to version 0.21 this method was called "simple_merge". Is was renamed to avoid ambiguities for empty input arrays. The old version is still available but deprecated.

Input:	in[]	The input edges.
Return:	Edge[]	The output edges.

13.27.17 Polygon[] simple_merge_e2p(Edge in[], resolve_holes, min_coherence) Merge the given edges in a simple "non-zero wrap count" fashion into polygons.

A synonym for: Polygon[] simple_merge_to_polygon(Edge in[], resolve_holes, min_coherence).

The egdes provided must form valid closed contours. Contours oriented differently "cancel" each other. Overlapping contours are merged when the orientation is the same.

This method produces polygons and allows to fine-tune the parameters for that purpose.

This is a convenience method that bundles filling of the edges, processing with a SimpleMerge operator and puts the result into an output vector.

Prior to version 0.21 this method was called "simple_merge_to_polygon". Is was renamed to avoid ambiguities for empty input arrays. The old version is still available but deprecated.

Input:	in[]	The input edges.
	resolve_holes	True, if holes should be resolved into the hull.
	min_coherence	True, if touching corners should be resolved into less connected contours.
Return:	Polygon[]	The output polygons.

13.27.18 Edge[] simple_merge_p2e(Polygon in[]) Merge the given polygons in a simple "non-zero wrap count" fashion.

A synonym for: Edge[] simple_merge(Polygon in[]).par The wrap count is computed over all polygons, i.e. overlapping polygons may "cancel" if they have different orientation (since a polygon is oriented by construction that is not easy to achieve). The other merge operation provided for this purpose is "merge" which normalizes each polygon individually before merging them. "simple_merge" is somewhat faster and consumes less memory.

The result is presented as a set of edges forming closed contours. Hulls are oriented clockwise while holes are oriented counter-clockwise.

This is a convenience method that bundles filling of the edges, processing with a SimpleMerge operator and puts the result into an output vector.

Prior to version 0.21 this method was called "simple_merge". Is was renamed to avoid ambiguities for empty input arrays. The old version is still available but deprecated.

Input:	in[]	The input polygons.
Return:	Edge[]	The output edges.

13.27.19 Polygon[] simple_merge_p2p(Polygon in[], resolve_holes, min_coherence) Merge the given polygons in a simple "non-zero wrap count" fashion into polygons.

A synonym for: Polygon[] simple_merge_to_polygon(Polygon in[], resolve_holes, min_coherence).

The wrap count is computed over all polygons, i.e. overlapping polygons may "cancel" if they have different orientation (since a polygon is oriented by construction that is not easy to achieve). The other merge operation provided for this purpose is "merge" which normalizes each polygon individually before merging them. "simple_merge" is somewhat faster and consumes less memory.

This method produces polygons and allows to fine-tune the parameters for that purpose.

This is a convenience method that bundles filling of the edges, processing with a SimpleMerge operator and puts the result into an output vector.

Prior to version 0.21 this method was called "simple_merge_to_polygon". Is was renamed to avoid ambiguities for empty input arrays. The old version is still available but deprecated.

Input:	in[]	The input polygons.
	resolve_holes	True, if holes should be resolved into the hull.
	min_coherence	True, if touching corners should be resolved into less connected contours.
Return:	Polygon[]	The output polygons.

13.27.20 Edge[] size_p2e(Polygon in[], dx, dy, unsigned mode) Size the given polygons (anisotropic).

A synonym for: Edge[] size(Polygon in[], dx, dy, unsigned mode).

This method sizes a set of polygons. Before the sizing is applied, the polygons are merged. After that, sizing is applied on the individual result polygons of the merge step. The result may contain overlapping contours, but no self-overlaps.

dx and dy describe the sizing. A positive value indicates oversize (outwards) while a negative one describes undersize (inwards). The sizing applied can be chosen differently in x and y direction. In this case, the sign must be identical for both dx and dy.

The "mode" parameter describes the corner fill strategy. Mode 0 connects all corner segments directly. Mode 1 is the "octagon" strategy in which square corners are interpolated with a partial octagon. Mode 2 is the standard mode in which corners are filled by expanding edges unless these edges form a sharp bend with an angle of more than 90 degree. In that case, the corners are cut off. In Mode 3, no cutoff occurs up to a bending angle of 135 degree. Mode 4 and 5 are even more aggressive and allow very sharp bends without cutoff. This strategy may produce long spikes on sharply bending corners. The result is presented as a set of edges forming closed contours. Hulls are oriented clockwise while holes are oriented counter-clockwise.

Prior to version 0.21 this method was called "size". Is was renamed to avoid ambiguities for empty input arrays. The old version is still available but deprecated.

Input:	in[]	The input polygons.
	dx	The sizing value in x direction.
	dy	The sizing value in y direction.
	mode	The sizing mode (standard is 2).
Return:	Edge[]	The output edges.

13.27.21 Edge[] size_p2e(Polygon in[], d, unsigned mode) Size the given polygons (isotropic).

A synonym for: Edge[] size(Polygon in[], d, unsigned mode).

This method is equivalent to calling the anisotropic version with identical dx and dy.

Prior to version 0.21 this method was called "size". Is was renamed to avoid ambiguities for empty input arrays. The old version is still available but deprecated.

Input:	in[]	The input polygons.
	d	The sizing value in x and y direction.
	mode	The sizing mode (standard is 2).
Return:	Edge[]	The output edges.

13.27.22 Polygon[] size_p2p(Polygon in[], d, unsigned mode, resolve_holes, min_coherence) Size the given polygons into polygons (isotropic).

A synonym for: Polygon[] size_to_polygon(Polygon in[], d, unsigned mode, resolve_holes, min_coherence).

This method is equivalent to calling the anisotropic version with identical dx and dy.

Prior to version 0.21 this method was called "size_to_polygon". Is was renamed to avoid ambiguities for empty input arrays. The old version is still available but deprecated.

Input:	in[]	The input polygons.
	d	The sizing value in x and y direction.
	mode	The sizing mode (standard is 2).
	resolve_holes	True, if holes should be resolved into the hull.
	min_coherence	True, if touching corners should be resolved into less connected contours.
Return:	Polygon[]	The output polygons.

13.27.23 Polygon[] size_p2p(Polygon in[], dx, dy, unsigned mode, resolve_holes, min_coherence) Size the given polygons into polygons.

A synonym for: Polygon[] size_to_polygon(Polygon in[], dx, dy, unsigned mode, resolve_holes, min_coherence).

This method sizes a set of polygons. Before the sizing is applied, the polygons are merged. After that, sizing is applied on the individual result polygons of the merge step. The result may contain overlapping polygons, but no self-overlapping ones. Polygon overlap occurs if the polygons are close enough, so a positive sizing makes polygons overlap.

dx and dy describe the sizing. A positive value indicates oversize (outwards) while a negative one describes undersize (inwards). The sizing applied can be chosen differently in x and y direction. In this case, the sign must be identical for both dx and dy.

The "mode" parameter describes the corner fill strategy. Mode 0 connects all corner segments directly. Mode 1 is the "octagon" strategy in which square corners are interpolated with a partial octagon. Mode 2 is the standard mode in which corners are filled by expanding edges unless these edges form a sharp bend with an angle of more than 90 degree. In that case, the corners are cut off. In Mode 3, no cutoff occurs up to a bending angle of 135 degree. Mode 4 and 5 are even more aggressive and allow very sharp bends without cutoff. This strategy may produce long spikes on sharply bending corners. This method produces polygons and allows to fine-tune the parameters for that purpose.

Prior to version 0.21 this method was called "size_to_polygon". Is was renamed to avoid ambiguities for empty input arrays. The old version is still available but deprecated.

Input:	in[]	The input polygons.
	dx	The sizing value in x direction.
	dy	The sizing value in y direction.
	mode	The sizing mode (standard is 2).
	resolve_holes	True, if holes should be resolved into the hull.
	min_coherence	True, if touching corners should be resolved into less connected contours.
Return:	Polygon[]	The output polygons.

13.28 Class FileDialog (version 0.21) Various methods to request a file name.

Method Overview

get_existing_dir	Open a dialog to select a directory.
get_open_file_names	Select one or multiple files for opening.
get_open_file_name	Select one file for opening.
get_save_file_name	Select one file for writing.
assign	Assign the contents of another object to self.
dup	Creates a copy of self.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.28.1 assign(FileDialog other) Assign the contents of another object to self.

This method assigns the contents of another object to self. This is a deep copy that does not only copy the reference but the actual content.

13.28.2 destroy Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.28.3 [const] boolean destroyed Tell, if the object was destroyed.

Return: true — The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
false — The object is still alive.

13.28.4 [const] DText dup Creates a copy of self.

Return: FileDialog — The copied object of self.

13.28.5 [static] StringValue get_existing_dir(title, dir) Open a dialog to select a directory.

Input: title — The title of the dialog.
dir — The directory selected initially.
Return: StringValue — A StringValue object that contains the directory path selected or ??? with has_value?= false if "Cancel" was pressed.

13.28.6 [static] StringValue get_open_file_name(title, dir, filter)
Select one file for opening.

Input:	title	The title of the dialog.
	dir	The directory selected initially.
	filter	The filters available, for example Images (`*.png`, `*.xpm`, `*.jpg`); Text files (`*.txt`); XML files (`*.xml`).
Return:	StringValue	A StringValue object that contains the file selected or ??? with has_value?= false if "Cancel" was pressed.

13.28.7 [static] StringListValue get_open_file_names(title, dir, filter)
Select one or multiple files for opening.

Input:	title	The title of the dialog.
	dir	The directory selected initially.
	filter	The filters available, for example Images (`*.png`, `*.xpm`, `*.jpg`); Text files (`*.txt`); XML files (`*.xml`). .
Return:	StringListValue	A StringListValue object that contains the files selected or ??? with has_value?= false if "Cancel" was pressed.

13.28.8 [static] StringValue get_save_file_name(title, dir, filter)
Select one file for writing.

Input:	title	The title of the dialog.
	dir	The directory selected initially.
	filter	The filters available, for example Images (`*.png`, `*.xpm`, `*.jpg`); Text files (`*.txt`); XML files (`*.xml`).
Return:	StringValue	A StringValue object that contains the file selected or ??? with has_value?= false if "Cancel" was pressed.

13.29 Class ICplxTrans (version 0.21) A complex transformation.

A complex transformation provides magnification, mirroring at the x-axis, rotation by an arbitrary angle and a displacement. This version can transform integer-coordinate objects into floating-point coordinate objects, which is the generic and exact case.

Method Overview

from_dtrans	Conversion constructor from an floating-point transformation.
from_trans	Conversion constructor from an exact complex transformation.
new	Creates a unit transformation.
new	Conversion constructor from a fix-point transformation.
new	Constructor from a magnification.
new	Constructor from a simple transformation and a magnification.
new	Constructor from a simple transformation alone.
new	The standard constructor using magnification, angle, mirror flag and displacement.
inverted	Inversion.
invert	In-place inversion.
ctrans	The transformation of a distance.
trans	The transformation of a point.
*****	Multiplication (concatenation) of transformations.
<	A sorting criterion.
==	Equality test.
!=	Inequality test.
to_s	String conversion.
disp	Gets the displacement.
disp=	Sets the displacement.
rot	Returns the respective rotation code if possible.
is_mirror?	Gets the mirror flag.
mirror=	Sets the mirror flag.
is_unity?	Test, whether this is a unit transformation.
is_ortho?	Test, if the transformation is an orthogonal transformation.
s_trans	Extract the simple transformation part.
angle	Gets the angle.
angle=	Sets the angle.
mag	Gets the magnification.
is_mag?	Test, if the transformation is a magnifying one.
mag=	Sets the magnification.
m_*/r_*	Various angle/mirror codes for the named transformation.
r0	"unrotated" transformation.
r90	"rotated by 90 degree counterclockwise" transformation.
r180	"rotated by 180 degree counterclockwise" transformation.
r270	"rotated by 270 degree counterclockwise" transformation.
m0	"mirrored at the x-axis" transformation.
m45	"mirrored at the 45 degree axis" transformation.
m90	"mirrored at the y (90 degree) axis" transformation.
m135	"mirrored at the 135 degree axis" transformation.
assign	Assign the contents of another object to self.
dup	Creates a copy of self.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.29.1 [const] boolean !=(ICplxTrans)
Inequality test.

Input:	ICplxTrans text	The object to compare against.
Return:	true	This object and the given one are not equal.
	false	???.

13.29.2 [const] ICplxTrans *(ICplxTrans t)
Multiplication (concatenation) of transformations.

The * operator returns self*t ("t is applied before this transformation").

Input:	t	The transformation to apply before.
Return:	ICplxTrans	The modified transformation.

13.29.3 [const] boolean <(ICplxTrans)
A sorting criterion.

Input:	e	The object to compare against.
Return:	true	The object is 'less' than the other.
	false	??.

13.29.4 [const] boolean ==(ICplxTrans)
Equality test.

Input:	e	The object to compare against.
Return:	true	This object and the given one are equal.
	false	??.

13.29.5 [const] double angle
Gets the angle.

To check, if the transformation represents a rotation by a angle that is a multiple of 90 degree, use this predicate.

Return:	double	The rotation angle this transformation provides in degree units (0..360 deg).

13.29.6 angle=(double a)
Sets the angle.

Input:	a	The new angle.

13.29.7 assign(ICplxTrans other)
Assign the contents of another object to self.

This method assigns the contents of another object to self. This is a deep copy that does not only copy the reference but the actual content.

13.29.8 [const] integer ctrans(d)
The transformation of a distance.

The ctransethod transforms the given distance: $e = t(d)$. For the simple transformations, there is no magnification and no modification of the distance therefore.

Input: d The distance to transform.
Return: integer The transformed distance.

13.29.9 destroy
Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.29.10 [const]boolean destroyed
Tell, if the object was destroyed.

Return: true The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
false The object is still alive.

13.29.11 [const] const ref Point disp
Gets the displacement.

13.29.12 disp=(Point u)
Sets the displacement.

Input: u The new displacement.

13.29.13 [const] ICplxTrans dup
Creates a copy of self.

Return: ICplxTrans The copy of self.

13.29.14 [static] ICplxTrans from_dtrans(DCplxTrans dbl_trans)
Conversion constructor from an floating-point transformation.

13.29.15 [static] ICplxTrans from_trans(CplxTrans dbl_trans)
Conversion constructor from an exact complex transformation.

13.29.16 ICplxTrans invert
In-place inversion.

Inverts the transformation and replaces this transformation by the inverted one.

Return: ICplxTrans The inverted transformation.

13.29.17 [const] ICplxTrans inverted
Inversion.

Return: ICplxTrans The inverted transformation.

13.29.18 [const] boolean is_mag?
Test, if the transformation is a magnifying one.

This is the recommended test for checking if the transformation represents a magnification.

13.29.19 [const] boolean is_mirror?
Gets the mirror flag.

Return: true — The transformation is composed of a mirroring at the x-axis followed by a rotation by the angle given by the angle property.
false — ???.

13.29.20 [const] boolean is_ortho?
Test, if the transformation is an orthogonal transformation.

Return: true — The rotation is by a multiple of 90 degree.
false — The rotation is not orthogonal.

13.29.21 [const] boolean is_unity?
Test, whether this is a unit transformation.

Return: true — A unit transformation.
false — An other transformation.

13.29.22 [static] integer m_*/r_*
Various angle/mirror codes for the named transformation.

13.29.22.1 [static] integer m0 – "mirrored at the x-axis".

Return: integer — The angle/mirror code for this transformation.

13.29.22.2 [static] integer m135 – "mirrored at the 135 degree axis".

Return: integer — The angle/mirror code for this transformation.

13.29.22.3 [static] integer m45 – "mirrored at the 45 degree axis".

Return: integer — The angle/mirror code for this transformation.

13.29.22.4 [static] integer m90 – "mirrored at the 90 degree axis".

Return: integer — The angle/mirror code for this transformation.

13.29.22.5 [static] integer r0 – "unrotated".

Return: integer — The angle/mirror code for this transformation.

13.29.22.6 [static] integer r180 – "rotated by 180 degree counterclockwise".

Return: integer The angle/mirror code for this transformation.

13.29.22.7 [static] integer r270 – "rotated by 270 degree counterclockwise".

Return: integer The angle/mirror code for this transformation.

13.29.22.8 [static] integer r90 – "rotated by 90 degree counterclockwise".

Return: integer The angle/mirror code for this transformation.

13.29.23 [const] double mag
Gets the magnification.

Return: integer The angle/mirror code for this transformation.

13.29.24 mag=(double m)
Sets the magnification.

Input: m The new magnification.

13.29.25 mirror=(boolean)
Sets the mirror flag.

"mirroring" describes a reflection at the x-axis which is included in the transformation prior to rotation.

Input: boolean The new mirror flag.

13.29.26 [static] ICplxTrans new
Creates a unit transformation.

13.29.27 [static] ICplxTrans new(f)
Conversion constructor from a fix-point transformation.

A synonym of: [static] ICplxTrans new_f(f).

This constructor will create a transformation with a fixpoint transformation but no displacement.

Input: f The rotation/mirror code (r0 .. m135 constants).

13.29.28 [static] ICplxTrans new(double m)
Constructor from a magnification.

A synonym of: [static] ICplxTrans new_m(double m).

Creates a magnifying transformation without displacement and rotation given the magnification m.

Input: double m The magnification.

13.29.29 [static] ICplxTrans new(Trans t, double m)
Constructor from a simple transformation and a magnification.

A synonym of: [static] ICplxTrans new_tm(Trans t, double m).

Creates a magnifying transformation from a simple transformation and a magnification.

13.29.30 [static] ICplxTrans new(Trans t)
Constructor from a simple transformation alone.

A synonym of: [static] ICplxTrans new_t(Trans t).

Creates a magnifying transformation from a simple transformation and a magnification of 1.0.

13.29.31 [static] ICplxTrans new(double m, double r, boolean, DPoint u)
The standard constructor using magnification, angle, mirror flag and displacement.

A synonym of: [static] ICplxTrans new_mrmu(double m, double r, boolean, DPoint u).

The sequence of operations is: magnification, mirroring at x axis, rotation, application of displacement.

Input:	double m	The magnification.
	double r	The rotation angle in units of degree.
	boolean	True, if mirrored at x axis.
	u	The displacement.

13.29.32 [const] integer rot
Returns the respective rotation code if possible.

If this transformation is orthogonal (is_ortho? = true), then this method will return the corresponding fix point transformation, not taking into account magnification and displacement. Otherwise, the result reflects the quadrant the rotation goes into with the guarantee to reproduce the correct quadrant in the exact case.

13.29.33 [const] Trans s_trans
Extract the simple transformation part.

The simple transformation part does not reflect magnification not arbitrary angles. On the angle contribution up to a multiple of 90 degree is reflected.

13.29.34 [const] string to_s
String conversion.

Return:	string	The resulting string.

13.29.35 [const] Point trans(Point p)
The transformation of a point.

The trans method transforms the given point. q = t(p).

Input:	p	The point to transform.
Return:	Point	The transformed point.

13.30 Class Image (version 0.21)
An image to be stored as a layout annotation.

Images can be put onto the layout canvas as annotations, along with rulers and markers. Images can be monochrome (represent scalar data) as well as color (represent color images). The display of images can be adjusted in various ways, i.e. color mapping (translation of scalar values to colors), geometrical transformations (including rotation by arbitrary angles) and similar. Images are always based on floating point data. The actual data range is not fixed and can be adjusted to the data set (i.e. 0 ... 255 or -1 ... 1). This gives a great flexibility when displaying data which is the result of some measurement or calculation for example. The basic parameters of an image are the width and height of the data set, the width and height of one pixel, the geometrical transformation to be applied, the data range (from min_value to max_value) and the data mapping which is described by an own class, ImageDataMapping.

Method Overview

new	Create a new image with the default attributes.
new	Constructor from a image file.
new	Constructor from a image file.
new	Constructor for a monochrome image with the given pixel values.
new	Constructor for a monochrome image with the given pixel values.
new	Constructor for a color image with the given pixel values.
new	Constructor for a color image with the given pixel values.
box	Get the bounding box of the image.
transformed	Transform the ruler or marker with the given simple transformation.
transformed	Transform the image with the given simple transformation.
transformed_cplx	Transform the image with the given complex transformation.
width	Get the width of the image in pixels.
height	Get the height of the image in pixels.
filename	Get the name of the file loaded of an empty string if not file is loaded.
is_empty?	Returns true, if the image does not contain any data (i.e. is default constructed).
is_color?	Returns true, if the image is a color image.
set_pixel	Set one pixel (monochrome).
set_pixel	Set one pixel (color).
get_pixel	Accessor to one pixel (monochrome and color).
get_pixel	Accessor to one pixel (monochrome and color).
set_data	Write the image data field (monochrome).
set_data	Write the image data field (color).
pixel_width=	Set the pixel width.
pixel_width	Get the pixel width.
pixel_height=	Set the pixel height.
pixel_height	Get the pixel height.
trans=	Set the transformation.
trans	Return the pixel-to-micron transformation.
min_value=	Set the minimum value.
min_value	Get the lower limit of the values in the data set.
max_value=	Set the maximum value.
max_value	Get the upper limit of the values in the data set.
visible=	Set the visibility.
is_visible?	Get a flag indicating whether the image object is visible.
id	Get the Id.
data_mapping=	Set the data mapping object.
data_mapping	Get the data mapping.
to_s	Convert the image to a string.

assign	Assign the contents of another object to self.
dup	Creates a copy of self.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.30.1 assign(Image other)
Assign the contents of another object to self.

This method assigns the contents of another object to self. This is a deep copy that does not only copy the reference but the actual content.

13.30.2 [const] DBox box
Get the bounding box of the image.

Return: DBox The bounding box.

13.30.3 [const] const ref ImageDataMapping data_mapping
.Get the data mapping

The data mapping describes the transformation of a pixel value (any double value) into pixel data which can be sent to the graphics cards for display. See ImageDataMapping for a more detailed description.

Return: ImageDataMapping The data mapping object.

13.30.4 data_mapping=(ImageDataMapping data_mapping)
Set the data mapping object.

The data mapping describes the transformation of a pixel value (any double value) into pixel data which can be sent to the graphics cards for display. See ImageDataMapping for a more detailed description.

13.30.5 destroy
Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.30.6 [const] boolean destroyed
Tell, if the object was destroyed.

Return: true The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
false The object is still alive.

13.30.7 [const] Image dup
Creates a copy of self.

Return: Image The copy of self.

13.30.8 [const] string filename
Get the name of the file loaded or an empty string if no file is loaded.

Return: string — The loaded path and file name or empty if no file is loaded.

13.30.9 [const] double get_pixel(unsigned x, unsigned y)
Accessor to one pixel (monochrome and color).

If the component index, x or y value exceeds the image bounds, this method returns 0.0.

Input:
- unsigned x — The x coordinate of the pixel (in mathematical order: 0 is the lowest, 0 ... width()-1 is the range).
- unsigned y — The y coordinate of the pixel (in mathematical order: 0 is the lowest, 0 ... height()-1 is the range).

13.30.10 [const] double get_pixel(unsigned x, unsigned y, unsigned component)
Accessor to one pixel (monochrome and color).

If the component index, x or y value exceeds the image bounds, this method returns 0.0. For monochrome images, the component index is ignored.

Input:
- unsigned x — The x coordinate of the pixel (in mathematical order: 0 is the lowest, 0 ... width()-1 is the range).
- unsigned y — The y coordinate of the pixel (in mathematical order: 0 is the lowest, 0 ... height()-1 is the range).
- unsigned component — 0 for red, 1 for green, 2 for blue.

13.30.11 [const] unsigned height
Get the height of the image in pixels.

Return: unsigned — The height of the image in pixels.

13.30.12 [const] integer id
Get the Id.

The Id is an arbitrary integer that can be used to track the evolution of an image object. The Id is not changed when the object is edited. On initialization, a unique Id is given to the object. The Id cannot be changed.

This behavior has been modified in version 0.20.

Return: integer — The image Id.

13.30.13 [const] boolean is_color?
Returns true, if the image is a color image.

Return:
- true — The image is a color image.
- false — The image is a monochrome image.

13.30.14 [const] boolean is_empty?

Returns true, if the image does not contain any data (i.e. is default constructed).

Return: true — The image is empty.
false — The image contains data.

13.30.15 [const] boolean is_visible?

Gets a flag indicating whether the image object is visible.

An image object can be made invisible by setting the visible property to false.

This method has been introduced in version 0.20.

Return: true — The image is visible.
false — The image is invisible.

13.30.16 [const] double max_value

Get the upper limit of the values in the data set.

This value determines the upper end of the data mapping (i.e. white value etc.). It does not necessarily correspond to the maximum value of the data set but it must be larger than that.

Return: double — The maximum value.

13.30.17 max_value=(double)

Set the maximum value.

See the max_value method for the description of the maximum value property.

Input: double — The maximum value.

13.30.18 [const] double min_value

Get the lower limit of the values in the data set.

This value determines the upper end of the data mapping (i.e. black value etc.). It does not necessarily correspond to the minimum value of the data set but it must be larger than that.

Return: double — The minimum value.

13.30.19 min_value=(double)

Set the minimum value.

See min_value for the description of the minimum value property.

Input: double — The minimum value.

13.30.20 [static] Image new

Create a new image with the default attributes.

This will create an empty image without data and no particular pixel width or related. Use the **??** or set_data methods to set image properties and pixel values. **Comment: Method Image.read_file not described.**

13.30.21 [static] Image new(filename, DCplxTrans t)
Constructor from a image file.

This constructor creates an image object from a file (which can have any format supported by Qt) and a transformation. The image will originally be put to position 0, 0 (lower left corner) and each pixel will have a size of 1. The transformation describes how to transform this image into micron space.

Input:	filename	The file name and path to the image file to load.
	t	The transformation to apply to the image when displaying it.
Return:	Image	The image object.

13.30.22 [static] Image new(filename)
Constructor from a image file.

This constructor creates an image object from a file (which can have any format supported by Qt) and a unit transformation. The image will originally be put to position 0, 0 (lower left corner) and each pixel will have a size of 1 (micron).

Input:	filename	The file name and path to the image file to load.
Return:	Image	The image object.

13.30.23 [static] Image new(unsigned w, unsigned h, double data[])
Constructor for a monochrome image with the given pixel values.

This constructor creates an image from the given pixel values. The values have to be organized line by line. Each line must consist of "w" values where the first value is the leftmost pixel. Note, that the rows are oriented in the mathematical sense (first one is the lowest) contrary to the common convention for image data. Initially the pixel width and height will be 1 micron and the data range will be 0 to 1.0 (black to white level). To adjust the data range use the min_value and max_value properties.

Input:	unsigned w	The width of the image.
	unsigned h	The height of the image.
	double data[]	The data set which will become owned by the image.
Return:	Image	The image object.

13.30.24 [static] Image new(unsigned w, unsigned h, DCplxTrans t, double data[])
Constructor for a monochrome image with the given pixel values.

This constructor creates an image from the given pixel values. The values have to be organized line by line. Each line must consist of "w" values where the first value is the leftmost pixel. Note, that the rows are oriented in the mathematical sense (first one is the lowest) contrary to the common convention for image data. Initially the pixel width and heigt will be 1 micron and the data range will be 0 to 1.0 (black to white level). To adjust the data range use the min_value and max_value properties.

Input:	unsigned w	The width of the image.
	unsigned h	The height of the image.
	t	The transformation from pixel space to micron space.
	double data[]	The data set which will become owned by the image.
Return:	Image	The image object.

13.30.25 [static] Image new(unsigned w, unsigned h, double red[], double green[], double blue[])
Constructor for a color image with the given pixel values.

This constructor creates an image from the given pixel values. The values have to be organized line by line and separated by color channel. Each line must consist of "w" values where the first value is the leftmost pixel. Note, that the rows are oriented in the mathematical sense (first one is the lowest) contrary to the common convention for image data. Initially the pixel width and height will be 1 micron and the data range will be 0 to 1.0 (black to white level). To adjust the data range use the min_value and max_value properties.

Input:	unsigned w	The width of the image.
	unsigned h	The height of the image.
	double red[]	The red channel data set which will become owned by the image.
	double green[]	The green channel data set which will become owned by the image.
	double blue[]	The blue channel data set which will become owned by the image.
Return:	Image	The image object.

13.30.26 [static] Image new(unsigned w, unsigned h, DCplxTrans t, double red[], double green[], double blue[])
Constructor for a color image with the given pixel values.

This constructor creates an image from the given pixel values. The values have to be organized line by line and separated by color channel. Each line must consist of "w" values where the first value is the leftmost pixel. Note, that the rows are oriented in the mathematical sense (first one is the lowest) contrary to the common convention for image data. Initially the pixel width and height will be 1 micron and the data range will be 0 to 1.0 (black to white level). To adjust the data range use the min_value and max_value properties.

Input:	unsigned w	The width of the image.
	unsigned h	The height of the image.
	t	The transformation from pixel space to micron space.
	double red[]	The red channel data set which will become owned by the image.
	double green[]	The green channel data set which will become owned by the image.
	double blue[]	The blue channel data set which will become owned by the image.
Return:	Image	The image object.

13.30.27 [const] double pixel_height
Get the pixel height.

See pixel_height= for a description of that property.

Return:	double	The pixel height.

13.30.28 pixel_height=(double)
Set the pixel height.

The pixel height determines the height of on pixel in the original space which is transformed to micron space with the transformation.

Input:	double	The pixel height.

13.30.29 [const] double pixel_width
Get the pixel height.

See pixel_width= for a description of that property.

Return: double The pixel width.

13.30.30 pixel_width=(double)
Set the pixel height.

The pixel width determines the width of on pixel in the original space which is transformed to micron space with the transformation.

Input: double The pixel width.

13.30.31 set_data(unsigned w, unsigned h, double d[])
Write the image data field (monochrome).

See the constructor description for the data organisation in that field.

Input: unsigned w The width of the new data.
unsigned h The height of the new data.
double d[] The monochrome data to load into the image.

13.30.32 set_data(unsigned w, unsigned h, double red[], double green[], double blue[])
Write the image data field (color).

See the constructor description for the data organization in that field.

Input: unsigned w The width of the new data.
unsigned h The height of the new data.
double red[] The red channel data to load into the image.
double green[] The green channel data to load into the image.
double blue[] The blue channel data to load into the image.

13.30.33 set_pixel(unsigned x, unsigned y, double v)
Set one pixel (monochrome).

If the component index, x or y value exceeds the image bounds, or the image is a color image, this method does nothing.

Input: unsigned x The x coordinate of the pixel (in mathematical order: 0 is the lowest, 0 ... width()-1 is the range).
unsigned y The y coordinate of the pixel (in mathematical order: 0 is the lowest, 0 ... height()-1 is the range).
double v The value.

13.30.34 set_pixel(unsigned x, unsigned y, double red, double green, double blue)
Set one pixel (color).

If the component index, x or y value exceeds the image bounds, or the image is a color image, this method does nothing.

Input: unsigned x — The x coordinate of the pixel (in mathematical order: 0 is the lowest, 0 ... width()-1 is the range).
unsigned y — The y coordinate of the pixel (in mathematical order: 0 is the lowest, 0 ... height()-1 is the range).
double red — The red component.
double green — The green component.
double blue — The blue component.

13.30.35 string to_s

Convert to a string.

Return: string — The string.

13.30.36 [const] const refDCplxTrans trans

Return the pixel-to-micron transformation.

This transformation converts pixel coordinates (0,0 being the lower left corner and each pixel having the dimension of pixel_width and pixel_height) to micron coordinates. The coordinate of the pixel is the lower left corner of the pixel.

13.30.37 trans=(DCplxTrans t)

Set the transformation.

This transformation converts pixel coordinates (0,0 being the lower left corner and each pixel having the dimension of pixel_width and pixel_height) to micron coordinates. The coordinate of the pixel is the lower left corner of the pixel.

Input: t — The transformation to apply.

13.30.38 [const] Image transformed(DTrans t)

Transform the ruler or marker with the given simple transformation.

Comment: Same as image transformation ?

Input: t — The transformation to apply.
Return: Image — The transformed image object.

13.30.39 [const] Image transformed(DTrans t)

Transform the image with the given simple transformation.

Input: t — The transformation to apply.
Return: Image — The transformed image object.

13.30.40 [const] Image transformed_cplx(DCplxTrans t)

Transform the image with the given complex transformation.

Input: t — The transformation to apply.
Return: Image — The transformed image object.

13.30.41 visible=(boolean)
Set the visibility.

See the is_visible? method for a description of this property.

This method has been introduced in version 0.20.

Input:	true	Set to visible.
	false	set to invisible.

13.30.42 [const] unsigned width
Get the width of the image in pixels.

Return:	unsigned	The width of the image in pixels.

13.31 Class ImageDataMapping (version 0.21) A structure describing the data mapping of an image object.

Data mapping is the process of transforming the data into RGB pixel values. This implementation provides four adjustment steps:

1. In the case of monochrome data, the data is converted to a RGB triplet using the color map. The default color map will copy the value to all channels rendering a gray scale.
2. The data is normalized to 0 … 1, corresponding to the min_value and max_value, and a color channel-independent brightness and contrast adjustment is applied.
3. A per-channel multiplier (red_gain, green_gain, blue_gain) is applied.
4. The gamma function is applied, the result converted into a 0 … 255 pixel value range and clipped.

Method Overview

new	Create a new data mapping object with default settings
clear_colormap	The the color map of this data mapping object.
add_colormap_entry	Add a color map entry for this data mapping object.
num_colormap_entries	Returns the current number of color map entries.
colormap_color	Returns the color for a given color map entry.
colormap_value	Returns the value for a given color map entry.
brightness=	Set the brightness value.
brightness	Get the brightness value.
contrast=	Set the contrast value.
contrast	Get the contrast value.
gamma=	Set the gamma value.
gamma	Get the gamma value.
red_gain=	Set the red channel gain.
red_gain	Get the red channel gain.
green_gain=	Set the green channel gain.
green_gain	Get the green channel gain.
blue_gain=	Set the blue channel gain.
blue_gain	Get the blue channel gain.
assign	Assign the contents of another object to self.
dup	Creates a copy of self.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.31.1 add_colormap_entry(double value, unsigned color) Add a colormap entry for this data mapping object.

This settings establishes a color mapping for a given value in the monochrome channel. The color must be given as a 32 bit integer, where the lowest order byte describes the blue component (0 to 255), the second byte the green component and the third byte the red component, i.e. 0xff0000 is red and 0x0000ff is blue.

Input:

double value	The value at which the given color should be applied.
unsigned color	The color to apply (a 32 bit RGB value).

13.31.2 assign(ImageDataMapping other)
Assign the contents of another object to self.

This method assigns the contents of another object to self. This is a deep copy that does not only copy the reference but the actual content.

13.31.3 [const] double blue_gain
Get the blue channel gain.

This value is the multiplier by which the blue channel is scaled after applying false color transformation and contrast/brightness/gamma.

1.0 is a neutral value. The gain should be >=0.0.

Return: double The blue channel gain.

13.31.4 blue_gain=(double)
Set the blue channel gain.

See blue_gain for a description of this property.

Input: double The blue channel gain.

13.31.5 [const] double brightness
Get the brightness value.

The brightness is a double value between roughly -1.0 and 1.0. Neutral (original) brightness is 0.0.

Return: double The brightness value.

13.31.6 brightness=(double)
Set the brightness value.

See brightness for a description of this property.

Input: double The brightness value.

13.31.7 clear_colormap
Clear the color map of this data mapping object.

13.31.8 [const] unsigned colormap_color(unsigned n)
Returns the color for a given color map entry.

Input: unsigned n The index of the entry (0 ... num_colormap_entries-1).
Return: unsigned The color (see add_colormap_entry for a description).

13.31.9 [const] double colormap_value(unsigned n)
Returns the vlue for a given color map entry.

Input: unsigned n The index of the entry (0 ... num_colormap_entries-1).
Return: unsigned The color (see add_colormap_entry for a description).

13.31.10 [const] double contrast
Get the contrast value.

The contrast is a double value between roughly -1.0 and 1.0. Neutral (original) contrast is 0.0.

Return: double The contrast value.

13.31.11 contrast=(double)
Set the contrast value.

See contrast for a description of this property.

Input: double The contrast value.

13.31.12 destroy
Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.31.13 [const] boolean destroyed
Tell, if the object was destroyed.

Return: true The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
false The object is still alive.

13.31.14 [const] ImageDataMapping dup
Creates a copy of self.

Return: ImageDataMapping The copy of self.

13.31.15 [const] double gamma
Get the gamma value.

The gamma value allows to adjust for non-linearity in the display chain and to enhance contrast. A value for linear intensity reproduction on the screen is roughly 0.5. The exact value depends on the monitor calibration. Values below 1.0 give a "softer" appearance while values above 1.0 give a "harder" appearance.

Return: double The gamma value.

13.31.16 gamma=(double)
Set the gamma value.

See gamma for a description of this property.

Input: double The gamma value.

13.31.17 [const] double green_gain
Get the green channel gain.

See blue_gain for a description of this property.

Return: double The green channel gain.

13.31.18 green_gain=(double)
Set the green channel gain.

See blue_gain for a description of this property.

Input: double The green channel gain.

13.31.19 [static] ImageDataMapping new
Create a new data mapping object with default settings.

13.31.20 [const] unsigned num_colormap_entries
Returns the current number of color map entries.

Return: unsigned The number of color map entries.

13.31.21 [const] double red_gain
Get the red channel gain.

See blue_gain for a description of this property.

Return: double The red channel gain.

13.31.22 red_gain=(double)
Set the red channel gain.

See blue_gain for a description of this property.

Input: double The red channel gain.

13.32 Class InputDialog (version 0.21)
Various methods to open a dialog requesting data entry.

Method Overview

get_string	Open an input dialog requesting a string.
get_item	Open an input dialog requesting an item from a list.
get_string_password	Open an input dialog requesting a string without showing the actual characters entered.
get_double	Open an input dialog requesting a floating-point value.
get_double_ex	Open an input dialog requesting a floating-point value with. enhanced capabilities
get_int	Open an input dialog requesting an integer value.
get_int_ex	Open an input dialog requesting an integer value with enhanced capabilities.
assign	Assign the contents of another object to self.
dup	Creates a copy of self.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.32.1 assign(InputDialog other)
Assign the contents of another object to self.

This method assigns the contents of another object to self. This is a deep copy that does not only copy the reference but the actual content.

13.32.2 destroy
Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.32.3 [const] boolean destroyed
Tell, if the object was destroyed.

Return:	true	The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
	false	The object is still alive.

13.32.4 [const] InputDialog dup
Creates a copy of self.

Return:	InputDialog	The copy of self.

13.32.5 [static] DoubleValue get_double(title, label, double value, digits)

Open an input dialog requesting a floating-point value.

Input:	title	The title to display for the dialog.
	label	The label text to display for the dialog.
	double value	The initial value for the input field.
	digits	The number of digits allowed.
Return:	DoubleValue	A DoubleValue object with has_value? set to true, if "Ok" was pressed and the value given in it's value attribute.

13.32.6 [static] DoubleValue get_double_ex(title, label, double value, double min, double max, digits)

Open an input dialog requesting a floating-point value.

Input:	title	The title to display for the dialog.
	label	The label text to display for the dialog.
	double value	The initial value for the input field.
	double min	The minimum value allowed.
	double max	The maximum value allowed.
	digits	The number of digits allowed.
Return:	IntValue	A IntValue object with has_value? set to true, if "Ok" was pressed and the value given in it's value attribute.

13.32.7 [static] IntValue get_int(title, label, integer)

Open an input dialog requesting a integer value.

Input:	title	The title to display for the dialog.
	label	The label text to display for the dialog.
	integer	The initial value for the input field.
Return:	IntValue	A IntValue object with has_value? set to true, if "Ok" was pressed and the value given in it's value attribute.

13.32.8 [static] IntValue get_int_ex(title, label, value, min, max, step)

Open an input dialog requesting an integer value with enhanced capabilities.

Input:	title	The title to display for the dialog.
	label	The label text to display for the dialog.
	value	The initial value for the input field.
	min	The minimum value allowed.
	max	The maximum value allowed.
	step	The step size for the spin buttons.
Return:	IntValue	A IntValue object with has_value? set to true, if "Ok" was pressed and the value given in it's value attribute.

13.32.9 [static] StringValue get_item(title, label, items[], selection)

Open an input dialog requesting an item from a list.

Input:	title	The title to display for the dialog.
	label	The label text to display for the dialog.
	items[]	The list of items to show in the selection element.
	selection	The initial selection (index of the element selected intially).
Return:	StringValue	A StringValue object with has_value? set to true, if "Ok" was pressed and the value given in it's value attribute.

13.32.10 [static] StringValue get_string(title, label, value)

Open an input dialog requesting a string.

Input:	title	The title to display for the dialog.
	label	The label text to display for the dialog.
	value	The initial value for the input field.
Return:	StringValue	A StringValue object with has_value? set to true, if "Ok" was pressed and the value given in it's value attribute.

13.32.11 [static] StringValue get_string_password(title, label, value)

Open an input dialog requesting a string without showing the actual characters entered.

Input:	title	The title to display for the dialog.
	label	The label text to display for the dialog.
	value	The initial value for the input field.
Return:	StringValue	A StringValue object with has_value? set to true, if "Ok" was pressed and the value given in it's value attribute.

13.33 Class InstElement (version 0.21) An element in an instantiation path.

This objects are used to reference a single instance in a instantiation path. The object is composed of a CellInstArray object (accessible through the cell_inst accessor) that describes the basic instance, which may be an array. The particular instance within the array can be further retrieved using the array_member_trans, specific_trans or specific_cplx_trans methods.

Method Overview

new	Default constructor.
new	Create an instance element from a single instance alone.
new	Create an instance element from an array instance pointing into a certain array member.
cell_inst	Accessor to the cell instance (array).
prop-id	Accessor to the property attached to this instance.
<	Provide an order criterion for two InstElement objects.
!=	Inequality test of two InstElement objects.
==	Equality test of two InstElement objects.
specific_trans	Returns the specific transformation for this instance.
specific_cplx_trans	Returns the specific complex transformation for this instance.
array_member_trans	Returns the transformation for this array member.
assign	Assign the contents of another object to self.
dup	Creates a copy of self.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.33.1 [const] boolean !=(InstElement b) Inequality test of two InstElement objects.

Warning:: **This operator returns true if both instance elements refer to the same instance, not just identical ones.**

Input:	InstElement b	The text object to compare against.
Return:	true	The objects are unequal.
	false	The objects are equal.

13.33.2 [const] boolean <(InstElement b) Less operator that provides an order criterion for two InstElement objects.

This operator is provided to establish any order, not necessarily a particular one.

Input:	InstElement b	The object to compare against.
Return:	true	This object is "less" than the given one.
	false	This object is "greater" or equal than the given one.

13.33.3 [const] boolean ==(InstElement b) Equality test.

Warning:: **This operator returns true if both instance elements refer to the same instance, not just identical ones.**

Input:	InstElement b	The object to compare against.
Return:	true	The objects are equal or refers to the same instance.
	false	The objects are unequal or refers not to the same instance.

13.33.4 [const] Trans array_member_trans
Returns the transformation for this array member.

The array member transformation is the one applicable in addition to the global transformation for the member selected from an array. If this instance is not an array instance, the specific transformation is a unit transformation without displacement.

13.33.5 assign(InstElement other)
Assign the contents of another object to self.

This method assigns the contents of another object to self. This is a deep copy that does not only copy the reference but the actual content.

13.33.6 [const] const ref CellInstArray cell_inst
Accessor to the cell instance (array).

13.33.7 destroy
Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.33.8 [const] boolean destroyed
Tell, if the object was destroyed.

Return:	true	The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
	false	The object is still alive.

13.33.9 [const] InstElement dup
Creates a copy of self.

Return:	InstElement	The copy of self.

13.33.10 [static] InstElement new
Default constructor.

13.33.11 [static] InstElement new(Instance inst)
Create an instance element from a single instance alone.

A synonym for: [static] InstElement new_i(Instance inst).

Starting with version 0.15, this method takes an Instance object (an instance reference) as the argument.

13.33.12 [static] InstElement new(Instance inst, a_index, b_index)
Create an instance element from an array instance pointing into a certain array member.

A synonym for: [static] InstElement new_iab(Instance inst, a_index, b_index).

Starting with version 0.15, this method takes an Instance object (an instance reference) as the argument.

Input: Instance inst — The instance reference.
a_index — The index a as unsigned long integer.
b_index — The index b as unsigned long integer.

Return:

13.33.13 [const] unsigned prop-id
Accessor to the property attached to this instance.

Return: unsigned — The property Id.

13.33.14 [const] CplxTrans specific_cplx_trans
Returns the specific complex transformation for this instance.

The specific transformation is the one applicable for the member selected from an array. This is the effective transformation applied for this array member. array_member_trans gives the transformation applied additionally to the instances' global transformation (in other words, specific_cplx_trans = array_member_trans * cell_inst.cplx_trans).

13.33.15 [const] Trans specific_trans
Returns the specific transformation for this instance.

The specific transformation is the one applicable for the member selected from an array. This is the effective transformation applied for this array member. array_member_trans gives the transformation applied additionally to the instances' global transformation (in other words, specific_cplx_trans = array_member_trans * cell_inst.trans). This method delivers a simple transformation that does not include magnification components. To get these as well, use specific_cplx_trans.

13.34 Class Instance (version 0.21)
An instance proxy.

An instance proxy is basically a pointer to an instance of different kinds, similar to Shape, the shape proxy. Instance objects can be duplicated without creating copies of the instances itself: the copy will still point to the same instance than the original.

Method Overview

prop_id	Get the properties Id associated with the instance.
has_prop_id?	Check, if the instance is associated with a properties Id.
is_null?	Check, if the instance is a valid one.
parent_cell_index	Retrieve the reference to the parent cell.
cell_index	Get the index of the cell this instance refers to.
is_regular_array?	Test, if this instance is a regular array.
a	Return the displacement vector for the "a" axis.
b	Return the displacement vector for the "b" axis.
na	Return the number of instances in the "a" axis.
nb	Return the number of instances in the "b" axis.
cplx_trans	Get the complex transformation of the instance or the first instance in the array.
trans	Get the transformation of the first instance in the array.
size	The number of single instances in the instance array.
is_complex?	Test, if the array is a complex array.
cell_inst	Get the basic CellInstArray object associated with this instance reference.
<	Less operator that provides an order criterion for two Instance objects.
!=	Equality test.
==	Inequality test.
to_s	Create a string showing the contents of the reference.
assign	Assign the contents of another object to self.
dup	Creates a copy of self.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.34.1 [const] boolean !=(Instance b)
Inequality of two Instance objects.

Warning: This operator returns true if both objects refer to the same instance, not just identical ones.

Input:	Instance b	The text object to compare against.
Return:	true	The objects are not equal.
	false	The objects are equal.

13.34.2 [const] boolean <(Instance b)
Provide an order criterion for two Instance objects.

Warning: This operator is just provided to establish any order, not a particular one.

Input:	Instance b	The object to compare against.
Return:	true	This object is "less" than the given one.
	false	This object is "greater" or equal than the given one.

13.34.3 [const] boolean ==(Instance b)
Equality of two Instance objects.

Warning: **This operator returns true if both objects refer to the same instance, not just identical ones.**

Input:	Instance b	The object to compare against.
Return:	true	The objects are equal or refers to the same instance.
	false	The objects are unequal or refers not to the same instance.

13.34.4 [const] Point a
Return the displacement vector for the "a" axis.

Return:	Point	The displacement vector for the "a" axis.

13.34.5 assign(Instance other)
Assign the contents of another object to self.

This method assigns the contents of another object to self. This is a deep copy that does not only copy the reference but the actual content.

13.34.6 [const] Point b
Return the displacement vector for the "b" axis.

Return:	Point	The displacement vector for the "b" axis.

13.34.7 [const] unsigned cell_index
Get the index of the cell this instance refers to.

Return:	unsigned	The index of the cell this instance refers to.

13.34.8 [const] const ref CellInstArray cell_inst
Get the basic CellInstArray object associated with this instance reference.

13.34.9 [const] CplxTrans cplx_trans
Get the complex transformation of the instance or the first instance in the array.

This method is always valid compared to trans, since simple transformations can be expressed as complex transformations as well.

13.34.10 destroy
Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.34.11 [const] boolean destroyed
Tell, if the object was destroyed.

Return: true — The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
false — The object is still alive.

13.34.12 [const] Instance dup
Creates a copy of self.

Return: Instance — The copy of self.

13.34.13 [const] boolean has_prop_id?
Check, if the instance is associated with a properties Id.

Return: true — The instance is associated with a properties Id.
false — The instance has no properties Id.

13.34.14 [const] boolean is_complex?
Test, if the array is a complex array.

Return: true — The array represents complex instances (that is, with magnification and arbitrary rotation angles).
false — The array represents simple instances (that is, without magnification and arbitrary rotation angles).

13.34.15 [const] boolean is_null?
Check, if the instance is a valid one.

Return: true — The instance is a valid one.
false — The instance is an invalid one.

13.34.16 [const] boolean is_regular_array?
Test, if this instance is a regular array.

Return: true — This instance is a regular array.
false — This instance is not a regular array.

13.34.17 [const] unsigned long na
Return the number of instances in the "a" axis.

Return: unsigned long — The number of instances in the "a" axis.

13.34.18 [const] unsigned long nb
Return the number of instances in the "b" axis.

Return: unsigned long — The number of instances in the "b" axis.

13.34.19 [const] unsigned parent_cell_index
Retrieve the reference to the parent cell.

Return: unsigned The reference to the parent cell.

13.34.20 [const] unsigned prop_id
Get the properties Id associated with the instance.

Return: unsigned The associated properties Id.

13.34.21 [const] unsigned size
The number of single instances in the instance array.

If the instance represents a single instance, the count is 1. Otherwise it is na*nb.

Return: unsigned The number of single instances in the instance array.

13.34.22 [const] string to_s
Create a string showing the contents of the reference.

This method has been introduced with version 0.16.

Return: string The contents of the reference as string.

13.34.23 [const] const ref Trans trans
Get the transformation of the first instance in the array.

The transformation returned is only valid if the array does not represent a complex transformation array.

13.35 Class IntValue (version 0.21) Encapsulate an integer value.

This class is provided as a return value of InputDialog::get_int. By using an object rather than a pure value, an object with has_value?= false can be returned indicating that the "Cancel" button was pressed.

Method Overview

has_value?	True, if a value is present.
to_i	Get the actual value (a synonym for value).
value	Get the actual value.
assign	Assign the contents of another object to self.
dup	Creates a copy of self.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.35.1 assign(IntValue other) Assign the contents of another object to self.

This method assigns the contents of another object to self. This is a deep copy that does not only copy the reference but the actual content.

13.35.2 destroy Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.35.3 [const] boolean destroyed Tell, if the object was destroyed.

Return:	true	The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
	false	The object is still alive.

13.35.4 [const] IntValue dup Creates a copy of self.

Return:	IntValue	The copy of self.

13.35.5 [const] boolean has_value? Query weather a value is present.

Return:	true	A value is present.
	false	Indication that the "Cancel" button was pressed.

13.35.6 [const] double to_i Get the actual value (a synonym for value).

Return:	integer	The actual value.

13.35.7 [const] double value
Get the actual value.

Return: integer The actual value.

13.36 Class LayerInfo (version 0.21) A structure encapsulating the layer properties.

The layer properties describe how a layer is stored in a GDSII or OASIS file for example.

Method Overview

new	The default constructor.
new	The constructor for a layer/data type pair.
new	The constructor for a named layer.
new	The constructor for a named layer with layer and data type.
to_s	Convert the layer info object to a string
==	Equality test of two layer info objects.
!=	Inequality test of two layer info objects.
is_equivalent?	Equivalence of two layer info objects.
is_named?	Returns true, if the layer is purely specified by name.
name=	Set the layer name.
name	Gets the layer name.
layer=	Sets the layer number.
layer	Gets the layer number.
datatype=	Set the data type.
datatype	Gets the data type.
assign	Assign the contents of another object to self.
dup	Creates a copy of self.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.36.1 [const] boolean !=(LayerInfo b) Inequality of two layer info objects.

This method was added in version 0.18.

Input:	p	The object to compare against.
Return:	true	Inequality, both are not equal.
	false	???.

13.36.2 [const] boolean == Equality of two layer info objects.

This method was added in version 0.18.

Input:	p	The object to compare against.
Return:	true	Equality, both are equal.
	false	???.

13.36.3 assign(LayerInfo other) Assign the contents of another object to self.

This method assigns the contents of another object to self. This is a deep copy that does not only copy the reference but the actual content.

13.36.4 [const] integer datatype
Gets the data type.

Return: integer The data type.

13.36.5 datatype=(integer)
Sets the data type.

Input: integer The data type.

13.36.6 destroy
Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.36.7 [const] boolean destroyed
Tell, if the object was destroyed.

Return: true The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
false The object is still alive.

13.36.8 [const] LayerInfo dup
Creates a copy of self.

Return: LayerInfo The copy of self.

13.36.9 [const] boolean is_equivalent?(LayerInfo b)
Equivalence of two layer info objects.

First, layer and data type are compared. The name is of second order and used only if no layer or data type is given. This is basically a weak comparison that reflects the search preferences.

This method was added in version 0.18.

Return: true Layer and data type are equivalent, or names are equivalent. Later used as fall back if no layer and data type is given.
false Layer and data type, if given, or names are different.

13.36.10 [const] boolean is_named?
Returns true, if the layer is purely specified by name.

This method was added in version 0.18.

Return: true The layer is purely specified by name.
false Layer or data type is given.

13.36.11 [const] integer layer
Gets the layer number.

Return: integer The layer number.

13.36.12 [const] integer layer=
Sets the layer number.

Input: integer The layer number.

13.36.13 [const] string name
Gets the layer name.

Return: string The layer name.

13.36.14 name=(string)
Sets the layer name.

The name is set on OASIS input for example, if the layer has a name.

Input: string The layer name.

13.36.15 [static] LayerInfo new
The default constructor.

Creates a default LayerInfo object.

This method was added in version 0.18.

Return: LayerInfo The new object.

13.36.16 [static] LayerInfo new(layer, datatype)
The constructor for a layer/data type pair.

This method was added in version 0.18.

Input: layer The layer number.
datatype The data type number.
Return: LayerInfo The new object representing a layer and data type.

13.36.17 [static] LayerInfo new(name)
The constructor for a named layer.

This method was added in version 0.18.

Input: name The name.
Return: LayerInfo The new object representing a named layer.

13.36.18 [static] LayerInfo new(layer, datatype, name)
The constructor for a named layer with layer and data type.

This method was added in version 0.18.

Input: layer The layer number.
datatype The data type number.
name The name.
Return: LayerInfo The new object representing a named layer with layer and data type.

13.36.19 [const] string to_s
Convert the layer info object to a string.

This method was added in version 0.18.

Return: string A string representing the layer info.

13.37 Class LayerMap (version 0.21)
An object representing an arbitrary mapping of physical layers to logical layers.

"Physical" layers are stream layers or other separated layers in a CAD file. "Logical" layers are the layers present in a Layout object. Logical layers are represented by an integer index while physical layers are given by a layer and data type number or name. A logical layer is created automatically in the layout on reading if it does not exist yet.

The mapping describes an association of a set of physical layers to a set of logical ones, where multiple physical layers can be mapped to a single logical one, which effectively merges the layers.

This class has been introduced in version 0.18.

Method Overview

is_mapped?	Check, if a given physical layer is mapped.
logical	Returns the logical layer (the layer index in the layout object) for a given physical layer.
mapping_str	Returns the mapping string for a given logical layer.
mapping	Returns the mapped physical (or target if one is specified) layer for a given logical layer.
map	Maps a physical layer to a logical one.
map	Maps a physical layer to a logical one with a target layer.
map	Maps a physical layer interval to a logical one.
map	Maps a physical layer interval to a logical one with a target layer.
map	Maps a physical layer given by a string to a logical one.
clear	Clears the map.
assign	Assign the contents of another object to self.
dup	Creates a copy of self.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.37.1 assign(LayerMap other)
Assign the contents of another object to self.

This method assigns the contents of another object to self. This is a deep copy that does not only copy the reference but the actual content.

13.37.2 clear
Clears the map.

13.37.3 destroy
Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.37.4 [const] boolean destroyed
Tell, if the object was destroyed.

Return:	true	The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
	false	The object is still alive.

13.37.5 [const] LayerMap dup
Creates a copy of self.

Return:	LayerMap	The copy of self.

13.37.6 [const] boolean is_mapped?(LayerInfo layer)
Check, if a given physical layer is mapped.

Return:	true	The layer is mapped.
	false	???.

13.37.7 [const] integer logical(LayerInfo layer)
Returns the logical layer (the layer index in the layout object) for a given physical layer.

Input:	layer	The physical layer specified with a LayerInfo object.
Return:	integer	The logical layer index, or
	-1	if the layer is not mapped.

13.37.8 map(LayerInfo phys_layer, unsigned log_layer)
Maps a physical layer to a logical one.

In general, there may be more than one physical layer mapped to one logical layer. This method will add the given physical layer to the mapping for the logical layer.

Input:	phys_layer	The physical layer (a LayerInfo object).
	unsigned log_layer	The logical layer to which the physical layer is mapped.

13.37.9 map(LayerInfo phys_layer, unsigned log_layer, LayerInfo target_layer)
Maps a physical layer to a logical one with a target layer.

In general, there may be more than one physical layer mapped to one logical layer. This method will add the given physical layer to the mapping for the logical layer.

This method has been added in version 0.20.

Input:	phys_layer	The physical layer (a LayerInfo object).
	unsigned log_layer	The logical layer to which the physical layer is mapped.
	target_layer	The properties of the layer that will be created unless it already exists.

13.37.10 map(LayerInfo pl_start, LayerInfo pl_stop, unsigned log_layer)
Maps a physical layer interval to a logical one.

This method maps an interval of layers l1 … l2 and data types d1 … d2 to the mapping for the given logical layer. l1 and d1 are given by the pl_start argument, while l2 and d2 are given by the pl_stop argument.

Input:	pl_start	The first physical layer (a LayerInfo object).
	pl_stop	The last physical layer (a LayerInfo object).
	unsigned log_layer	The logical layer to which the physical layers are mapped.

13.37.11 map(LayerInfo pl_start, LayerInfo pl_stop, unsigned log_layer, LayerInfo target_layer)
Maps a physical layer interval to a logical one with a target layer.

This method maps an interval of layers l1 … l2 and data types d1 … d2 to the mapping for the given logical layer. l1 and d1 are given by the pl_start argument, while l2 and d2 are given by the pl_stop argument.

This method has been added in version 0.20.

Input:	pl_start	The first physical layer (a LayerInfo object).
	pl_stop	The last physical layer (a LayerInfo object).
	unsigned log_layer	The logical layer to which the physical layers are mapped.
	target_layer	The properties of the layer that will be created unless it already exists.

13.37.12 map(map_expr, unsigned log_layer)
Maps a physical layer given by a string to a logical one.

The string expression is constructed using the syntax: "list[/list][;..]" for layer/data type pairs. "list" is a sequence of numbers, separated by comma values or a range separated by a hyphen. Examples are: "1/2", "1-5/0", "1,2,5/0", "1/5;5/6".

A target layer can be specified with the ":<target>" notation where the target is a valid layer specification string (i.e. "1/0").

Target mapping has been added in version 0.20.

Input:	map_expr	The string describing the physical layer to map.
	unsigned log_layer	The logical layer to which the physical layers are mapped.

13.37.13 [const] LayerInfo mapping(unsigned log_layer)
Returns the mapped physical (or target if one is specified) layer for a given logical layer.

In general, there may be more than one physical layer mapped to one logical layer. This method will return a single one of them. It will return the one with the lowest layer and data type.

Input:	unsigned log_layer	The logical layer to which the physical layers are mapped.
Return:	LayerInfo	A LayerInfo object which is the physical layer mapped to the logical layer.

13.37.14 [const] string mapping_str(unsigned log_layer)
Returns the mapping string for a given logical layer.

The mapping string is compatible with the string that the map method accepts.

Input:	unsigned log_layer	The logical layer to which the physical layers are mapped.
Return:	string	A string describing the mapping.

13.38 Class LayerProperties (version 0.21) The layer properties structure.

The layer properties encapsulate the settings relevant for the display and source of a layer.

Each attribute is present in two incarnations: local and real. "real" refers to the effective attribute after collecting the attributes from the parents to the leaf property node. In the spirit of this distinction, all read accessors are present in "local" and "real" form. The read accessors take a boolean parameter "real" that must be set to true, if the real value shall be returned.

"brightness" is an index that indicates how much to make the color brighter to darker rendering the effective color (eff_frame_color, eff_fill_color). It's value is roughly between -255 and 255.

Method Overview

==	Equality.
!=	Inequality.
eff_frame_color	Get the effective frame color.
eff_fill_color	Get the effective frame color.
frame_color	Get the frame color.
frame_color=	Set the frame color to the given value.
clear_frame_color	Reset the frame color.
has_frame_color?	Test, if the frame color is set.
fill_color	Get the fill color.
fill_color=	Set the fill color to the given value.
clear_fill_color	Reset the fill color.
has_fill_color?	Test, if the frame color is set.
frame_brightness=	Set the frame brightness.
frame_brightness	Get the frame brightness value.
fill_brightness=	Set the fill brightness.
fill_brightness	Get the fill brightness value.
flat	Return the "flattened" object.
dither_pattern=	Set the dither pattern index.
eff_dither_pattern	Get the effective dither pattern index.
dither_pattern	Get the dither pattern index.
clear_dither_pattern	Clear the dither pattern.
has_dither_pattern?	Test, if the dither pattern is set.
visible=	Set the visibility state.
visible?	Get the visibility state.
transparent=	Set the transparency state.
transparent?	Get the transparency state.
width=	Set the line width to the given width.
width	Get the line width.
marked=	Set the marked state.
marked?	Get the marked state.
animation=	Set the animation state.
animation	Get the animation state.
name=	Set the name to the given string.
name	Get the name.
trans	Get the transformations that the layer is transformed with.
trans=	Set the transformations that the layer is transformed with.
source_cellview	Get the cell view index that this layer refers to.
source_cellview=	Set the cell view index that this layer refers to.
source_layer_index	Get the layer index that the shapes are taken from.

source_layer_index=	Set the layer index specification that the shapes are taken from.
source_layer	Get the stream layer that the shapes are taken from.
source_layer=	Set the stream layer that the shapes are taken from.
source_datatype	Get the stream data type that the shapes are taken from.
source_datatype=	Set the stream data type that the shapes are taken from.
clear_source_name	Remove any stream layer name specification from this layer.
source_name	Get the stream name that the shapes are taken from.
has_source_name	Tell, if a stream layer name is specified for this layer.
source_name=	Set the stream layer name that the shapes are taken from.
upper_hier_level	The upper hierarchy level shown.
upper_hier_level_relative	Specifies if the upper hierarchy level is relative.
upper_hier_level_mode	Specifies the mode for the upper hierarchy level.
upper_hier_level=	Specify a upper hierarchy level.
set_upper_hier_level	Specify the upper hierarchy level and if it is relative to the context cell.
set_upper_hier_level	Specify the upper hierarchy level, if it is relative to the context cell and the mode.
has_upper_hier_level?	True, if a upper hierarchy level is explicitly specified.
clear_upper_hier_level	Disable a upper hierarchy level specification.
lower_hier_level	The lower hierarchy level shown.
lower_hier_level_relative	Specifies if the lower hierarchy level is relative..
lower_hier_level_mode	Specifies the mode for the lower hierarchy level.
lower_hier_level=	Specify a lower hierarchy level.
set_lower_hier_level	Specify the lower hierarchy level and if it is relative to the context cell.
set_lower_hier_level	Specify the lower hierarchy level, if it is relative to the context cell and the mode.
has_lower_hier_level?	True, if a lower hierarchy level is explicitly specified.
clear_lower_hier_level	Disable a lower hierarchy level specification.
source	The source specification.
source=	Load the source specification from a string.
cellview	Access to the cell view index.
layer_index	Access to the layer index.
assign	Assign the contents of another object to self.
dup	Creates a copy of self..
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.38.1 [const] boolean !=(LayerProperties other)

Inequality test.

Input:	other	The other object to compare against.
Return:	true	Inequality.
	false	???.

13.38.2 [const] boolean ==(LayerProperties other)

Equality test.

Input:	other	The other object to compare against.
Return:	true	Equality.
	false	???.

13.38.3 [const] integer animation(boolean)
Get the animation state.

Input:	true	Return the real value.
	false	Return the local value.
Return:	integer	The animation state is an integer either being
	0	static,
	1	scrolling,
	2	blinking or
	3	inversely blinking.

13.38.4 animation=(integer)
Set the animation state.

See the description of the animation method for details about the animation state.

13.38.5 assign(LayerProperties other)
Assign the contents of another object to self.

This method assigns the contents of another object to self. This is a deep copy that does not only copy the reference but the actual content.

13.38.6 [const] integer cellview
Access to the cell view index.

This is the index of the actual cell view to use. Basically, this method returns source_cellview in "real" mode. The result may be different, if the cell view is not valid for example. In this case, a negative value is returned.

13.38.7 clear_dither_pattern
Clear the dither pattern.

13.38.8 clear_fill_color
Reset the fill color.

13.38.9 clear_frame_color
Reset the frame color.

13.38.10 clear_lower_hier_level
Disable a lower hierarchy level specification.

See has_lower_hier_level? for a description of this property.

13.38.11 clear_source_name
Remove any stream layer name specification from this layer.

13.38.12 clear_upper_hier_level
Disable a upper hierarchy level specification.

See has_upper_hier_level? for a description of this property.

13.38.13 destroy
Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.38.14 [const] boolean destroyed
Tell, if the object was destroyed.

Return:	true	The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
	false	The object is still alive.

13.38.15 [const] integer dither_pattern(boolean)
Get the dither pattern index.

This method may deliver an invalid dither pattern index if it is not set.

Input:	true	Return the real value.
	false	Return the local value.
Return:	integer	The dither pattern index.

13.38.16 dither_pattern=(integer)
Set the dither pattern index.

The dither pattern index must be one of the valid indices. Indices 0 to 31 denote built-in pattern, indices above 32 denote one of the custom pattern. Index 0 is always solid filled and 1 is always the hollow filled pattern. **Input:** integer The dither pattern index.

13.38.17 [const] LayerProperties dup
Creates a copy of self.

Return:	LayerProperties	The copy of self.

13.38.18 [const] unsigned eff_dither_pattern(boolean)
Get the effective dither pattern index.

The effective dither pattern index is always a valid index, even if no dither pattern is set.

Input:	true	Return the real value.
	false	Return the local value.
Return:	unsigned	The effective dither pattern index.

13.38.19 [const] unsigned eff_fill_color(boolean)
Get the effective fill color.

The effective fill color is computed from the frame color brightness and the frame color.

Input:	true	Return the real value.
	false	Return the local value.
Return:	unsigned	The effective fill color.

13.38.20 [const] unsigned eff_frame_color(boolean)
Get the effective frame color.

The effective fill color is computed from the frame color brightness and the frame color.

Input:	true	Return the real value.
	false	Return the local value.
Return:	unsigned	The effective frame color.

13.38.21 [const] integer fill_brightness(boolean)
Get the fill brightness value.

If the brightness is not set, this method may return an invalid value.

Input:	true	Return the real value.
	false	Return the local value.
Return:	integer	The fill brightness value.

13.38.22 fill_brightness=(integer)
Set the fill brightness.

For neutral brightness set this value to 0. For darker colors set it to a negative value (down to -255), for brighter colors to a positive value (up to 255)

Input:	integer	The fill brightness.

13.38.23 [const] integer fill_color(boolean)
Get the fill color.

This method may return an invalid color if the color is not set.

Input:	true	Return the real value.
	false	Return the local value.
Return:	integer	The fill color.

13.38.24 fill_color=(unsigned)
Set the fill color to the given value.

The color is a 32 bit value encoding the blue value in the lower 8 bits, the green value in the next 8 bits and the red value in the 8 bits above that.

Input:	unsigned	The fill color.

13.38.25 flat
??.

Comment: Method not described.

13.38.26 [const] integer frame_brightness(boolean)
Get the frame brightness.

For neutral brightness set this value to 0. For darker colors set it to a negative value (down to -255), for brighter colors to a positive value (up to 255)

Input:	true	Return the real value.
	false	Return the local value.
Return:	integer	The frame color.

13.38.27 frame_brightness=(integer)
Set the frame brightness.

If the brightness is not set, this method may return an invalid value.

Input:	integer	The frame brightness.

13.38.28 frame_color(integer)
Get the frame color.

If the brightness is not set, this method may return an invalid value.

Input:	true	Return the real value.
	false	Return the local value.
Return:	integer	The frame color.

13.38.29 frame_color=(integer)
Set the frame color.

The color is a 32 bit value encoding the blue value in the lower 8 bits, the green value in the next 8 bits and the red value in the 8 bits above that..

Input:	integer	The frame color.

13.38.30 [const] boolean has_dither_pattern?(boolean)
Test, if the dither pattern is set.

Input:	true	Return the real value.
	false	Return the local value.
Return:	true	The dither pattern is set.
	false	The dither pattern is not set.

13.38.31 [const] boolean has_fill_color?(boolean)
Test, if the fill color is set.

Input:	true	Return the real value.
	false	Return the local value.
Return:	true	The fill color is set.
	false	The fill color is not set.

13.38.32 [const] boolean has_frame_color?(boolean)
Test, if the frame color is set.

Input:	true	Return the real value.
	false	Return the local value.
Return:	true	The frame color is set.
	false	The frame color is not set.

13.38.33 [const] boolean has_lower_hier_level?(boolean)
True, if a lower hierarchy level is explicitly specified.

Input:	true	Return the real value.
	false	Return the local value.
Return:	true	A lower hierarchy level is explicitly specified.
	false	No lower hierarchy level is explicitly specified.

13.38.34 [const] boolean has_source_name(boolean)
Tell, if a stream layer name is specified for this layer.

Input:	true	Return the real value.
	false	Return the local value.
Return:	true	A stream layer name is specified for this layer.
	false	No stream layer name is specified for this layer.

13.38.35 [const] boolean has_upper_hier_level?(boolean)
True, if a upper hierarchy level is explicitly specified.

Input:	true	Return the real value.
	false	Return the local value.
Return:	true	An upper hierarchy level is explicitly specified.
	false	No upper hierarchy level is explicitly specified.

13.38.36 [const] integer layer_index
Access to the layer index.

This is the index of the actual layer used. The source specification given by source_layer, source_datatype, source_name is evaluated and the corresponding layer is looked up in the layout object. If a source_layer_index is specified, this layer index is taken as the layer index to use.

Return:	integer	The layer index.

13.38.37 [const] integer lower_hier_level(boolean)
The lower hierarchy level shown.

This is the hierarchy level at which the drawing starts. This property is only meaningful, if has_lower_hier_level? is true. The hierarchy level can be relative in which case, 0 refers to the context cell's level. A mode can be specified for the hierarchy level which is 0 for absolute, 1 for minimum of specified level and set level and 2 for maximum of specified level and set level.

Input:	true	Return the real value.
	false	Return the local value.
Return:	integer	The lower hierarchy level.

13.38.38 lower_hier_level=(integer)
Specify a lower hierarchy level.

If this method is called, the lower hierarchy level is enabled. See lower_hier_level for a description of this property.

13.38.39 [const] integer lower_hier_level_mode(boolean)
Specifies the mode for the lower hierarchy level.

See lower_hier_level for a description of this property.

This method has been introduced in version 0.20.

Comment: Really a boolean as input argument?

Input:	true	Set the lower hierarchy level to relative.
	false	Set the lower hierarchy level to absolute.
Return:	integer	???.

13.38.40 [const] boolean lower_hier_level_relative(boolean)
Specifies if the lower hierarchy level is relative.

See lower_hier_level for a description of this property.

This method has been introduced in version 0.19.

Input:	true	Set the lower hierarchy level to relative.
	false	Set the lower hierarchy level to absolute.
Return:	true	???.
	false	???.

13.38.41 marked=(boolean)
Set the marked state.

Input:	true	Set the marked state.
	false	Unset the marked state.

13.38.42 [const] boolean marked?(boolean)
Get the marked state.

Input:	true	Return the real value.
	false	Return the local value.
Return:	true	The marked state is set.
	false	The marked state is unset.

Comment: Check input argument and return value.

13.38.43 [const] string name
Get the name.

Return:	integer	The layer name.

13.38.44 name=(string)
Set the name to the given string.

Input:	integer	The layer name.

13.38.45 set_lower_hier_level(level, boolean[, mode])
Specify the lower hierarchy level, if it is relative to the context cell [and the mode].

If this method is called, the lower hierarchy level is enabled. See lower_hier_level for a description of this property.

This method has been extended by mode selection in version 0.20.

This method (w/o mode selection) has been introduced in version 0.19.

Input:	level	The lower hierarchy level.
	true	Set relative to the context cell.
	false	Set absolute to the context cell.
	mode	The mode.

13.38.46 set_upper_hier_level(level, boolean[, mode])
Specify the upper hierarchy level, if it is relative to the context cell [and the mode].

If this method is called, the lower hierarchy level is enabled. See upper_hier_level for a description of this property.

This method has been extended by mode selection in version 0.20.

This method (w/o mode selection) has been introduced in version 0.19.

Input:	level	The upper hierarchy level.
	true	Set relative to the context cell.
	false	Set absolute to the context cell.
	mode	The mode.

13.38.47 [const] string source(boolean)
The source specification.

Input:	true	Return the real value.
	false	Return the local value.
Return:	string	The source specification.

13.38.48 source=(string)
Load the source specification from a string.

Input:	string	Sets the source specification to the given string. The source specification may contain the cell view index, the source layer (given by layer/data type or layer name), transformation, property selector etc. This method throws an exception if the specification is not valid.

Comment: Syntax?

13.38.49 [const] integer source_cellview(boolean)
Get the cell view index that this layer refers to.

Input:	true	Return the real value.
	false	Return the local value.
Return:	integer	The cell view index that this layer refers to.

13.38.50 source_cellview=(integer)
Set the cell view index that this layer refers to.

See cellview for a description of the transformations.

Input: integer — The index of the actual cell view to use. Basically, this method returnssource_-cellview in "real" mode. The result may be different, if the cell view is not valid for example. In this case, a negative value is returned.

13.38.51 [const] integer source_datatype(boolean)
Get the stream data type that the shapes are taken from.

Input: true — Return the real value.
false — Return the local value.
Return: integer — The stream data type that the shapes are taken from.
If the data type is positive, the actual layer is looked up by this stream data type.
If a name or layer index is specified, the stream data type is not used.

13.38.52 source_datatype=(integer)
Set the stream data type that the shapes are taken from.

See source_datatype for a description of this property.

Input: integer — The stream data type that the shapes are taken from.

13.38.53 [const] integer source_layer(boolean)
Get the stream layer that the shapes are taken from.

Input: true — Return the real value.
false — Return the local value.
Return: integer — The stream layer that the shapes are taken from.
If the layer is positive, the actual layer is looked up by this stream layer.
If a name or layer index is specified, the stream layer is not used.

13.38.54 source_layer=(integer)
Set the stream layer that the shapes are taken from.

See source_layer for a description of this property.

Input: integer — The stream layer that the shapes are taken from.

13.38.55 [const] integer source_layer_index(boolean)
Get the layer index that the shapes are taken from.

Input: true — Return the real value.
false — Return the local value.
Return: integer — The layer index that the shapes are taken from.
If the layer index is positive, the shapes drawn are taken from this layer rather than searched for by layer and data type.
This property is stronger than the layer/data type or name specification.

The similar method layer_index returns the actual layer index used, not the given one. The latter may be negative indicating that layer/data type or name specifications are used.

13.38.56 source_layer_index=(integer)
Set the layer index specification that the shapes are taken from.

See source_layer_index for a description of this property.

13.38.57 [const] string source_name(boolean)
Get the stream name that the shapes are taken from.

Input:	true	Return the real value.
	false	Return the local value.
Return:	string	The stream name that the shapes are taken from. If the name is non-empty, the actual layer is looked up by this stream layer name. If a layer index (see layer_index) is specified, the stream data type is not used. A name is only meaningful for OASIS files.

13.38.58 source_name=(string)
Set the stream layer name that the shapes are taken from.

See name for a description of this property.

13.38.59 [const] CplxTrans[] trans(boolean)
Get the transformations that the layer is transformed with.

The transformations returned by this accessor is the one used for displaying this layer. The layout is transformed with each of these transformations before it is drawn.

Input:	true	Return the real value.
	false	Return the local value.
Return:	CplxTrans[]	The returned transformations is the one used for displaying this layer. The layout is transformed with each of these transformations before it is drawn.

13.38.60 CplxTrans(trans= t_vector[])
Set the transformations that the layer is transformed with.

See trans for a description of the transformations.

13.38.61 transparent=(boolean)
Set the transparency state.

Input:	true	Set the transparency state.
	false	Set the opaque state.

13.38.62 [const] boolean transparent?(boolean)
Get the transparency state.

Input:	true	Return the real value.
	false	Return the local value.
Return:	true	The transparency state is set.
	false	The opaque state is set.

13.38.63 [const] integer upper_hier_level(boolean)
The upper hierarchy level shown.

This is the hierarchy level at which the drawing ends. This property is only meaningful, if has_upper_-hier_level? is true. The hierarchy level can be relative in which case, 0 refers to the context cell's level. A mode can be specified for the hierarchy level which is 0 for absolute, 1 for minimum of specified level and set level and 2 for minimum of specified level and set level.

Input:	true	Return the real value.
	false	Return the local value.
Return:	integer	The lower hierarchy level.

13.38.64 upper_hier_level=(integer)
Specify a upper hierarchy level.

If this method is called, the lower hierarchy level is enabled. See upper_hier_level for a description of this property.

13.38.65 [const] integer upper_hier_level_mode(boolean)
Specifies the mode for the upper hierarchy level.

See upper_hier_level for a description of this property.

This method has been introduced in version 0.20.

Comment: Really a boolean as input argument?

13.38.66 [const] boolean upper_hier_level_relative(boolean)
Specifies if the upper hierarchy level is relative.

See upper_hier_level for a description of this property.

This method has been introduced in version 0.19.

Input:	true	Set the upper hierarchy level to relative.
	false	Set the upper hierarchy level to absolute.
Return:	true	???.
	false	???.

13.38.67 visible=(boolean)
Set the visibility state.

Input:	true	Set the visibility state.
	false	Set the invisibility state.

13.38.68 [const] boolean visible?(boolean)
Get the visibility state.

Input:	true	Return the real value.
	false	Return the local value.
Return:	true	The visibility state is set.
	false	The invisibility state is set.

13.38.69 width=(integer)
Set the line width to the given width.

Input:	integer	The line width.

13.38.70 [const] integer width(boolean)
Get the line width.

Input:	true	Return the real value.
	false	Return the local value.
Return:	integer	The line width.

13.39 Class LayerPropertiesIterator (version 0.21) Flat layer iterator.

This iterator provides a flat view for the layers in the layer tree.

Method Overview

!=	Inequality test.
==	Equality test.
<	Comparison.
at_top?	At-the-top property.
at_end?	At-the-end property.
is_null?	"is null" predicate.
next	Increment operator.
up	Move up.
next_sibling	Move to the next sibling by a given distance.
to_sibling	Move to the sibling with the given index.
num_siblings	Return the number of siblings.
down_first_child	Move to the first child.
down_last_child	Move to the last child.
current	Access to the current element.
parent	Obtain the parent iterator.
first_child	Obtain the iterator pointing to the first child.
last_child	Obtain the iterator pointing to the last child.
child_index	Obtain the index of the child within the parent.
assign	Assign the contents of another object to self.
dup	Creates a copy of self.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.39.1 [const] boolean !=(LayerPropertiesIterator other) Inequality test.

Input:	other	The other object to compare against.
Return:	true	The objects are not equal.
	false	???.

13.39.2 [const] boolean <(LayerPropertiesIterator other) Comparison.

Input:	other	The other object to compare against.
Return:	true	Self points to an object that comes before other.
	false	???.

13.39.3 [const] boolean ==(LayerPropertiesIterator other) Equality test.

Input:	other	The other object to compare against.
Return:	true	The objects are equal.
	false	???.

13.39.4 assign(LayerPropertiesIterator other)
Assign the contents of another object to self.

This method assigns the contents of another object to self. This is a deep copy that does not only copy the reference but the actual content.

13.39.5 [const] boolean at_end?
At-the-end property.

Return: true — The iterator is at the end of either all elements or at the end of the child list (if down_last_child or down_first_child is used to iterate).
false — ???.

13.39.6 [const] boolean at_top?
At-the-top property.

Return: true — At top - there is no parent level.
false — ???.

13.39.7 [const] unsigned child_index
Obtain the index of the child within the parent.

Return: unsigned — The index in the list of children of it's parent, that the element pointed to.
If the element does not have a parent, the index of the element in the global list.

13.39.8 [const] const ref current
Access to the current element.

Return: ref — The reference to the current element.

13.39.9 destroy
Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.39.10 [const] boolean destroyed
Tell, if the object was destroyed.

Return: true — The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
false — The object is still alive.

13.39.11 down_first_child
Move to the first child.

This method moves to the first child of the current element. If there is no child, at_end? will be true. Even then, the iterator points to the child level and method up can be used to move back.

13.39.12 down_last_child
Move to the last child.

This method moves to the last child of the current element. If there is no child, at_end? will be true. Even then, the iterator points to the child level and method up can be used to move back.

13.39.13 [const] LayerPropertiesIterator dup
Creates a copy of self.

Return: LayerPropertiesIterator The copy of self.

13.39.14 [const] LayerPropertiesIterator first_child
Obtain the iterator pointing to the first child.

Return: LayerPropertiesIterator Obtain the iterator pointing to the first child.
If there is no children, the iterator will be a valid insert point but not pointing to any valid element. It will report at_end? = true.

13.39.15 [const] boolean is_null?
"is null" predicate.

Return: true The iterator is "null". Such an iterator can be created with the default constructor or by moving a top-level iterator up.
false ???.

13.39.16 [const] LayerPropertiesIterator last_child
Obtain the iterator pointing to the first child.

Return: LayerPropertiesIterator Obtain the iterator pointing to the last child.
If there is no children, the iterator will be a valid insert point but not pointing to any valid element. It will report at_end? = true.

13.39.17 ref next
Increment operator.

The iterator will be incremented to point to the next layer entry. It will descend into the hierarchy to address children if there are any.

13.39.18 next_sibling(n)
Move to the next sibling by a given distance.

The iterator is moved to the nth next sibling of the current element.

Input: n The distance to move.

13.39.19 [const] unsigned num_siblings
Return the number of siblings.

Return: unsigned The number of siblings.

13.39.20 [const] LayerPropertiesIterator parent
Obtain the parent iterator.

Return: LayerPropertiesIterator Obtain the iterator pointing to parent.
If there is no parent, the returned iterator will "null".

13.39.21 to_sibling(index)
Move to the sibling with the given index.

The iterator is moved to the nth next sibling of the current element.

Input: index The given index.

13.39.22 ref up
Move up.

The iterator is moved to point to the current element's parent. If the current element does not have a parent, the iterator will be undefined.

13.40 Class LayerPropertiesNode (version 0.21) The layer properties structure.

This class is derived from LayerProperties. Objects of this class are used in the hierarchy of layer views that are arranged in a tree while the LayerProperties object reflects the properties of a single node.

Method Overview

==	Equality test on LayerProperties.
!=	Inequality test on LayerProperties.
flat	Return the "flattened" object.
has_children?	Test, if there are children.
bbox	Compute the bounding box of this layer.
id	Obtain the unique ID.
==	Equality test on LayerPropertiesNode.
!=	Inequality test on LayerPropertiesNode.
eff_frame_color	Get the effective frame color.
eff_fill_color	Get the effective frame color.
frame_color	Get the frame color.
frame_color=	Set the frame color to the given value.
clear_frame_color	Reset the frame color.
has_frame_color?	Test, if the frame color is set.
fill_color	Get the fill color.
fill_color=	Set the fill color to the given value.
clear_fill_color	Reset the fill color.
has_fill_color?	Test, if the frame color is set.
frame_brightness=	Set the frame brightness.
frame_brightness	Get the frame brightness value.
fill_brightness=	Set the fill brightness.
fill_brightness	Get the fill brightness value.
dither_pattern=	Set the dither pattern index.
eff_dither_pattern	Get the effective dither pattern index.
dither_pattern	Get the dither pattern index.
clear_dither_pattern	Clear the dither pattern.
has_dither_pattern?	Test, if the dither pattern is set.
visible=	Set the visibility state.
visible?	Get the visibility state.
transparent=	Set the transparency state.
transparent?	Get the transparency state.
width=	Set the line width to the given width.
width	Get the line width.
marked=	Set the marked state.
marked?	Get the marked state.
animation=	Set the animation state.
animation	Get the animation state.
name=	Set the name to the given string.
name	Get the name.
trans	Get the transformations that the layer is transformed with.
trans=	Set the transformations that the layer is transformed with.
source_cellview	Get the cell view index that this layer refers to.
source_cellview=	Set the cell view index that this layer refers to.
source_layer_index	Get the layer index that the shapes are taken from.
source_layer_index=	Set the layer index specification that the shapes are taken from.

source_layer	Get the stream layer that the shapes are taken from.
source_layer=	Set the stream layer that the shapes are taken from.
source_datatype	Get the stream data type that the shapes are taken from.
source_datatype=	Set the stream data type that the shapes are taken from.
clear_source_name	Remove any stream layer name specification from this layer.
source_name	Get the stream name that the shapes are taken from.
has_source_name	Tell, if a stream layer name is specified for this layer.
source-name=	Set the stream layer name that the shapes are taken from.
upper_hier_level	The upper hierarchy level shown.
upper_hier_level_relative	Specifies if the upper hierarchy level is relative.
upper_hier_level_mode	Specifies the mode for the upper hierarchy level.
upper_hier_level=	Specify a upper hierarchy level.
set_upper_hier_level	Specify the upper hierarchy level and if it is relative to the context cell.
set_upper_hier_level	Specify the upper hierarchy level, if it is relative to the context cell and the mode.
has_upper_hier_level?	True, if a upper hierarchy level is explicitly specified.
clear_upper-hier_level	Disable a upper hierarchy level specification.
lower_hier_level	The lower hierarchy level shown.
lower_hier_level_relative	Specifies if the lower hierarchy level is relative..
lower_hier_level_mode	Specifies the mode for the lower hierarchy level.
lower_hier_level=	Specify a lower hierarchy level.
set_lower_hier_level	Specify the lower hierarchy level and if it is relative to the context cell.
set_lower_hier_level	Specify the lower hierarchy level, if it is relative to the context cell and the mode.
has_lower_hier_level?	True, if a lower hierarchy level is explicitly specified.
clear_lower_hier_level	Disable a lower hierarchy level specification.
source	The source specification.
source=	Load the source specification from a string.
cellview	Access to the cell view index.
layer_index	Access to the layer index.
assign	Assign the contents of another object to self.
dup	Creates a copy of self.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.40.1 [const] boolean !=(LayerProperties other)

Inequality test.

Input:	other	The other object to compare against.
Return:	true	Inequality.
	false	???.

13.40.2 [const] boolean !=(LayerPropertiesNode other)

Inequality test.

Input:	other	The other object to compare against.
Return:	true	Inequality.
	false	???.

13.40.3 [const] boolean ==(LayerProperties other)
Equality test.

Input:	other	The other object to compare against.
Return:	true	Equality.
	false	???.

13.40.4 [const] boolean ==(LayerPropertiesNode other)
Equality test.

Input:	other	The other object to compare against.
Return:	true	Equality.
	false	???.

13.40.5 [const] integer animation(boolean)
Get the animation state.

Return:		The animation state is an integer either being
	0	static,
	1	scrolling,
	2	blinking or
	3	inversely blinking.

13.40.6 animation=(integer)
Set the animation state.

See the description of the animation method for details about the animation state.

13.40.7 assign(LayerPropertiesNode other)
Assign the contents of another object to self.

This method assigns the contents of another object to self. This is a deep copy that does not only copy the reference but the actual content.

13.40.8 [const] DBox bbox
Compute the bbox of this layer.

This takes the layout and path definition (supported by the given default layout or path, if no specific is given). The node must have been attached to a view to make this operation possible.

Return:	DBox	A bbox in micron units.

13.40.9 [const] integer cellview
Access to the cell view index.

This is the index of the actual cell view to use. Basically, this method returns source_cellview in "real" mode. The result may be different, if the cell view is not valid for example. In this case, a negative value is returned.

13.40.10 clear_dither_pattern
Clear the dither pattern.

13.40.11 clear_fill_color
Reset the fill color.

13.40.12 clear_frame_color
Reset the frame color.

13.40.13 clear_lower_hier_level
Disable a lower hierarchy level specification.

See has_lower_hier_level? for a description of this property.

13.40.14 clear_source_name
Remove any stream layer name specification from this layer.

13.40.15 clear_upper-hier_level
Disable a upper hierarchy level specification.

See has_upper_hier_level? for a description of this property.

13.40.16 destroy
Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.40.17 [const] boolean destroyed
Tell, if the object was destroyed.

Return:	true	The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
	false	The object is still alive.

13.40.18 [const] integer dither_pattern(boolean)
Get the dither pattern index.

This method may deliver an invalid dither pattern index if it is not set.

Input:	true	Return the real value.
	false	Return the local value.
Return:	integer	The dither pattern index.

13.40.19 dither_pattern=(integer)
Set the dither pattern index.

The dither pattern index must be one of the valid indices. Indices 0 to 31 denote built-in pattern, indices above 32 denote one of the custom pattern. Index 0 is always solid filled and 1 is always the hollow filled pattern.

Input:	integer	The dither pattern index.

13.40.20 [const] LayerPropertiesNode dup
Creates a copy of self.

Return: LayerPropertiesNode The copy of self.

13.40.21 [const] unsigned eff_dither_pattern(boolean)
Get the effective dither pattern index.

The effective dither pattern index is always a valid index, even if no dither pattern is set.

Input:	true	Return the real value.
	false	Return the local value.
Return:	unsigned	The effective dither pattern index.

13.40.22 [const] unsigned eff_fill_color(boolean)
Get the effective fill color.

The effective fill color is computed from the frame color brightness and the frame color.

Input:	true	Return the real value.
	false	Return the local value.
Return:	unsigned	The effective fill color.

13.40.23 [const] unsigned eff_frame_color(boolean)
Get the effective frame color.

The effective fill color is computed from the frame color brightness and the frame color.

Input:	true	Return the real value.
	false	Return the local value.
Return:	unsigned	The effective frame color.

13.40.24 [const] integer fill_brightness(boolean)
Get the fill brightness value.

If the brightness is not set, this method may return an invalid value.

Input:	true	Return the real value.
	false	Return the local value.
Return:	integer	The fill brightness value.

13.40.25 fill_brightness=(integer)
Set the fill brightness.

For neutral brightness set this value to 0. For darker colors set it to a negative value (down to -255), for brighter colors to a positive value (up to 255).

Input:	integer	The fill brightness.

13.40.26 [const] integer fill_color(boolean)
Get the fill color.

This method may return an invalid color if the color is not set.

Input:	true	Return the real value.
	false	Return the local value.
Return:	integer	The fill color.

13.40.27 fill_color=(unsigned)
Set the fill color to the given value.

The color is a 32 bit value encoding the blue value in the lower 8 bits, the green value in the next 8 bits and the red value in the 8 bits above that.

Input:	unsigned	The fill color.

13.40.28 [const] flat
Return the "flattened" object.

Contrary to what the name suggests, this method does not flatten the hierarchy but rather returns an object that does not need a parent for the "real" properties. See flat for a description of this process. The child list of the returned object will be the same that of the original object.

13.40.29 [const] integer frame_brightness(boolean)
Get the frame brightness.

For neutral brightness set this value to 0. For darker colors set it to a negative value (down to -255), for brighter colors to a positive value (up to 255)

Input:	true	Return the real value.
	false	Return the local value.
Return:	integer	The frame color.

13.40.30 frame_brightness=(integer)
Set the frame brightness.

If the brightness is not set, this method may return an invalid value.

Input:	integer	The frame brightness.

13.40.31 frame_color(integer)
Get the frame color.

If the brightness is not set, this method may return an invalid value.

Input:	true	Return the real value.
	false	Return the local value.
Return:	integer	The frame color.

13.40.32 frame_color=(integer)
Set the frame color.

The color is a 32 bit value encoding the blue value in the lower 8 bits, the green value in the next 8 bits and the red value in the 8 bits above that..

Input:	integer	The frame color.

13.40.33 [const] boolean has_children?
Test, if there are children.

Return:	true	There are children.
	false	There are no children.

13.40.34 [const] boolean has_dither_pattern?(boolean)
Test, if the dither pattern is set.

Input:	true	Return the real value.
	false	Return the local value.
Return:	true	The dither pattern is set.
	false	The dither pattern is not set.

13.40.35 [const] boolean has_fill_color?(boolean)
Test, if the fill color is set.

Input:	true	Return the real value.
	false	Return the local value.
Return:	true	The fill color is set.
	false	The fill color is not set.

13.40.36 [const] boolean has_frame_color?(boolean)
Test, if the frame color is set.

Input:	true	Return the real value.
	false	Return the local value.
Return:	true	The frame color is set.
	false	The frame color is not set.

13.40.37 [const] boolean has_lower_hier_level?(boolean)
True, if a lower hierarchy level is explicitly specified.

Input:	true	Return the real value.
	false	Return the local value.
Return:	true	A lower hierarchy level is explicitly specified.
	false	No lower hierarchy level is explicitly specified.

13.40.38 [const] boolean has_source_name(boolean)
Tell, if a stream layer name is specified for this layer.

Input:	true	Return the real value.
	false	Return the local value.
Return:	true	A stream layer name is specified for this layer.
	false	No stream layer name is specified for this layer.

13.40.39 [const] boolean has_upper_hier_level?(boolean)
True, if a upper hierarchy level is explicitly specified.

Input:	true	Return the real value.
	false	Return the local value.
Return:	true	An upper hierarchy level is explicitly specified.
	false	No upper hierarchy level is explicitly specified.

13.40.40 [const] unsigned id
Obtain the unique ID.

Each layer properties node object has a unique ID that is created when a new LayerPropertiesNode object is instantiated. The ID is copied when the object is copied. The ID can be used to identify the object irregardless of it's content.

Return: unsigned The unique object ID.

13.40.41 [const] integer layer_index
Access to the layer index.

This is the index of the actual layer used. The source specification given by source_layer, source_datatype, source_name is evaluated and the corresponding layer is looked up in the layout object. If a source_layer_-index is specified, this layer index is taken as the layer index to use.

Return: integer The layer index.

13.40.42 [const] integer lower_hier_level(boolean)
The lower hierarchy level shown.

This is the hierarchy level at which the drawing starts. This property is only meaningful, if has_lower_-hier_level? is true. The hierarchy level can be relative in which case, 0 refers to the context cell's level. A mode can be specified for the hierarchy level which is 0 for absolute, 1 for minimum of specified level and set level and 2 for maximum of specified level and set level.

Input:	true	Return the real value.
	false	Return the local value.
Return:	integer	The lower hierarchy level.

13.40.43 lower_hier_level=(integer)
Specify a lower hierarchy level.

If this method is called, the lower hierarchy level is enabled. See lower_hier_level for a description of this property.

13.40.44 [const] integer lower_hier_level_mode(boolean)
Specifies the mode for the lower hierarchy level.

See lower_hier_level for a description of this property.

This method has been introduced in version 0.20.

Comment: Really a boolean as input argument?

13.40.45 [const] boolean lower_hier_level_relative(boolean)
Specifies if the lower hierarchy level is relative.

See lower_hier_level for a description of this property.

This method has been introduced in version 0.19.

Input:	true	Set the lower hierarchy level to relative.
	false	Set the lower hierarchy level to absolute.
Return:	true	???.
	false	???.

13.40.46 marked=(boolean)
Set the marked state.

Input:	true	Set the marked state.
	false	Unset the marked state.

13.40.47 [const] boolean marked?(boolean)
Get the marked state.

Input:	true	Return the real value.
	false	Return the local value.
Return:	true	The marked state is set.
	false	The marked state is unset.

Comment: Check input argument and return value.

13.40.48 [const] string name
Get the name.

Return:	integer	The layer name.

13.40.49 name=(string)
Set the name to the given string.

Input:	integer	The layer name.

13.40.50 set_lower_hier_level(level, boolean[, mode])
Specify the lower hierarchy level, if it is relative to the context cell [and the mode].

If this method is called, the lower hierarchy level is enabled. See lower_hier_level for a description of this property.

This method has been extended by mode selection in version 0.20.

This method (w/o mode selection) has been introduced in version 0.19.

Input:	level	The lower hierarchy level.
	true	Set relative to the context cell.
	false	Set absolute to the context cell.
	mode	The mode.

13.40.51 set_upper_hier_level(level, boolean[, mode]) Specify the upper hierarchy level, if it is relative to the context cell [and the mode].

If this method is called, the lower hierarchy level is enabled. See upper_hier_level for a description of this property.

This method has been extended by mode selection in version 0.20.

This method (w/o mode selection) has been introduced in version 0.19.

Input:	level	The upper hierarchy level.
	true	Set relative to the context cell.
	false	Set absolute to the context cell.
	mode	The mode.

13.40.52 [const] string source(boolean) The source specification.

Input:	true	Return the real value.
	false	Return the local value.
Return:	string	The source specification.

13.40.53 source=(string) Load the source specification from a string.

Input:	string	Sets the source specification to the given string. The source specification may contain the cell view index, the source layer (given by layer/data type or layer name), transformation, property selector etc. This method throws an exception if the specification is not valid.

Comment: Syntax?

13.40.54 [const] integer source_cellview(boolean) Get the cell view index that this layer refers to.

Input:	true	Return the real value.
	false	Return the local value.
Return:	integer	The cell view index that this layer refers to.

13.40.55 source_cellview=(integer) Set the cell view index that this layer refers to.

See cellview for a description of the transformations.

Input: integer — The index of the actual cell view to use. Basically, this method returnssource_cellview in "real" mode. The result may be different, if the cell view is not valid for example. In this case, a negative value is returned.

13.40.56 [const] integer source_datatype(boolean)
Get the stream data type that the shapes are taken from.

Input: true — Return the real value.
false — Return the local value.
Return: integer — The stream data type that the shapes are taken from.
If the data type is positive, the actual layer is looked up by this stream data type.
If a name or layer index is specified, the stream data type is not used.

13.40.57 source_datatype=(integer)
Set the stream data type that the shapes are taken from.

See source_datatype for a description of this property.

Input: integer — The stream data type that the shapes are taken from.

13.40.58 [const] integer source_layer(boolean)
Get the stream layer that the shapes are taken from.

Input: true — Return the real value.
false — Return the local value.
Return: integer — The stream layer that the shapes are taken from.
If the layer is positive, the actual layer is looked up by this stream layer.
If a name or layer index is specified, the stream layer is not used.

13.40.59 source_layer=(integer)
Set the stream layer that the shapes are taken from.

See source_layer for a description of this property.

Input: integer — The stream layer that the shapes are taken from.

13.40.60 [const] integer source_layer_index(boolean)
Get the layer index that the shapes are taken from.

Input: true — Return the real value.
false — Return the local value.
Return: integer — The layer index that the shapes are taken from.
If the layer index is positive, the shapes drawn are taken from this layer rather than searched for by layer and data type.
This property is stronger than the layer/data type or name specification.

The similar method layer_index returns the actual layer index used, not the given one. The latter may be negative indicating that layer/data type or name specifications are used.

13.40.61 source_layer_index=(integer)
Set the layer index specification that the shapes are taken from.

See source_layer_index for a description of this property.

13.40.62 [const] string source_name(boolean)
Get the stream name that the shapes are taken from.

Input:	true	Return the real value.
	false	Return the local value.
Return:	string	The stream name that the shapes are taken from. If the name is non-empty, the actual layer is looked up by this stream layer name. If a layer index (see layer_index) is specified, the stream data type is not used. A name is only meaningful for OASIS files.

13.40.63 source-name=(string)
Set the stream layer name that the shapes are taken from.

See name for a description of this property.

13.40.64 [const] CplxTrans[] trans(boolean)
Get the transformations that the layer is transformed with.

The transformations returned by this accessor is the one used for displaying this layer. The layout is transformed with each of these transformations before it is drawn.

Input:	true	Return the real value.
	false	Return the local value.
Return:	CplxTrans[]	The returned transformations is the one used for displaying this layer. The layout is transformed with each of these transformations before it is drawn.

13.40.65 CplxTrans(trans= t_vector[])
Set the transformations that the layer is transformed with.

See trans for a description of the transformations.

13.40.66 transparent=(boolean)
Set the transparency state.

Input:	true	Set the transparency state.
	false	Set the opaque state.

13.40.67 [const] boolean transparent?(boolean)
Get the transparency state.

Input:	true	Return the real value.
	false	Return the local value.
Return:	true	The transparency state is set.
	false	The opaque state is set.

13.40.68 [const] integer upper_hier_level(boolean)
The upper hierarchy level shown.

This is the hierarchy level at which the drawing ends. This property is only meaningful, if has_upper_-hier_level? is true. The hierarchy level can be relative in which case, 0 refers to the context cell's level. A mode can be specified for the hierarchy level which is 0 for absolute, 1 for minimum of specified level and set level and 2 for minimum of specified level and set level.

Input:	true	Return the real value.
	false	Return the local value.
Return:	integer	The lower hierarchy level.

13.40.69 upper_hier_level=(integer)
Specify the upper hierarchy level.

If this method is called, the lower hierarchy level is enabled. See upper_hier_level for a description of this property.

13.40.70 [const] integer upper_hier_level_mode(boolean)
Specifies the mode for the upper hierarchy level.

See upper_hier_level for a description of this property.

This method has been introduced in version 0.20.

Comment: Really a boolean as input argument?

13.40.71 [const] boolean upper_hier_level_relative(boolean)
Specifies if the upper hierarchy level is relative.

See upper_hier_level for a description of this property.

This method has been introduced in version 0.19.

Input:	true	Set the upper hierarchy level to relative.
	false	Set the upper hierarchy level to absolute.
Return:	true	???.
	false	???.

13.40.72 visible=(boolean)
Set the visibility state.

Input:	true	Set the visibility state.
	false	Set the invisibility state.

13.40.73 [const] boolean visible?(boolean)
Get the visibility state.

Input:	true	Return the real value.
	false	Return the local value.
Return:	true	The visibility state is set.
	false	The invisibility state is set.

13.40.74 width=(integer)
Set the line width to the given width.

Input:	integer	The line width.

13.40.75 [const] integer width(boolean)
Get the line width.

Input:	true	Return the real value.
	false	Return the local value.
Return:	integer	The line width.

13.41 Class Layout (version 0.21) The layout object.

The layout object basically wraps the cell graphs and adds functionality for managing cell names and layer names. The cell graph is a container for the cells and their hierarchical arrangement. The cell graph is constructed by creating cells and adding child instances to it.

Method Overview

new	Create a layout object attached to a manager.
new	Create a layout object.
clear	Clears the layout.
properties_id	Get the properties ID for a given properties set.
properties	Get the properties set for a given properties ID.
has_cell?	Tell, if the cell with a given name exists.
cell_by_name	Get the cell index for a given name.
cell_name	Get the name for a cell with the given index.
add_cell	Add a cell with the given name.
rename_cell	Rename a cell with the given name.
delete_cell	Delete a cell.
delete_cells	Delete multiple cells.
prune_subcells	Delete all sub cells of the cell which are not used otherwise down to the specified level of hierarchy.
prune_cell	Delete a cell plus sub cells not used otherwise.
delete_cell_rec	Delete a cell plus all sub cells.
flatten	Flatten the given cell.
start_changes	Signal the start of an operation bringing the layout into invalid state.
end_changes	Cancel the "in changes" state (see start_changes).
under_construction	Tell if the layout object is under construction.
update	Update the internals of the layout.
dbu=	Database unit write accessor.
dbu	Database unit read accessor.
insert_layer	Insert a new layer with the given properties.
insert_layer_at	Insert a new layer with the given properties at the given index.
insert_special_layer	Insert a new special layer with the given properties.
insert_special_layer_at	Insert a new special layer with the given properties at the given index.
set_info	Set the info structure for a specified layer.
get_info	Get the info structure for a specified layer.
cells	Return the number of cells.
cell	Address a cell by index.
each_cell	Iterate the unsorted cell list.
each_cell_bottom_up	Iterate the bottom-up sorted cell list.
each_cell_top_down	Iterate of the top-down sorted cell list.
each_top_cell	Iterate the top cells.
swap_layers	Swap layers.
move_layer	Move a layer.
copy_layer	Copy a layer.
clear_layer	Clear a layer.
delete_layer	Delete a layer.
layer_indices	Return a list of valid layer indices.
layers	Return the number of layers.
is_valid_cell_index?	Tell, if a cell index is a valid index.
is_valid_layer?	Tell, if a layer index is a valid index.

is_special_layer?	Tell, if a layer index is a special layer index.
begin_shapes	Delivers a recursive shape iterator for the shapes below the given cell on the given layer.
begin_shapes_touching	Delivers a recursive shape iterator for the shapes below the given cell on the given layer using a region search.
begin_shapes_overlapping	Delivers a recursive shape iterator for the shapes below the given cell on the given layer using a region search.
write	Write the layout to a stream file.
write	Write the layout to a stream file with options.
clip	Clips the given cell by the given rectangle and produce a new cell with the clip.
clip_into	Clips the given cell by the given rectangle and produce a new cell with the clip.
multi_clip	Clips the given cell by the given rectangles and produce new cells with the clips, one for each rectangle..
multi_clip_into	Clips the given cell by the given rectangles and produce new cells with the clips, one for each rectangle..
read	Load the layout from the given file.
read	Load the layout from the given file with options.
assign	Assign the contents of another object to self.
dup	Creates a copy of self..
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.41.1 unsigned add_cell(name)
Add a cell with the given name.

Input:	name	The given name.
Return:	unsigned	The index of the newly created cell.

13.41.2 assign(Layout other)
Assign the contents of another object to self.

This method assigns the contents of another object to self. This is a deep copy that does not only copy the reference but the actual content.

13.41.3 [const] RecursiveShapeIterator begin_shapes(unsigned cell_index, unsigned layer)
Delivers a recursive shape iterator for the shapes below the given cell on the given layer.

For details see the description of the RecursiveShapeIterator class.

This method has been added in version 0.18.

Input:	unsigned cell_index	The index of the starting cell.
	unsigned layer	The layer from which to get the shapes.
Return:	RecursiveShapeIterator	A suitable iterator.

13.41.4 [const] RecursiveShapeIterator begin_shapes_overlapping(unsigned cell_index, unsigned layer, Box region)

Delivers a recursive shape iterator for the shapes below the given cell on the given layer using a region search.

For details see the description of the RecursiveShapeIterator class. This version gives an iterator delivering shapes whose bounding box overlaps the given region.

This method has been added in version 0.18.

Input:	unsigned cell_index	The index of the starting cell.
	unsigned layer	The layer from which to get the shapes.
	Box region	The search region.
Return:	RecursiveShapeIterator	A suitable iterator.

13.41.5 [const] RecursiveShapeIterator begin_shapes_touching(unsigned cell_index, unsigned layer, Box region)

Delivers a recursive shape iterator for the shapes below the given cell on the given layer using a region search.

For details see the description of the RecursiveShapeIterator class. This version gives an iterator delivering shapes whose bounding box touches the given region.

This method has been added in version 0.18.

Input:	unsigned cell_index	The index of the starting cell.
	unsigned layer	The layer from which to get the shapes.
	Box region	The search region.
Return:	RecursiveShapeIterator	A suitable iterator.

13.41.6 ref Cell cell(unsigned i)

Address a cell by index.

Input:	unsigned i	The cell index.
Return:	ref Cell	A reference to the cell.

13.41.7 unsigned cell_by_name(name)

Get the cell index for a given name.

Input:	name	The given cell name.
Return:	unsigned	The associated cell index. If no cell with this name exists, an exception is thrown.

13.41.8 [const] name cell_name(unsigned)

Get the name for a cell with the given index.

Input:	unsigned	The given cell index.
Return:	name	The associated cell name.

13.41.9 [const] unsigned cells

Return the number of cells.

Return:	unsigned	The number of cells (the maximum cell index).

13.41.10 clear
Clears the layout.

Clears the layout completely.

13.41.11 clear_layer(unsigned layer_index)
Clear a layer.

Clears the layer: removes all shapes.

This method was introduced in version 0.19.

Input:	unsigned layer_index	The index of the layer to delete.

13.41.12 unsigned clip(unsigned cell_index, Box region)
Clips the given cell by the given rectangle and produce a new cell with the clip.

This method will cut a rectangular region given by the box from the given cell. The clip will be stored in a new cell whose index is returned. The clip will be performed hierarchically. The resulting cell will hold a hierarchy of child cells, which are potentially clipped versions of child cells of the original cell.

This method has been added in version 0.21.

Input:	unsigned cell_index	The cell index of the cell to clip.
	Box region	The search region.
Return:	unsigned	The index of the new cell.

13.41.13 unsigned clip_into(unsigned cell_index, ref Box box, Layout target)
Clips the given cell by the given rectangle and produce a new cell with the clip.

This method will cut a rectangular region given by the box from the given cell. The clip will be stored in a new cell in the target layout. The clip will be performed hierarchically. The resulting cell will hold a hierarchy of child cells, which are potentially clipped versions of child cells of the original cell.

Please note that it is important that the database unit of the target layout is identical to the database unit of the source layout to achieve the desired results.This method also assumes that the target layout holds the same layers than the source layout. It will copy shapes to the same layers than they have been on the original layout.

This method has been added in version 0.21.

Input:	unsigned cell_index	The cell index of the cell to clip.
	Box box	The clip box in database units.
	Layout target	The target layout.
Return:	unsigned	The index of the new cell in the target layout.

13.41.14 copy_layer(unsigned src, unsigned dest)
Copy a layer.

Copy a layer from the source to the target. The target is not cleared before, so that this method merges shapes from the source with the target layer.

This method was introduced in version 0.19.

Input: unsigned src The layer index of the source layer.
unsigned dest The layer index of the destination layer.

13.41.15 [const] double dbu
Database unit read accessor.

Return: double The database unit.

13.41.16 dbu=(double)
Database unit write accessor.

Input: double The database unit.

13.41.17 delete_cell(unsigned cell_index)
Delete a cell .

This deletes a cell but not the sub cells of the cell. These sub cells will likely become new top cells unless they are used otherwise. All instances of this cell are deleted as well.

Hint:: **To delete multiple cells, use delete_cells which is far more efficient in this case.**

This method has been introduced in version 0.20.

Input: unsigned cell_index The cell index of the cell to delete.

13.41.18 delete_cell_rec(unsigned cell_index)
Delete a cell plus all sub cells.

This deletes a cell and also all sub cells of the cell. In contrast to prune_cell, all cells are deleted together with their instances even if they are used otherwise.

This method has been introduced in version 0.20.

Input: unsigned cell_index The cell index of the cell to delete.

13.41.19 delete_cells(unsigned cell_index_list[])
Delete multiple cells.

This deletes the cells but not the sub cells of these cells. These sub cells will likely become new top cells unless they are used otherwise. All instances of these cells are deleted as well.

This method has been introduced in version 0.20.

Input: unsigned cell_index_list[] An array of cell indices of the cells to delete.

13.41.20 delete_layer(unsigned layer_index)
Delete a layer.

This does free the shapes of the cells and remembers the layer's index for recycling.

Input: unsigned layer_index The index of the layer to delete.

13.41.21 destroy
Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.41.22 [const] boolean destroyed
Tell, if the object was destroyed.

Return: true — The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
false — The object is still alive.

13.41.23 [const] Layout dup
Creates a copy of self.

Return: Layout — The copy of self.

13.41.24 yield ref Cell each_cell
Iterate the unsorted cell list.

Return: yield unsigned — An array of unsorted cell indices.

13.41.25 yield unsigned each_cell_bottom_up
Iterate the bottom-up sorted cell list.

In bottom-up traversal a cell is not delivered before the last child cell of this cell has been delivered. The bottom-up iterator does not deliver cells but cell indices actually.

Return: yield unsigned — An array of cell indices bottom-up sorted.

13.41.26 yield unsigned each_cell_top_down
Iterate the top-down sorted cell list.

The top-down cell list has the property of delivering all cells before they are instantiated. In addition the first cells are all top cells. There is at least one top cell. The top-down iterator does not deliver cells but cell indices actually.

Return: yield unsigned — An array of cell indices top-down sorted.

13.41.27 yield unsigned each_top_cell
Iterate the top cells.

A layout may have an arbitrary number of top cells. The usual case however is that there is one top cell.
Return: yield unsigned — An array of top cell indices.

13.41.28 end_changes
Cancel the "in changes" state (see start_changes).

13.41.29 flatten(unsigned cell_index, levels, prune)
Flatten the given cell.

This method propagates all shapes from the specified number of hierarchy levels below into the given cell. It also removes the instances of the cells from which the shapes came from, but does not remove the cells themselves if prune is set to false. If prune is set to true, these cells are removed if not used otherwise.

This method has been introduced in version 0.20.

Input:	unsigned cell_index	The cell which should be flattened.
	levels	The number of hierarchy levels to flatten (-1: all, 0: none, 1: one level etc.).
	prune	Set to true to remove orphan cells.

13.41.30 [const] const ref LayerInfo get_info(unsigned index)
Get the info structure for a specified layer.

13.41.31 boolean has_cell?(name)
Tell, if the cell with a given name exists.

Return:	true	The layout has a cell with the given name.
	false	This layout has no cell with the given name.

13.41.32 unsigned insert_layer(LayerInfo props)
Insert a new layer with the given properties.

Input:	LayerInfo props	The given properties.
Return:	unsigned	The index of the newly created layer.

13.41.33 insert_layer_at(unsigned index, LayerInfo props)
Insert a new layer with the given properties at the given index.

Input:	unsigned index	The given index.
	LayerInfo props	The given properties.

13.41.34 unsigned insert_special_layer(LayerInfo props)
Insert a new special layer with the given properties.

Special layers can be used to represent objects that should not participate in normal viewing or other related operations. Special layers are not reported as valid layers.

Input:	LayerInfo props	The given properties.
Return:	unsigned	The index of the newly created layer.

13.41.35 insert_special_layer_at(unsigned index, LayerInfo props)
Insert a new special layer with the given properties at the given index.

See insert_special_layer for a description of special layers.

Input: unsigned index The given index.
LayerInfo props The given properties.

13.41.36 [const] boolean is_special_layer?(unsigned index)
Tell, if a layer index is a special layer index.

Return: true The layer is a special layer.
false The layer is a usual layer.

13.41.37 [const] boolean is_valid_cell_index?(unsigned index)
Tell, if a cell index is valid index.

Return: true The cell index is a valid index.
false The cell index is invalid.

13.41.38 [const] boolean is_valid_layer?(unsigned index)
Tell, if a layer index is a valid index.

Return: true The layer index is a valid index.
false The layer index is invalid.

13.41.39 [const] unsigned[] layer_indices
Return a list of valid layer indices.

This method was introduced in version 0.19.

Return: unsigned[] An array with layer indices representing valid layers.

13.41.40 [const] unsignedlayers
Return the number of layers.

The number of layers reports the maximum (plus 1) layer index used so far. Not all of the layers with an index in the range of 0 to layers-1 needs to be a valid layer. These layers can be either valid, special or unused. Use is_valid_layer? and is_special_layer? to test for the first two states.

Return: unsigned[] The maximum (plus 1) layer index used so far.

13.41.41 move_layer(unsigned src, unsigned dest)
Move a layer.

Move a layer from the source to the target. The target is not cleared before, so that this method merges shapes from the source with the target layer. The source layer is empty after that operation. This method was introduced in version 0.19.

Input: unsigned src The layer index of the source layer.
unsigned dest The layer index of the destination layer.

13.41.42 unsigned[] multi_clip(unsigned, Box boxes[])

Clips the given cell by the given rectangles and produce new cells with the clips, one for each rectangle.

This method will cut rectangular regions given by the boxes from the given cell. The clips will be stored in a new cells whose indexed are returned. The clips will be performed hierarchically. The resulting cells will hold a hierarchy of child cells, which are potentially clipped versions of child cells of the original cell. This version is somewhat more efficient than doing individual clips because the clip cells may share clipped versions of child cells.

This method has been added in version 0.21.

Input:	unsigned	The cell index of the cell to clip.
	Box boxes[]	The clip boxes in database units.
Return:	unsigned[]	The indexes of the new cells.

13.41.43 unsigned[] multi_clip_into(unsigned, ref Box boxes[], Layout target)

Clips the given cell by the given rectangles and produce new cells with the clips, one for each rectangle.

This method will cut rectangular regions given by the boxes from the given cell. The clips will be stored in a new cells in the given target layout. The clips will be performed hierarchically. The resulting cells will hold a hierarchy of child cells, which are potentially clipped versions of child cells of the original cell. This version is somewhat more efficient than doing individual clips because the clip cells may share clipped versions of child cells.

Please note that it is important that the database unit of the target layout is identical to the database unit of the source layout to achieve the desired results. This method also assumes that the target layout holds the same layers than the source layout. It will copy shapes to the same layers than they have been on the original layout.

This method has been added in version 0.21.

Input:	unsigned	The cell index of the cell to clip.
	Box boxes[]	The clip boxes in database units.
	Layout target	The target layout.
Return:	unsigned[]	The indexes of the new cells.

Comment: Box and Layout exchanged.

13.41.44 [static] Layout new

Create a layout object.

13.41.45 [static] Layout new(ref Manager)

Create a layout object attached to a manager.

This method was introduced in version 0.19.

13.41.46 [const] [] properties(unsigned)

Get the properties set for a given properties ID.

Input:	unsigned	The properties ID to get the properties for.
Return:	[]	The array of variants (see properties_id).

13.41.47 unsigned properties_id(properties[])
Get the properties ID for a given properties set.

Before a set of properties can be attached to a shape, it must be converted into an ID that is unique for that set. The properties set must be given as a list of pairs of variants, each pair describing a name and a value. The name acts as the key for the property and does not need to be a string (it can be an integer or double value as well). The backward conversion can be performed with the 'properties' method.

Input: properties[] The array of pairs of variants (both elements can be integer, double or string).
Return: unsigned The unique properties ID for that set.

13.41.48 prune_cell(unsigned cell_index, levels)
Delete a cell plus sub cells not used otherwise.

This deletes a cell and also all sub cells of the cell which are not used otherwise. The number of hierarchy levels to consider can be specified as well. One level of hierarchy means that only the direct children of the cell are deleted with the cell itself. All instances of this cell are deleted as well.

This method has been introduced in version 0.20.

Input: unsigned cell_index The index of the cell to delete.
levels The number of hierarchy levels to consider (-1: all, 0: none, 1: one level etc.).

13.41.49 prune_subcells(unsigned cell_index, levels)
Delete all sub cells of the cell which are not used otherwise down to the specified level of hierarchy.

This deletes all sub cells of the cell which are not used otherwise. All instances of the deleted cells are deleted as well. It is possible to specify how many levels of hierarchy below the given root cell are considered.

This method has been introduced in version 0.20.

Input: unsigned cell_index The index of the cell to delete.
levels The number of hierarchy levels to consider (-1: all, 0: none, 1: one level etc.).

13.41.50 LayerMap read(filename, LoadLayoutOptions options)
Load the layout from the given file with options.

The format of the file is determined automatically and automatic unzipping is provided. In this version, some reader options can be specified.

This method has been added in version 0.18.

Input: filename The name of the file to load.
LoadLayoutOptions options The options object specifying further options for the reader.
Return: LayerMap A layer map that contains the mapping used by the reader including the layers that have been created.

13.41.51 LayerMap read(filename)
Load the layout from the given file.

The format of the file is determined automatically and automatic unzipping is provided. No particular options can be specified.

This method has been added in version 0.18.

Input:	filename	The name of the file to load.
Return:	LayerMap	A layer map that contains the mapping used by the reader including the layers that have been created.

13.41.52 rename_cell(unsigned, name)
Rename a cell.

Input:	unsigned	The index of the cell to rename.
	name	The new cell name.

13.41.53 set_info(unsigned, LayerInfo properties)
Set the info structure for a specified layer.

Input:	unsigned	The index of the layer.
	LayerInfo properties	The info structure for a specified layer.

13.41.54 start_changes
Signal the start of an operation bringing the layout into invalid state.

This method should be called whenever the layout is about to be brought into an invalid state. After calling this method, under_construction returns false, which tells foreign code (such as update, which might be called asynchronously, for example, because of a repaint event) not to use this layout object.

This state is cancelled by the end_changes method. The start_changes method can be called multiple times and must be cancelled the same number of times.

Using this method is only required currently if a repaint event may happen while the layout object is in an invalid state.

13.41.55 swap_layers(unsigned a, unsigned b)
Swap layers.

Swaps the shapes of both layers.

This method was introduced in version 0.19.

Input:	unsigned a	The first of the layers to swap.
	unsigned b	The second of the layers to swap.

13.41.56 [const] boolean under_construction
Tell if the layout object is under construction.

Return:	true	The layout object is either under construction if a transaction is ongoing or the layout is brought into invalid state by start_changes.
	false	The layout object is neither under construction nor brought into invalid state.

13.41.57 update
Update the internals of the layout.

This method updates the internal state of the layout. Usually this is done automatically. This method is provided to ensure this state explicitly.

13.41.58 [const] write(filename, gzip, SaveLayoutOptions options)
Write the layout to a stream file.

Input:	filename	The file to which to write the layout.
	gzip	True, if the file should be compressed.
	SaveLayoutOptions options	The option set to use for writing. See SaveLayoutOptions for details.

13.41.59 write[const] write(filename)
Write the layout to a stream file.

Input:	filename	The file to which to write the layout.

13.42 Class LayoutView (version 0.21)
The view object presenting one or more layout objects.

The visual part of the view is the tab panel in the main window. The non-visual part are the redraw thread, the layout handles, cell lists, layer view lists etc. This object controls these aspects of the view and controls the appearance of the data.

Method Overview

stop_redraw	Stop the redraw thread.
set_title	Set the title of the view.
reset_title	Reset the title to the standard title.
title	Return the view's title string.
save_layer_props	Save the layer properties.
load_layer_props	Load the layer properties.
load_layer_props	Load the layer properties with options.
load_layer_props	Load the layer properties with more options.
min_hier_levels=	Set the minimum hierarchy level at which to display geometries.
min_hier_levels?	Query the minimum hierarchy level at which to display geometries.
max_hier_levels=	Set the maximum hierarchy level up to which to display geometries.
max_hier_levels?	Query the maximum hierarchy level up to which to display geometries.
reload_layout	Reload the given cell view.
create_layout	Create a new, empty layout.
erase_cellview	Erase the cell view with the given index.
rename_cellview	Rename the cell view with the given index.
load_layout	Load a (new) file into the layout view.
load_layout	Load a (new) file into the layout view.
active_cellview	Get the active cell view (shown in hierarchy browser).
active_cellview_index	Get the index of the active cell view (shown in hierarchy browser).
set_active_cellview_index	Make the cell view with the given index the active one (shown in hierarchy browser).
get_current_cell_path	Cell path of the current cell.
set_current_cell_path	Set the path to the current cell.
cellviews	Get the number of cell views.
cellview	Get the cell view object for a given index.
zoom_fit	Fit the contents of the current view into the window.
zoom_box	Set the view port to the given box.
zoom_in	Zoom in somewhat.
zoom_out	Zoom out somewhat.
pan_up	Pan upward.
pan_down	Pan down.
pan_left	Pan to the left.
pan_right	Pan to the right.
pan_center	Pan to the given point.
box	Return the displayed box in micron space.
viewport_trans	Return the transformation that converts micron coordinates to pixels.
viewport_width	Return the view port width in pixels.
viewport_height	Return the view port height in pixels.
bookmark_view	Bookmark the current view under the given name.
add_missing_layers	Add new layers to layer list.
remove_unused_layers	Remove unused layers from layer list.
init_layer_properties	Fill the layer properties for a new layer.
cancel	Cancel all edit operations.

stop	Stop redraw thread and close any browsers.
enable_edits	Enable or disable editing.
select_cell_path	Select a cell by cell index for a certain cell view.
select_cell	Select a cell by index for a certain cell view.
descend	Descend further into the hierarchy.
ascend	Ascend upwards in the hierarchy.
is_cell_hidden	Tell, if the cell is hidden.
hide_cell	Hide the given cell for the given cell view.
show_cell	Show the given cell for the given cell view (cancel effect of hide-cell).
show_all_cells	Make all cells shown (cancel effects of hide_cell).
update_content	Update the layout view to the current state.
max_hier	Select all hierarchy levels available.
save_screenshot	Save a screen shot to the given file.
save_image	Save the layout as an image to the given file.
save_as	Save a layout to the given stream file.
set_layer_properties	Set the layer properties of the layer pointed to by the iterator.
set_layer_properties	Set the layer properties of the layer pointed to by the iterator for the given layer properties list.
expand_layer_properties	Expands the layer properties for all tabs.
expand_layer_properties	Expands the layer properties for the given tab.
replace_layer_node	Replace the layer node at the position given by "iter" with a new one.
replace_layer_node	Replace the layer node at the position given by "iter" with a new one for the given layer properties list.
insert_layer	Insert the given layer properties node into the list before the given position.
insert_layer	Insert the given layer properties node into the list before the given position for the given layer properties list.
delete_layer	Delete the layer properties node.
delete_layer	Delete the layer properties node for the given layer properties list.
begin_layers	Begin iterator for the layers.
end_layers	End iterator for the layers.
begin_layers	Begin iterator for the layers for the given layer properties list.
end_layers	End iterator for the layers for the given layer properties list.
clear_layers	Clear all layers.
clear_layers	Clear all layers for the given layer properties list.
delete_layer_list	Deletes the given properties list.
insert_layer_list	Inserts a new layer properties list at the given index.
current_layer_list	Gets the index of the currently selected layer properties tab..
set_current_layer_list	Sets the index of the currently selected layer properties tab..
rename_layer_list	Sets the title of the given layer properties tab..
remove_stipple	Remove the stipple pattern with the given index.
clear_stipples	Remove all stipple pattern.
add_stipple	Add a stipple pattern.
current_layer	Get the current layer view.
selected_layers	Get the selected layers.
add_cellview_list_observer	Add a cell view list observer.
remove_cellview_list_observer	Remove a cell view list observer.
add_cellview_observer	Add a cell view observer.
remove_cellview_observer	Remove a cell view observer.
add_file_open_observer	Add a file open observer.
remove_file_open_observer	Remove a file open observer.
add_viewport_changed_observer	Add a view port changed observer.
remove_viewport_changed_observer	Remove a view port changed observer.
add_layer_list_observer	Add a layer list observer.

remove_layer_list_observer Remove a layer list observer.
add_cell_visibility_observer Add a cell visibility observer.
remove_cell_visibility_observer Remove a cell visibility observer.
add_transient_selection_changed_observer Add a transient selection observer.
remove_transient_selection_changed_observer Remove a transient selection observer.
add_selection_changed_observer Add a selection observer.
remove_selection_changed_observer Remove a selection observer.
add_rdb-list_changed_observer Add a observer for the list of report databases.
remove_rdb_list_changed_observer Remove a observer for the list of report databases.
num_rdbs Get the number of report databases loaded into this view.
remove_rdb Remove a report database with the given index.
rdb Gets the report database with the given index.
create_rdb Creates a new report database and returns the index of the new database.
clear_config Clear the local configuration parameters.
get_config Query a local configuration parameter.
set_config Set a local configuration parameter with the given name to the given value.
transaction Begin a transaction.
commit End a transaction.
transacting Tell, if a transaction is ongoing.
clear_transactions Clear all transactions.
has_object_selection? Returns true, if geometrical objects (shapes or cell instances) are selected in this view.
each_object_selected Iterate over each selected geometrical object, yielding a ObjectInstPath object for each of them.
has_transient_object_selection? Returns true, if geometrical objects (shapes or cell instances) are selected in this view in the transient selection.
each_object_selected_transient Iterate over each geometrical objects in the transient selection, yielding a ObjectInstPath object for each of them.
clear_images Clear all images on this view.
replace_image Replace an image object with the new image.
erase_image Erase the given image.
show_image Shows or hides the given image.
insert_image Insert an image object into the given view.
each_image Iterate over all images attached to this view.
has_image_selection? Returns true, if images are selected in this view.
each_image_selected Iterate over each selected image object, yielding a Image object for each of them.
clear_annotations Clear all annotations on this view.
insert_annotation Insert an annotation object into the given view.
each_annotation Iterate over all annotations attached to this view.
has_annotation_selection? Returns true, if annotations (rulers) are selected in this view.
each_annotation_selected Iterate over each selected annotation objects, yielding a Annotation object for each of them.
destroy Explicitly destroy the object.
destroyed Tell, if the object was destroyed.

13.42.1 [const] const Refe CellView active_cellview
Get the active cell view (shown in hierarchy browser).

This is a convenience method which is equivalent to cellview(active_cellview_index()).

This method has been introduced in version 0.19.

13.42.2 [const] integer active_cellview_index
Get the index of the active cell view (shown in hierarchy browser).

13.42.3 add_cell_visibility_observer(ref ObserverBase observer)
Add a cell visibility observer.

If a cell is hidden or shown, this observer is triggered.

13.42.4 add_cellview_list_observer(ref ObserverBase observer)
Add a cellview list observer.

If a cell view is added or removed, this observer is triggered.

13.42.5 add_cellview_observer(ref ObserverBase observer)
Add a cell view observer.

If a cell view is changed (i.e. the cell is changed) this event is sent. The integer argument slot (signal_int) of the observer will be triggered as well with the index of the cell view that has changed.

13.42.6 add_file_open_observer(ref ObserverBase observer)
Add a file open observer.

If a new file is loaded, this observer is triggered.

13.42.7 add_layer_list_observer(ref ObserverBase observer)
Add a layer list observer.

If the layer list changes, the observer's signal_int and signal slot is triggered. The integers value bit 0 is set, if the properties have changed. If the arguments bit 1 is set, the hierarchy has changed.

13.42.8 add_missing_layers
Add new layers to layer list.

This method was introduced in version 0.19.

13.42.9 add_rdb-list_changed_observer(ref ObserverBase observer)
Add a observer for the list of report databases.

If a report database is added or removed, this observer is triggered.

13.42.10 add_selection_changed_observer(ref ObserverBase observer)
Add a selection observer.

If the selection is changed, this observer is triggered.

This method was added in version 0.18.

13.42.11 unsigned add_stipple(name, unsigned data[], unsigned bits)
Add a stipple pattern.

Input:	name	The name under which this pattern will appear in the stipple editor.
	unsigned data[]	An array of unsigned integers describing the bits that make up the stipple pattern. If the array has less than 32 entries, the pattern will be repeated vertically. The number of bits used can be less than 32 bit which can be specified by the "bits" parameter. Logically, the pattern will be put at the end of the list.
	unsigned bits	The number of bits used.
Return:	unsigned	The index of the newly created stipple pattern, which can be used as the dither pattern index of LayerProperties.

13.42.12 add_transient_selection_changed_observer(ref ObserverBase observer)
Add a transient selection observer.

If the transient selection is changed, this observer is triggered.

This method was added in version 0.18.

13.42.13 add_viewport_changed_observer(ref ObserverBase observer)
Add a view port changed observer.

If the view port (the rectangle that is shown) changes, this observer is triggered.

13.42.14 InstElement ascend(index)
Ascend upwards in the hierarchy.

Removes one element from the specific path of the cell view with the given index.

Input:	index	The cell view with the given index.
Return:	InstElement	The removed element.

13.42.15 [const] LayerPropertiesIterator begin_layers
Begin iterator for the layers.

This iterator delivers the layers of this view, either in a recursive or non-recursive fashion, depending which iterator increment methods are used. The iterator delivered by end_layers is the past-the-end iterator. It can be compared against a current iterator to check, if there are no further elements.

13.42.16 [const] LayerPropertiesIterator begin_layers(unsigned index)
Begin iterator for the layers.

This iterator delivers the layers of this view, either in a recursive or non-recursive fashion, depending which iterator increment methods are used. The iterator delivered by end_layers is the past-the-end iterator. It can be compared against a current iterator to check, if there are no further elements. This version addresses a specific list in a multi-tab layer properties arrangement with the "index" parameter.

This method has been introduced in version 0.21.

13.42.17 bookmark_view(name)
Bookmark the current view under the given name.

Input: name The name under which to bookmark the current state.

13.42.18 [const] DBox box
Return the displayed box in micron space.

Return: DBox The displayed box in micron space.

13.42.19 cancel
Cancel all edit operations.

13.42.20 [const] const ref CellView cellview(unsigned index)
Get the cell view object for a given index.

Input: unsigned index The cell view index for which to get the object for.

13.42.21 [const] unsigned cellviews
Get the number of cell views.

Return: unsigned The number of cell views.

13.42.22 clear_annotations
Clear all annotations on this view.

13.42.23 clear_config
Clear the local configuration parameters.

See set_config for a description of the local configuration parameters.

13.42.24 clear_images
Clear all images on this view.

13.42.25 clear_layers
Clear all layers.

13.42.26 clear_layers(unsigned index)
Clear all layers for the given layer properties list.

This method has been introduced in version 0.21.

Input: unsigned index A specific list in a multi-tab layer properties arrangement.

13.42.27 clear_stipples
Remove all stipple pattern.

All stipple pattern except the fixed ones are removed. If any of the custom stipple pattern is still used by the layers displayed, the results will not be predictable.

13.42.28 clear_transactions
Clear all transactions.

Discard all actions in the undo buffer. After clearing that buffer, no undo is available. It is important to clear the buffer when making database modifications outside transactions, i.e after that modifications have been done. If failing to do so, "undo" operations are likely to produce invalid results.

This method was introduced in version 0.16.

13.42.29 commit
End a transaction.

See transaction for a detailed description of transactions.

This method was introduced in version 0.16.

13.42.30 unsigned index create_layout(add_cellview)
Create a new, empty layout.

Input:	true	Create a new cell view.
	false	Clear all cell views before.
Return:	unsigned index	The index of the cellview created.

13.42.31 unsigned create_rdb(name)
Creates a new report database and returns the index of the new database.

This method returns an index of the new report database. Use rdb to get the actual object. If a report database with the given name already exists, a unique name will be created. The name will be replaced by the file name when a file is loaded into the report database.

Input:	name	The name of the new report database.
Return:	unsigned	The index of the new database.

13.42.32 [const] LayerPropertiesIterator current_layer
Get the current layer view.

Return:	LayerPropertiesIterator	The LayerPropertiesIterator pointing to the current layer view (the one that has the focus). If no layer view is active currently, a null iterator is returned.

13.42.33 [const] unsigned current_layer_list
Gets the index of the currently selected layer properties tab.

This method has been introduced in version 0.21.

13.42.34 delete_layer(refLayerPropertiesIterator iter)
Delete the layer properties node.

This method deletes the object that the iterator points to and invalidates the iterator since the object that the iterator points to is no longer valid.

13.42.35 delete_layer(unsigned index, refLayerPropertiesIterator iter)
Delete the layer properties node.

This method deletes the object that the iterator points to and invalidates the iterator since the object that the iterator points to is no longer valid. This version addresses a specific list in a multi-tab layer properties arrangement with the index parameter.

This method has been introduced in version 0.21.

13.42.36 delete_layer_list(unsigned index)
Deletes the given properties list.

At least one layer properties list must remain. This method may change the current properties list.

This method has been introduced in version 0.21.

13.42.37 descend(InstElement path[], index)
Deletes the given properties list.

At least one layer properties list must remain. This method may change the current properties list.

This method has been introduced in version 0.21.

13.42.38 destroy
Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.42.39 [const] boolean destroyed
Tell, if the object was destroyed.

Return:	true	The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
	false	The object is still alive.

13.42.40 yield const ref Annotation each_annotation
Iterate over all annotations attached to this view.

13.42.41 [const] yield const ref Annotation each_annotation_selected
Iterate over each selected annotation objects, yielding a Annotation object for each of them.

This method was introduced in version 0.19.

13.42.42 yield const ref Image each_image

Iterate over all images attached to this view.

13.42.43 [const] yield const ref Image each_image_selected

Iterate over each selected image object, yielding a Image object for each of them.

This method was introduced in version 0.19.

13.42.44 [const] yield const ref ObjectInstPath each_object_selected

Iterate over each selected geometrical object, yielding a ObjectInstPath object for each of them.

13.42.45 [const] yield const ref ObjectInstPath each_object_selected_transient

Iterate over each geometrical objects in the transient selection, yielding a ObjectInstPath object for each of them.

This method was introduced in version 0.18.

13.42.46 enable_edits(enable)

Enable or disable editing.

Input:	true	Edit mode enabled.
	false	View mode enabled.

13.42.47 [const] LayerPropertiesIterator end_layers(unsigned index)

End iterator for the layers.

See begin_layers for a description about this iterator This version addresses a specific list in a multi-tab layer properties arrangement with the index parameter.

This method has been introduced in version 0.21.

13.42.48 [const] LayerPropertiesIterator end_layers

End iterator for the layers.

See begin_layers for a description about this iterator.

13.42.49 erase_cellview(unsigned index)

Erase the cell view with the given index.

This closes the given cell view and unloads the layout associated with it, unless referred to by another cell view.

13.42.50 erase_image(unsigned id)

Erase the given image.

Erases the image with the given Id. The Id can be obtained with if id method of the image object.

This method has been introduced in version 0.20.

Input:	unsigned id	The id of the object to erase.

13.42.51 expand_layer_properties
Expands the layer properties for all tabs.

This method will expand all wild card specifications in the layer properties by iterating over the specified objects (i.e. layers, cell views) and by replacing default colors and stipples by the ones specified with the palettes.

This method was introduced in version 0.21.

13.42.52 expand_layer_properties(unsigned)
Expands the layer properties for the given tab.

This method will expand all wild card specifications in the layer properties by iterating over the specified objects (i.e. layers, cell views) and by replacing default colors and stipples by the ones specified with the palettes.

This method was introduced in version 0.21.

13.42.53 [const] string get_config(name)
Query a local configuration parameter.

See set_config for a description of the local configuration parameters.

Input:	name	The name of the configuration parameter whose value shall be obtained (a string).
Return:	string	The value of the parameter.

13.42.54 [const] unsigned[] get_current_cell_path(index)
Cell path of the current cell.

The current cell is the one highlighted in the browser with the focus rectangle. The current path is returned for the cell view given by index. The cell path is a list of cell indices from the top cell to the current cell.

Input:	index	The cell view index.
Return:	unsigned[]	The current path for the cell view given by index.

13.42.55 [const] boolean has_annotation_selection?
Returns true, if annotations (rulers) are selected in this view.

This method was introduced in version 0.19.

Return:	true	Annotations (rulers) are selected in this view.
	false	No annotations (rulers) are selected in this view.

13.42.56 [const] boolean has_image_selection?
Returns true, if images are selected in this view.

This method was introduced in version 0.19.

Return:	true	There are selected images in this view.
	false	No selected images in this view.

13.42.57 [const] boolean has_object_selection?

Returns true, if geometrical objects (shapes or cell instances) are selected in this view.

Return:	true	There are selected geometrical objects in this view.
	false	No selected geometrical objects in this view.

13.42.58 [const] boolean has_transient_object_selection?

Returns true, if geometrical objects (shapes or cell instances) are selected in this view in the transient selection.

The transient selection represents the objects selected when the mouse hovers over the layout windows. This selection is not used for operations but rather to indicate which object would be selected if the mouse is clicked.

This method was introduced in version 0.18.

Return:	true	There are transient selected geometrical objects in this view.
	false	No transient selected geometrical objects in this view.

13.42.59 hide_cell(unsigned cell_index, cellview_index)

Hide the given cell for the given cell view.

Input:	unsigned cell_index	The cell index.
	cellview_index	The cell view index.

13.42.60 [const] init_layer_properties(ref LayerProperties props)

Fill the layer properties for a new layer.

This method initializes a layer properties object's color and stipples according to the defaults for the given layer source specification. The layer's source must be set already on the layer properties object.

This method was introduced in version 0.19.

Input:	props	The layer properties object to initialize.

13.42.61 insert_annotation(Annotation obj)

Insert an annotation object into the given view.

Input:	obj	The annotation object to insert into this view.

13.42.62 insert_image(Image obj)

Insert an image object into the given view.

Input:	obj	The image object to insert into this view.

13.42.63 [const] ref LayerPropertiesNode insert_layer(LayerPropertiesIterator iter, LayerPropertiesNode node)

Insert the given layer properties node into the list before the given position.

Input:	node	The new properties node to insert.
	iter	The position to insert before.
Return:	[const] ref	A constant reference to the element created.

13.42.64 [const] ref LayerPropertiesNode insert_layer(unsigned index, LayerPropertiesIterator iter, LayerPropertiesNode node)

Insert the given layer properties node into the list before the given position.

Input:	unsigned index	The index of a specific list in a multi-tab layer properties arrangement.
	node	The new properties node to insert.
	iter	The position to insert before.
Return:	[const] ref	A constant reference to the element created.

13.42.65 insert_layer_list(unsigned index)

Inserts a new layer properties list at the given index.

This method inserts a new tab at the given position. The current layer properties list will be changed to the new list.

This method has been introduced in version 0.21.

Input:	unsigned index	The given position.

13.42.66 [const] boolean is_cell_hidden(unsigned cell_index, unsigned cellview_index)

Tell, if the cell is hidden.

Input:	unsigned cell_index	The cell index.
	cellview_index	The cell view index.
Return:	true	The cell with given cell index is hidden in cell view with given cell view index.
	false	???.

13.42.67 load_layer_props(filename)

Load the layer properties.

Input:	filename	Load the layer properties from this file.

13.42.68 load_layer_props(filename, boolean)

Load the layer properties with options.

This variant has been added on version 0.21.

Input:	filename	Load the layer properties from this file.
	true	Use defaults for all other layers.
	false	Don't use defaults for all other layers.

13.42.69 load_layer_props(filename, cellview_index, boolean)
Load the layer properties with options.

This variant has been added on version 0.21.

Input:	filename	Load the layer properties from this file.
	cellview_index	Load the layer properties for this specific cell view. All present definitions for this layout will be removed before the properties file is loaded. Or
	-1	load the layer properties for each layout individually.
	true	Use defaults for all other layers.
	false	Don't use defaults for all other layers.

13.42.70 unsigned load_layout(filename, LoadLayoutOptions options, boolean)
Load a (new) file into the layout view.

This method has been introduced in version 0.18.

Input:	filename	Load the layout from this file.
	options	Use this options.
	true	Create a new cell view.
	false	Clear all cell views before load.
Return:	unsigned	The index of the cell view loaded.

13.42.71 unsigned load_layout(filename, boolean)
Load a (new) file into the layout view.

Input:	filename	Load the layout from this file.
	true	Create a new cell view.
	false	Clear all cell views before load.
Return:	unsigned	The index of the cell view loaded.

13.42.72 max_hier
Select all hierarchy levels available.

Show the layout in full depth down to the deepest level of hierarchy. This method may cause a redraw.

13.42.73 max_hier_levels=(level)
Set the maximum hierarchy level up to which to display geometries.

This methods allows to set the maximum hierarchy below which to display geometries.This method may cause a redraw if required.

Input:	level	The maximum level below which to display something.

13.42.74 [const] level max_hier_levels?
Query the maximum hierarchy level up to which to display geometries.

Return:	level	The maximum level up to which to display geometries.

13.42.75 min_hier_levels=(level)
Set the minimum hierarchy level at which to display geometries.

This methods allows to set the minimum hierarchy above which to display geometries.This method may cause a redraw if required.

Input: level The minimum level above which to display something.

13.42.76 [const] level min_hier_levels?
Query the minimum hierarchy level at which to display geometries.

Return: level The minimum level at which to display geometries.

13.42.77 [const] unsigned num_rdbs
Get the number of report databases loaded into this view.

Return: unsigned The number of ReportDatabase objects present in this view.

13.42.78 pan_center(DPoint point)
Pan to the given point.

Input: point The window is positioned such this point becomes the new center.

13.42.79 pan_down
Pan downwards.

13.42.80 pan_left
Pan to the left.

13.42.81 pan_right
Pan to the right.

13.42.82 pan_up
Pan upwards.

13.42.83 ref ReportDatabase rdb(index)
Gets the report database with the given index.

Return: ref A reference to the report database object, or
Return: ReportDatabase The report database object, or
nil if the index is invalid.
Comment: Returns the reference to or the report database itself?

13.42.84 reload_layout(unsigned index)
Reload the given cellview.

Input: unsigned index The index of the cell view to reload.

13.42.85 remove_cell_visibility_observer(ref ObserverBase observer)
Remove a cell visibility observer.

13.42.86 remove_cellview_list_observer(ref ObserverBase observer)
Remove a cell view list observer.

13.42.87 remove_cellview_observer(ref ObserverBase observer)
Remove a cell view observer.

13.42.88 remove_file_open_observer(ref ObserverBase observer)
Remove a file open observer.

13.42.89 remove_layer_list_observer(ref ObserverBase observer)
Remove a layer list observer.

13.42.90 remove_rdb(unsigned index)
Remove a report database with the given index.

Input: unsigned index The index of the report database to remove from this view.

13.42.91 remove_rdb_list_changed_observer(ref ObserverBase observer)
Remove a observer for the list of report databases.

13.42.92 remove_selection_changed_observer(ref ObserverBase observer)
Remove a selection observer.

This method was added in version 0.18.

13.42.93 remove_stipple(unsigned index)
Remove the stipple pattern with the given index.

The pattern with an index less than 16 cannot be removed. If a stipple pattern is removed that is still used, the results are not predictable.

13.42.94 remove_transient_selection_changed_observer(ref ObserverBase observer)
Remove a transient selection observer.

This method was added in version 0.18.

13.42.95 remove_unused_layers
Remove unused layers from layer list.

This method was added in version 0.19.

13.42.96 remove_viewport_changed_observer(ref ObserverBase observer)
Remove a viewport changed observer.

13.42.97 rename_cellview(name, index)
Rename the cell view with the given index.

If the name is not unique, a unique name will be constructed from the name given. The name may be different from the file name but is associated with the layout object. If a layout is shared between multiple cell views (which may happen due to a clone of the layout view for example), both cell views are renamed.

Input:	name	The given name.
	index	The index of the cell view to rename.

13.42.98 rename_layer_list(unsigned index, name)
Sets the title of the given layer properties tab.

This method has been introduced in version 0.21.

Input:	unsigned index	The given layer properties tab.
	name	The title to set.

13.42.99 replace_image(unsigned id, ref Image new_obj
Replace an image object with the new image.

Replaces the image with the given Id with the new object. The Id can be obtained with the id method of the image object.

This method has been introduced in version 0.20.

Input:	unsigned id	The id of the object to replace.
	new_obj	The new object to replace the old one.

13.42.100 replace_layer_node(LayerPropertiesIterator iter, LayerPropertiesNode node)
Replace the layer node at the position given by iter with a new one.

This version addresses a specific list in a multi-tab layer properties arrangement with the index parameter.

This method has been introduced in version 0.21.

Input:	node	The new properties node to insert.
	iter	The position to insert before.

13.42.101 replace_layer_node(unsigned index, LayerPropertiesIterator iter, LayerPropertiesNode node)
Replace the layer node at the position given by iter with a new one.

This version addresses a specific list in a multi-tab layer properties arrangement with the index parameter.

This method has been introduced in version 0.21.

Input:	unsigned index	The index of a specific list in a multi-tab layer properties arrangement.
	node	The new properties node to insert.
	iter	The position to insert before.

13.42.102 reset_title
Reset the title to the standard title.

See set_title and title for a description about how titles are handled.

13.42.103 save_as(unsigned index, filename, boolean, SaveLayoutOptions options)
Save a layout to the given stream file.

The given layout (with the given index) is written to the stream file with the given options. options is a SaveLayoutOptions object that specifies which format to write and further options such as scaling factor etc. Calling this method is equivalent to calling "write" on the respective layout object.

Input:	unsigned index	The cell view index of the layout to save.
	filename	The file to write.
	true	Compress the file (gzip).
	false	No file compress.
	options	Writer options.

13.42.104 save_image(filename, unsigned width, unsigned height)
Save the layout as an image to the given file.

The image contains the current scene (layout, annotations etc.). The image is written as a PNG file to the given file. The image is drawn synchronously with the given width and height. Drawing may take some time.

Input:	filename	The file to which to write the image to.
	unsigned width	The width of the image to render in pixel.
	unsigned height	The height of the image to render in pixel.

13.42.105 save_layer_props(filename)
Save the layer properties.

Input:	filename	The file to which to write the layer properties.

13.42.106 save_screenshot(filename)
Save a screenshot to the given file.

The screen shot is written as a PNG file to the given file. This requires the drawing to be complete. Ideally, synchronous mode is switched on for the application to guarantee this condition. The image will have the size of the view port showing the current layout.

Input:	filename	The file to which to write the screen shot to.

13.42.107 select_cell(unsigned cell_index, unsigned cellview_index)
Select a cell by index for a certain cell view.

Select the current (top) cell by specifying a path (a list of cell indices from top to the actual cell) and the cell view index for which this cell should become the currently shown one. This method selects the cell to be drawn. In contrast, the set_current_cell_path method selects the cell that is highlighted in the cell tree (but not necessarily drawn).

Input: unsigned cell_index — The cell index.
cellview_index — The cell view index.

13.42.108 select_cell_path(unsigned cell_index[], unsigned cellview_index)
Select a cell by cell index for a certain cell view.

Select the current (top) cell by specifying a cell index and the cell view index for which this cell should become the currently shown one. The path to the cell is constructed by selecting one that leads to a top cell. This method selects the cell to be drawn. In contrast, the set_current_cell_path method selects the cell that is highlighted in the cell tree (but not necessarily drawn).

Input: unsigned cell_index — The cell index.
cellview_index — The cell view index.

13.42.109 [const] LayerPropertiesIterator[ ] selected_layers
Get the selected layers.

Return: LayerPropertiesIterator — An array of LayerPropertiesIterator objects pointing to the currently selected layers. If no layer view is selected currently, an empty array is returned.

13.42.110 set_active_cellview_index(index)
Make the cell view with the given index the active one (shown in hierarchy browser).

See active_cellview_index.

Input: index — The cell view index to become active.

13.42.111 set_config(name, value)
Set a local configuration parameter with the given name to the given value.

This method sets a local configuration parameter with the given name to the given value. Values can only be strings. Numerical values have to be converted into strings first. Local configuration parameters override global configurations for this specific view. This allows, for example, to override global settings of background colors. Any local settings are not written to the configuration file.

Input: name — The name of the configuration parameter to set.
value — The value to which to set the configuration parameter.

13.42.112 set_current_cell_path(cellview_index, unsigned[])
Set the path to the current cell.

The current cell is the one highlighted in the browser with the focus rectangle. The cell given by the path is highlighted and scrolled into view. To select the cell to be drawn, use the select_cell or select_cell_path method.

Input: cellview_index — The cellview index for which to set the current path for (usally this will be the active cellview index).
path — The path to the current cell.

Comment: path?

13.42.113 set_current_layer_list(unsigned index)
Sets the index of the currently selected layer properties tab.

This method has been introduced in version 0.21.

Input: unsigned index — The index of the layer properties tab to become current.

13.42.114 set_layer_properties(LayerPropertiesIterator iter, LayerProperties props)
Set the layer properties of the layer pointed to by the iterator.

Input: iter — Replace the layer properties of this element.
props — The new properties. The hierarchy will not change but just the properties of the given node.

13.42.115 set_layer_properties(unsigned index, LayerPropertiesIterator iter, LayerProperties props)
Set the layer properties of the layer pointed to by the iterator.

This method has been introduced in version 0.21.

Input: unsigned index — A specific list in a multi-tab layer properties arrangement.
iter — Replace the layer properties of this element.
props — The new properties. The hierarchy will not change but just the properties of the given node.

13.42.116 set_title(title)
Set the title of the view.

Override the standard title of the view indicating the file names loaded by the specified title string. The title string can be reset with reset_title to the standard title again.

Input: title — The new title string to use.

13.42.117 show_all_cells
Make all cells shown (cancel effects of hide_cell).

13.42.118 show_cell(unsigned cell_index, cellview_index)
Show the given cell for the given cellview (cancel effect of hide_cell).

Input: unsigned cell_index — The index of the cell to show.
cellview_index — The index of the cell view.

13.42.119 show_image(unsigned id, visible)
Shows or hides the given image.

Sets the visibility of the image with the given Id. The Id can be obtained with the id method of the image object.

This method has been introduced in version 0.20.

Input: unsigned id — The ID of the image.
Return: true — Set to visible.
false — Set to invisible.

13.42.120 stop
Stop redraw thread and close any browsers.

This method usually does not need to be called explicitly. The redraw thread is stopped automatically.

13.42.121 stop_redraw
Stop the redraw thread.

It is very important to stop the redraw thread before applying changes to the layout or the cell views and the LayoutView configuration. This is usually done automatically. For rare cases, where this is not the case, this method is provided.

13.42.122 [const] string title
Return the view's title string.

The title string is either a string composed of the file names loaded (in some "readable" manner) or a customized title string set by set_title.

Return: string The title string.

13.42.123 boolean transacting
Tell if a transaction is ongoing.

See transaction for a detailed description of transactions.

This method was introduced in version 0.16.

Return: true Transaction is ongoing.
false Transaction is finished.

13.42.124 transaction(string)
Begin a transaction.

A transaction brackets a sequence of database modifications that appear as a single undo action. Only modifications that are wrapped inside a transaction...commit call pair can be undone. Each transaction must be terminated with a commit method call, even if some error occurred. It is advisable therefore to catch errors and issue a commit call in this case.

This method was introduced in version 0.16.

Input: string A text that appears in the undo description.

13.42.125 update_content
Update the layout view to the current state.

This method triggers an update of the hierarchy tree and layer view tree. Usually, this method does not need to be called. The widgets are updated automatically in most cases.

Currently, this method **must** be called however, after the layer view tree has been changed by the insert_layer, replace_layer_node or delete_layer methods.

13.42.126 [const] integer viewport_height
Return the view port height in pixels.

This method was introduced in version 0.18.

Return: integer The view port height in pixels.

13.42.127 [const] DCplxTrans viewport_trans
Return the transformation that converts micron coordinates to pixels.

Hint: The transformation returned will convert any point in micron coordinate space into a pixel coordinate. Contrary to usual convention, the y pixel coordinate is given in a mathematically oriented space - which means the bottom coordinate is 0.

This method was introduced in version 0.18.

13.42.128 [const] integer viewport_width
Return the view port height in pixels.

This method was introduced in version 0.18.

Return: integer The view port width in pixels.

13.42.129 zoom_box(DBox box)
Set the viewport to the given box.

Input: box The box to which to set the view in micron coordinates.

13.42.130 zoom_fit
Fit the contents of the current view into the window.

13.42.131 zoom_in
Zoom in somewhat.

13.42.132 zoom_out
Zoom out somewhat.

13.43 Class LoadLayoutOptions (version 0.21) Layout reader options.

This object describes various layer reader options used for loading layouts.

This class has been introduced in version 0.18.

Method Overview

set_layer_map	Set a layer map.
select_all_layers	Select all layers.
layer_map	Access to the layer map member.
is_creating_other_layers?	Tell whether other layers should be created.
create_other_layers=	Specifies whether other layers should be created.
is_text_enabled?	Tell whether text objects should be read.
text_enabled=	Specifies whether text objects should be read.
is_properties_enabled?	Tell whether properties should be read.
properties_enabled=	Specifies whether properties should be read.
assign	Assign the contents of another object to self.
dup	Creates a copy of self.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.43.1 assign(LoadLayoutOptions other) Assign the contents of another object to self.

This method assigns the contents of another object to self. This is a deep copy that does not only copy the reference but the actual content.

13.43.2 create_other_layers=(boolean) Specifies whether other layers should be created.

Input:	true	Other layers should be created.
	false	No other layers should be created.

13.43.3 destroy Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.43.4 [const] boolean destroyed Tell, if the object was destroyed.

Return:	true	The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
	false	The object is still alive.

13.43.5 [const] LoadLayoutOptions dup Creates a copy of self.

Return: LoadLayoutOptions The copy of self.

13.43.6 [const] boolean is_creating_other_layers?

Tell whether other layers should be created.

Input:	true	Other layers should be created.
	false	No other layers should be created.

13.43.7 is_properties_enabled?

Tell whether properties should be read.

Input:	true	Properties should be read.
	false	No properties should be read.

13.43.8 is_text_enabled?

Tell whether text objects should be read.

Input:	true	Text objects should be read.
	false	No text objects should be read.

13.43.9 ref LayerMap layer_map

Access to the layer map member.

Return:	ref	Reference to the layer map.

13.43.10 properties_enabled=

Specifies whether properties should be read..

Input:	true	Properties should be read.
	false	No properties should be read.

13.43.11 select_all_layers

Select all layers.

This disables any layer map and enables reading of all layers while new layers will be created when required.

13.43.12 set_layer_map(LayerMap map, boolean)

Set a layer map.

Input:	map	The layer map to be read.
	true	Other layers should be created and automatically assign layers to them.
	false	Only layers in the mapping table should be read.

13.43.13 text_enabled=(boolean)

Specifies whether text objects should be read.

Input:	true	Text objects should be read.
	false	No text objects should be read.

13.44 Class MainWindow (version 0.21)
The main application window and central controller object.

This object first is the main window but also the main controller. The main controller is the port by which access can be gained to all the data objects, view and other aspects of the program.

Method Overview

menu	Return a reference to the abstract menu.
message	Display a message in the status bar.
resize	Re-size the window.
grid_micron	Get the global grid in micron.
create_layout	Create a new, empty layout.
load_layout	Load a new layout.
clone_current_view	Clone the current view and make it current.
save_session	Save the session to the given file.
restore_session	Restore a session from the given file.
enable_edits	Enable or disable edits.
synchronous=	Put the main window into synchronous mode.
close_all	Closes all views.
close_current_view	Close the current view.
cancel	Cancel current editing operations.
redraw	Redraw the current view.
exit	Schedule an exit for the application.
select_view	Select the view with the given index.
current_view_index	Return the current view's index.
current_view	Return a reference to the current view's object.
views	Return the number of views.
view	Return a reference to a view object by index.
reader_options	Access to the current reader options.
add_current_view_observer	Add an observer for the "current view changed" event.
remove_current_view_observer	Remove an observer for the change of the "current view changed" event.
add_new_view_observer	Add an observer for a "new view" event.
remove_new_view_observer	Remove an observer for a "new view" event.
cm_...	Various command action bound to a menu.
cm_undo	"cm_undo" action.
cm_redo	"cm_redo" action.
cm_delete	"cm_delete" action.
cm_show-properties	"cm_show_properties" action.
cm_copy	"cm_copy" action.
cm_paste	"cm_paste" action.
cm_cut	"cm_cut" action.
cm_zoom_fit_sel	"cm_zoom_fit_sel" action.
cm_zoom_fit	"cm_zoom_fit" action.
cm_zoom_in	"cm_zoom_in" action.
cm_zoom_out	"cm_zoom_out" action.
cm_pan_up	"cm_pan_up" action.
cm_pan_down	"cm_pan_down" action.
cm_pan_left	"cm_pan_left" action.
cm_pan_right	"cm_pan_right" action.
cm_save_session	"cm_save_session" action.
cm_restore_session	"cm_restore_session" action.

cm_setup	“cm_setup” action.
cm_save_as	“cm_save_as” action.
cm_save	“cm_save” action.
cm_reload	“cm_reload” action.
cm_close	“cm_close” action.
cm_clone	“cm_clone” action.
cm_layout_props	“cm_layout_props” action.
cm_inc_max_hier	“cm_inc_max_hier” action.
cm_dec-max-hier	“cm_dec_max_hier” action.
cm_max_hier	“cm_max_hier” action.
cm_max_hier_0	“cm_max_hier_0” action.
cm_max_hier_1	“cm_max_hier_1” action.
cm_last_display_state	“cm_last_display_state” action.
cm_next_display_state	“cm_next_display_state” action.
cm_cancel	“cm_cancel” action.
cm_redraw	“cm_redraw” action.
cm_screenshot	“cm_screenshot” action.
cm_save_layer_props	“cm_save_layer_props” action.
cm_load_layer_prop	“cm_load_layer_props” action.
cm_save_bookmarks	“cm_save_bookmarks” action.
cm_load_bookmark	“cm_load_bookmarks” action.
cm_select_cell	“cm_select_cell” action.
cm_select_current_cell	“cm_select_current_cell” action.
cm_exit	“exit” action.
cm_view_log	“cm_view_log” action.
cm_bookmark_view	“cm_bookmark_view” action.
cm_manage_bookmarks	“cm_manage_bookmarks” action.
cm_goto_position	“cm_goto_position” action.
cm_help_about	“cm_help_about” action.
cm_console	“cm_console” action.
cm_open_too	“cm_open_too” action.
cm_open_new_view	“cm_open_new_view” action.
cm_open	“cm_open” action.
cm_pull_in	“cm_pull_in” action.
cm_reader_options	“cm_reader_options” action.
cm_new_layout	“cm_new_layout” action.
cm_new_panel	“cm_new_panel” action.
cm_adjust_origin	“cm_adjust_origin” action.
cm_new_cell	“cm_new_cell” action.
cm_new_layer	“cm_new_layer” action.
cm_clear_layer	“cm_clear_layer” action.
cm_delete_layer	“cm_delete_layer” action.
cm_edit_layer	“cm_edit_layer” action.
cm_edit_boolean	“cm_edit_boolean” action.
cm_edit_size	“cm_edit_size” action.
cm_edit_merge	“cm_edit_merge” action.
cm_sel_flip_x	“cm_sel_flip_x” action.
cm_sel_flip_y	“cm_sel_flip_y” action.
cm_sel_rot_cw	“cm_sel_rot_cw” action.
cm_sel_rot_ccw	“cm_sel_rot_ccw” action.
cm_sel_free_rot	“cm_sel_free_rot” action.
cm_sel_scale	“cm_sel_scale” action.
cm_sel_move	“cm_sel_move” action.

cm_lv_new_tab	"cm_lv_new_tab" action.
cm_lv_remove_tab	"cm_lv_remove_tab" action.
cm_lv_rename_tab	"cm_lv_rename_tab" action.
cm_lv_hide	"cm_lv_hide" action.
cm_lv_hide_all	"cm_lv_hide_all" action.
cm_lv_show	"cm_lv_show" action.
cm_lv_show_all	"cm_lv_show_all" action.
cm_lv_show_only	"cm_lv_show_only" action.
cm_lv_rename	"cm_lv_rename" action.
cm_lv_select_all	"cm_lv_select_all" action.
cm_lv_delete	"cm_lv_delete" action.
cm_lv_insert	"cm_lv_insert" action.
cm_lv_group	"cm_lv_group" action.
cm_lv_ungroup	"cm_lv_ungroup" action.
cm_lv_source	"cm_lv_source" action.
cm_lv_sort_by_name	"cm_lv_sort_by_name" action.
cm_lv_sort_by_ild	"cm_lv_sort_by_ild" action.
cm_lv_sort_by_idl	"cm_lv_sort_by_idl" action.
cm_lv_sort_by_ldi	"cm_lv_sort_by_ldi" action.
cm_lv_sort_by_dli	"cm_lv_sort_by_dli" action.
cm_lv_regroup_by_index	"cm_lv_regroup_by_index" action.
cm_lv_regroup_by_datatype	"cm_lv_regroup_by_datatype" action.
cm_lv_regroup_by_layer	"cm_lv_regroup_by_layer" action.
cm_lv_regroup_flatten	"cm_lv_regroup_flatten" action.
cm_lv_expand_all	"cm_lv_expand_all" action.
cm_lv_add_missing	"cm_lv_add_missing" action.
cm_lv_remove_unused	"cm_lv_remove_unused" action.
cm_cell_delete	"cm_cell_delete" action.
cm_cell_rename	"cm_cell_rename" action.
cm_cell_copy	"cm_cell_copy" action.
cm_cell_cut	"cm_cell_cut" action.
cm_cell_paste	"cm_cell_paste" action.
cm_cell_select	"cm_cell_select" action.
cm_open_current_cell	"cm_open_current_cell" action.
cm_save_current_cell_as	"cm_save_current_cell_as" action.
cm_cell_hide	"cm_cell_hide" action.
cm_cell_flatten	"cm_cell_flatten" action.
cm_cell_show	"cm_cell_show" action.
cm_cell_show_all	"cm_cell_show_all" action.
cm_navigator_close	"cm_navigator_close" action.
cm_navigator_freeze	"cm_navigator_freeze" action.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.44.1 add_current_view_observer(ref ObserverBase observer)
Add an observer for the "current view changed" event.

If the current view changes, this observer is triggered. The integer slot of the observer will receive the number of the view active before. The current view's reference is already updated when this event is issued.

13.44.2 add_new_view_observer(ref ObserverBase observer)
Add an observer for a "new view" event.

If a new view is created, this observer will receive a signal. The integer slot of this observer will receive the index of the newly created view.

13.44.3 cancel
Cancel current editing operations.

This method call cancels all current editing operations and restores normal mouse mode.

13.44.4 clone_current_view
Clone the current view and make it current.

13.44.5 close_all
Closes all views.

This method unconditionally closes all views. No dialog will be opened if unsaved edits exist.

This method was added in version 0.18.

13.44.6 close_current_view
Close the current view.

This method does not open a dialog to query which cell view to close if multiple cells are opened in the view but rather closes all cells.

13.44.7 cm_...
Various command action bound to a menu.

13.44.7.1 cm_adjust_origin – "cm_adjust_origin" action (bound to a menu).

This method has been added in version 0.18.

13.44.7.2 cm_bookmark_view – "cm_bookmark_view" action (bound to a menu).

13.44.7.3 cm_cancel – "cm_cancel" action (bound to a menu).

13.44.7.4 cm_cell_copy – "cm_cell_copy" action (bound to a menu).

This method has been added in version 0.20.

13.44.7.5 cm_cell_cut – "cm_cell_cut" action (bound to a menu).

This method has been added in version 0.20.

13.44.7.6 cm_cell_delete – "cm_cell_delete" action (bound to a menu).

This method has been added in version 0.18.

13.44.7.7 cm_cell_flatten – "cm_cell_flatten" action (bound to a menu).

13.44.7.8 cm_cell_hide – "cm_cell_hide" action (bound to a menu).

13.44.7.9 cm_cell_paste – "cm_cell_paste" action (bound to a menu).

This method has been added in version 0.20.

13.44.7.10 cm_cell_rename – "cm_cell_rename" action (bound to a menu).

This method has been added in version 0.18.

13.44.7.11 cm_cell_select – "cm_cell_select" action (bound to a menu).

13.44.7.12 cm_cell_show – "cm_cell_show" action (bound to a menu).

13.44.7.13 cm_cell_show_all – "cm_cell_show_all" action (bound to a menu).

13.44.7.14 cm_clear_layer – "cm_clear_layer" action (bound to a menu).

This method has been added in version 0.18.

13.44.7.15 cm_clone – "cm_clone" action (bound to a menu).

13.44.7.16 cm_close – "cm_close" action (bound to a menu).

13.44.7.17 cm_console – "cm_console" action (bound to a menu).

This method has been added in version 0.18.

13.44.7.18 cm_copy – "cm_copy" action (bound to a menu).

13.44.7.19 cm_cut – "cm_cut" action (bound to a menu).

13.44.7.20 cm_dec-max-hier – "cm_dec_max_hier" action (bound to a menu).

13.44.7.21 cm_delete – "cm_delete" action (bound to a menu).

13.44.7.22 cm_delete_layer – "cm_delete_layer" action (bound to a menu).

This method has been added in version 0.18.

13.44.7.23 cm_edit_boolean – "cm_edit_boolean" action (bound to a menu).

This method has been added in version 0.18.

13.44.7.24 cm_edit_layer – "cm_edit_layer" action (bound to a menu).

This method has been added in version 0.18.

13.44.7.25 cm_edit_merge – "cm_edit_merge" action (bound to a menu).

This method has been added in version 0.18.

13.44.7.26 cm_edit_size – "cm_edit_size" action (bound to a menu).

This method has been added in version 0.18.

13.44.7.27 cm_exit – "cm_exit" action (bound to a menu).

13.44.7.28 cm_goto_position – "cm_goto_position" action (bound to a menu).

13.44.7.29 cm_help_about – "cm_help_about" action (bound to a menu).

13.44.7.30 cm_inc_max_hier – "cm_inc_max_hier" action (bound to a menu).

13.44.7.31 cm_last_display_state – "cm_last_display_state" action (bound to a menu).

13.44.7.32 cm_layout_props – "cm_layout_props" action (bound to a menu).

13.44.7.33 cm_load_bookmark – "cm_load_bookmarks" action (bound to a menu).

13.44.7.34 cm_load_layer_prop – "cm_load_layer_props" action (bound to a menu).

13.44.7.35 cm_lv_add_missing – "cm_lv_add_missing" action (bound to a menu).

13.44.7.36 cm_lv_delete – "cm_lv_delete" action (bound to a menu).

13.44.7.37 cm_lv_expand_all – "cm_lv_expand_all" action (bound to a menu).

13.44.7.38 cm_lv_group – "cm_lv_group" action (bound to a menu).

13.44.7.39 cm_lv_hide – "cm_lv_hide" action (bound to a menu).

13.44.7.40 cm_lv_hide_all – "cm_lv_hide_all" action (bound to a menu).

13.44.7.41 cm_lv_insert – "cm_lv_insert" action (bound to a menu).

13.44.7.42 cm_lv_new_tab – "cm_lv_new_tab" action (bound to a menu).

13.44.7.43 cm_lv_regroup_by_datatype – "cm_lv_regroup_by_datatype" action (bound to a menu).

13.44.7.44 cm_lv_regroup_by_index – "cm_lv_regroup_by_index" action (bound to a menu).

13.44.7.45 cm_lv_regroup_by_layer – "cm_lv_regroup_by_layer" action (bound to a menu).

13.44.7.46 cm_lv_regroup_flatten – "cm_lv_regroup_flatten" action (bound to a menu).

13.44.7.47 cm_lv_remove_tab – "cm_lv_remove_tab" action (bound to a menu).

13.44.7.48 cm_lv_remove_unused – "cm_lv_remove_unused" action (bound to a menu).

13.44.7.49 cm_lv_rename – "cm_lv_rename" action (bound to a menu).

13.44.7.50 cm_lv_rename_tab – "cm_lv_rename_tab" action (bound to a menu).

13.44.7.51 cm_lv_select_all – "cm_lv_select_all" action (bound to a menu).

13.44.7.52 cm_lv_show – "cm_lv_show" action (bound to a menu).

13.44.7.53 cm_lv_show_all – "cm_lv_show_all" action (bound to a menu).

13.44.7.54 cm_lv_show_only – "cm_lv_show_only" action (bound to a menu).

This method has been added in version 0.20.

13.44.7.55 cm_lv_sort_by_dli – "cm_lv_sort_by_dli" action (bound to a menu).

13.44.7.56 cm_lv_sort_by_idl – "cm_lv_sort_by_idl" action (bound to a menu).

13.44.7.57 cm_lv_sort_by_ild – "cm_lv_sort_by_ild" action (bound to a menu).

13.44.7.58 cm_lv_sort_by_ldi – "cm_lv_sort_by_ldi" action (bound to a menu).

13.44.7.59 cm_lv_sort_by_name – "cm_lv_sort_by_name" action (bound to a menu).

13.44.7.60 cm_lv_source – "cm_lv_source" action (bound to a menu).

13.44.7.61 cm_lv_ungroup – "cm_lv_ungroup" action (bound to a menu).

13.44.7.62 cm_manage_bookmarks – "cm_manage_bookmarks" action (bound to a menu).

13.44.7.63 cm_max_hier – "cm_max_hier" action (bound to a menu).

13.44.7.64 cm_max_hier_0 – "cm_max_hier_0" action (bound to a menu).

13.44.7.65 cm_max_hier_1 – "cm_max_hier_1" action (bound to a menu).

13.44.7.66 cm_navigator_close – "cm_navigator_close" action (bound to a menu).

13.44.7.67 cm_navigator_freeze – "cm_navigator_freeze" action (bound to a menu).

13.44.7.68 cm_new_cell – "cm_new_cell" action (bound to a menu).

This method has been added in version 0.18.

13.44.7.69 cm_new_layer – "cm_new_layer" action (bound to a menu).

This method has been added in version 0.18.

13.44.7.70 cm_new_layout – cm_new_layout" action (bound to a menu).

This method has been added in version 0.18.

13.44.7.71 cm_new_panel – "cm_new_panel" action (bound to a menu).

This method has been added in version 0.20.

13.44.7.72 cm_next_display_state – "cm_next_display_state" action (bound to a menu).

13.44.7.73 cm_open – "cm_open" action (bound to a menu).

13.44.7.74 cm_open_current_cell – "cm_open_current_cell" action (bound to a menu).

This method has been added in version 0.18.

13.44.7.75 cm_open_new_view – "cm_open_new_view" action (bound to a menu).

13.44.7.76 cm_open_too – "cm_open_too" action (bound to a menu).

13.44.7.77 cm_pan_down – "cm_pan_down" action (bound to a menu).

13.44.7.78 cm_pan_left – "cm_pan_left" action (bound to a menu).

13.44.7.79 cm_pan_right – "cm_pan_right" action (bound to a menu).

13.44.7.80 cm_pan_up – "cm_pan_up" action (bound to a menu).

13.44.7.81 cm_paste – "cm_paste" action (bound to a menu).

13.44.7.82 cm_pull_in – "cm_pull_in" action (bound to a menu).

This method has been added in version 0.20.

13.44.7.83 cm_reader_options – "cm_reader_options" action (bound to a menu).

This method has been added in version 0.18.

13.44.7.84 cm_redo – "cm_redraw" action (bound to a menu).

13.44.7.85 cm_redraw – "cm_redraw" action (bound to a menu).

13.44.7.86 cm_reload – "cm_reload" action (bound to a menu).

13.44.7.87 cm_restore_session – "cm_restore_session" action (bound to a menu).

This method has been added in version 0.18.

13.44.7.88 cm_save – "cm_save" action (bound to a menu).

This method has been added in version 0.18.

13.44.7.89 cm_save_as – "cm_save_as" action (bound to a menu).

This method has been added in version 0.18.

13.44.7.90 cm_save_bookmarks – "cm_save_bookmarks" action (bound to a menu).

13.44.7.91 cm_save_current_cell_as – "cm_save_current_cell_as" action (bound to a menu).

This method has been added in version 0.18.

13.44.7.92 cm_save_layer_props – "cm_save_layer_props" action (bound to a menu).

13.44.7.93 cm_save_session – "cm_save_session" action (bound to a menu).

This method has been added in version 0.18.

13.44.7.94 cm_screenshot – "cm_screenshot" action (bound to a menu).

13.44.7.95 cm_sel_flip_x – "cm_sel_flip_x" action (bound to a menu).

This method has been added in version 0.18.

13.44.7.96 cm_sel_flip_y – "cm_sel_flip_y" action (bound to a menu).

This method has been added in version 0.18.

13.44.7.97 cm_sel_free_rot – "cm_sel_free_rot" action (bound to a menu).

This method has been added in version 0.18.

13.44.7.98 cm_sel_move – "cm_sel_move" action (bound to a menu).

This method has been added in version 0.18.

13.44.7.99 cm_sel_rot_ccw – "cm_sel_rot_ccw" action (bound to a menu).

This method has been added in version 0.18.

13.44.7.100 cm_sel_rot_cw – "cm_sel_rot_cw" action (bound to a menu).

This method has been added in version 0.18.

13.44.7.101 cm_sel_scale – "cm_sel_scale" action (bound to a menu).

This method has been added in version 0.18.

13.44.7.102 cm_select_cell – "cm_select_cell" action (bound to a menu).

13.44.7.103 cm_select_current_cell – "cm_select_current_cell" action (bound to a menu).

13.44.7.104 cm_setup – "cm_setup" action (bound to a menu).

13.44.7.105 cm_show-properties – "cm_show_properties" action (bound to a menu).

13.44.7.106 cm_undo – "cm_undo" action (bound to a menu).

13.44.7.107 cm_view_log – "cm_view_log" action (bound to a menu).

This method has been added in version 0.20.

13.44.7.108 cm_zoom_fit – "cm_zoom_fit" action (bound to a menu).

13.44.7.109 cm_zoom_fit_sel – "cm_zoom_fit_sel" action (bound to a menu).

This method has been added in version 0.18.

13.44.7.110 cm_zoom_in – "cm_zoom_in" action (bound to a menu).

13.44.7.111 cm_zoom_out – "cm_zoom_out" action (bound to a menu).

13.44.8 [const] ref CellView create_layout(integer)
Create a new, empty layout.

Input:	0	Create a new layout in the current view, replacing the current layouts. Or
	1	Create a new layout in a new view and make this view the current one. Or
	2	Create a new layout adding it to the current view.
Return:	ref	A reference to a CellView object in which the layout was created.

13.44.9 ref LayoutView current_view
Return a reference to the current view's object.

Return:	ref	A reference to the LayoutView object representing the current view.

13.44.10 [const] integer current_view_index
Return the current view's index.

Return:	integer	The index of the current view.

13.44.11 destroy
Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.44.12 [const] boolean destroyed
Tell, if the object was destroyed.

Return:	true	The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
	false	The object is still alive.

13.44.13 enable_edits(boolean)
Enable or disable edits.

This method allows to put the application into read-only mode by disabling all edit functions. For doing so, this method has be called with a 'false" argument. Calling it with a 'true" parameter enables all edits again.

Input:	true	Enable edits, set the application into edit mode.
	false	Disable edits, set the application into read-only mode.

13.44.14 exit
Schedule an exit for the application.

This method does not immediately exit the application but sends an exit request to the application which will cause a clean shutdown of the GUI.

13.44.15 [const] double grid_micron
Get the global grid in micron.

The global grid is used at various places, i.e. for ruler snapping, for grid display etc. With this method it can be set to the desired value.

Return: double — The global grid in micron.

13.44.16 [const] ref CellView load_layout(filename, integer)
Load a new layout.

Input: filename — The file name to read.
- 0 — Loads the given file in the current view, replacing the current layouts. Or
- 1 — Loads the given file in a new view and make this view the current one. Or
- 2 — Loads the given file adding it to the current view.

Return: ref — A reference to a CellView object into which the layout was loaded.

13.44.17 ref AbstractMenu menu
Return a reference to the abstract menu.

Return: ref — A reference to an AbstractMenu object representing the menu system.

13.44.18 message(message, time)
Display a message in the status bar.

This given message is shown in the status bar for the given time.

This method has been added in version 0.18.

Input: message — The message to display.
time — The time how long to display the message in milliseconds.

13.44.19 ref LoadLayoutOptions reader_options
Access to the current reader options.

Modifying the current reader options will have an effect on the next load_layout operation but might not be reflected correctly in the reader options dialog and changes will be reset when the application is restarted.

This method was added in version 0.18.

Return: ref — A reference to a LoadLayoutOptions object representing the current reader options.

13.44.20 redraw
Redraw the current view.

Issues a redraw request to the current view. This usually happens automatically, so this method does not need to be called in most relevant cases.

13.44.21 remove_current_view_observer(ref ObserverBase observer)
Remove an observer for the change of the "current view changed" event.

13.44.22 remove_new_view_observer(ref ObserverBase observer)
Remove an observer for a "new view" event.

13.44.23 resize(width, height)
Re-size the window.

This method re-sizes the window to the given target size including decoration such as menu bar and control panels

Input:	width	The new width of the window.
	height	The new width of the window.

13.44.24 restore_session(filename)
Restore a session from the given file.

The session stored in the given session file is restored. All existing views are closed and all layout edits are discarded without notification.

This method was added in version 0.18.

Input:	filename	The path and file name of the session file to restore from.

13.44.25 save_session(filename)
Save the session to the given file.

The session is saved to the given session file. Any existing layout edits are not automatically saved together with the session. The session just holds display settings and annotation objects. If layout edits must be saved, this has to be done explicitly in a separate step.

This method was added in version 0.18.

Input:	filename	The path and file name of the session file to save into.

13.44.26 select_view(integer)
Select the view with the given index.

This method will make the view with the given index the current (front) view.

Input:	integer	The index of the view to select (0 is the first one).

13.44.27 synchronous=(boolean)
Put the main window into synchronous mode.

A synonym for: synchroneous(boolean).

In synchronous mode, an application is allowed to block on redraw. While redrawing, no user interactions are possible. Although this is not desirable for smooth operation, it can be beneficial for test or automation purposes, i.e. if a screen shot needs to be produced once the application has finished drawing.

Input:	true	The application should behave synchronously.
	false	The application should behave asynchronously.

13.44.28 ref LayoutView view(index)
Return a reference to a view object by index.

Return: ref A reference to a LayoutView object representing the view with the given index.

13.44.29 [const] unsigned views
Return the number of views.

Return: unsigned The number of views available so far.

13.45 Class Manager (version 0.21) A transaction manager class.

Manager objects control layout and potentially other objects in the layout database and allow to queue operations to form transactions. A transaction is a sequence of operations that can be undone or redone.

In order to equip a layout object with undo/redo support, instantiate the layout object with a manager attached and embrace the operations to undo/redo with transaction/commit calls.

The use of transactions is subject to certain constraints, i.e. transacted sequences may not be mixed with non-transacted ones.

This class has been introduced in version 0.19.

Method Overview

transaction	Begin a transaction.
commit	Close a transaction.
undo	Undo the current transaction.
redo	Redo the next available transaction.
has_undo?	Determine if a transaction is available for "undo".
transaction_for_undo	Return the description of the next transaction for "undo".
has_redo?	Determine if a transaction is available for "redo".
transaction_for_redo	Return the description of the next transaction for "redo".
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.45.1 commit
Close a transaction.

13.45.2 destroy
Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.45.3 [const] boolean destroyed
Tell, if the object was destroyed.

Return:	true	The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
	false	The object is still alive.

13.45.4 [const] boolean has_redo?
Determine if a transaction is available for "redo".

Return:	true	A transaction is available.
	false	No transaction is available.

13.45.5 has_undo?
Determine if a transaction is available for "undo".

Return:	true	A transaction is available.
	false	No transaction is available.

13.45.6 redo
Redo the next available transaction.

The next transaction is redone with this method. The has_redo? method can be used to determine whether there are transactions to undo.

13.45.7 transaction(description)
Begin a transaction.

This call will open a new transaction. A transaction consists of a set of operations issued with the "queue" method. A transaction is closed with the commit method.

Input: description The description for this transaction.
Comment: Which "queue" method?

13.45.8 [const] description transaction_for_redo
Return the description of the next transaction for "redo".

Return: description The description of the next transaction for "redo".

13.45.9 [const] description transaction_for_undo
Return the description of the next transaction for "undo".

Return: description The description of the next transaction for "undo".

13.45.10 undo
Undo the current transaction.

The current transaction is undone with this method. The has_undo? method can be used to determine whether there are transactions to undo.

13.46 Class Marker (version 0.21)
The floating-point coordinate marker object.

The marker is a visual object that "marks" (highlights) a certain area of the layout, given by a database object. This object accepts database objects with floating-point coordinates in micron values.

Method Overview

new	The constructor for a marker.
set	Set the box the marker is to display.
set	Set the text the marker is to display.
set	Set the edge the marker is to display.
set	Set the path the marker is to display.
set	Set the polygon the marker is to display.
color=	Set the color of the marker.
reset_color	Reset the color of the marker.
color	Get the color of the marker.
has_color?	True, if the marker has a specific color.
frame_color=	Set the frame color of the marker.
reset_frame_color	Reset the frame color of the marker.
frame_color	Get the frame color of the marker.
has_frame_color?	True, if the marker has a specific frame color.
line_width=	Set the line width of the marker.
line_width	Get the line width of the marker.
vertex_size=	Set the vertex size of the marker.
vertex_size	Get the vertex size of the marker.
halo=	Set the halo flag.
halo	Get the halo flag.
dither_pattern=	Set the stipple pattern index.
dither_pattern	Get the stipple pattern index.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.46.1 [const] unsigned color
Get the color of the marker.

This value is valid only if has_color? is true.

Return: unsigned The color of the marker.

13.46.2 color=(unsigned)
Set the color of the marker.

The color is a 32 bit unsigned integer encoding the RGB values in the lower 3 bytes (blue in the lowest significant byte). The color can be reset with reset_color, in which case, the default foreground color is used.

Input: unsigned The color of the marker.

13.46.3 destroy
Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.46.4 [const] boolean destroyed
Tell, if the object was destroyed.

Return: true — The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
false — The object is still alive.

13.46.5 [const] index dither_pattern
Get the stipple pattern index.

See dither_pattern= for a description of the stipple pattern index.

Return: index — The stipple pattern index.

13.46.6 dither_pattern=(index)
Set the stipple pattern index.

A value of -1 or less than zero indicates that the marker is not filled. Otherwise, the value indicates which pattern to use for filling the marker.

Input: index — The stipple pattern index.

13.46.7 [const] unsigned frame_color
Get the frame color of the marker.

This value is valid only if has_frame_color? is true.

The set method has been added in version 0.20.

Return: unsigned — The frame color of the marker.

13.46.8 frame_color=(unsigned)
Set the frame color of the marker.

The color is a 32 bit unsigned integer encoding the RGB values in the lower 3 bytes (blue in the lowest significant byte). The color can be reset with reset_frame_color, in which case the fill color is used.

The set method has been added in version 0.20.

Input: unsigned — The frame color of the marker.

13.46.9 [const] integer halo
Get the halo flag.

See halo= for a description of the halo flag.

Return: integer — The halo flag.

13.46.10 halo=(integer)
Set the halo flag.

Input:	-1	Take the default.
	0	Disable the halo.
	1	Enable the halo: a pixel border with the background color is drawn around the marker, the vertices and texts.

13.46.11 [const] boolean has_color?
True, if the marker has a specific color.

Return:	true	The marker has a specific color.
	false	The marker has no specific color.

13.46.12 [const] boolean has_frame_color?
True, if the marker has a specific frame color.

The set method has been added in version 0.20.

Return:	true	The marker has a specific frame color.
	false	The marker has no specific frame color.

13.46.13 [const] integer line_width
Get the line width of the marker.

See line_width= for a description of the line width.

Return:	integer	The line width of the marker.

13.46.14 line_width=(integer)
Set the line width of the marker.

This is the width of the line drawn for the outline of the marker.

Input:	integer	The line width of the marker.

13.46.15 [const] Marker new(ref LayoutView view)
The constructor for a marker.

A marker is always associated with a view, in which it is shown. The view this marker is associated with must be passed to the constructor.

Input:	ref	A reference to the view the marker is associated with.
Return:	Marker	The marker object.

13.46.16 reset_color
Reset the color of the marker.

See color= for a description of the color property of the marker.

13.46.17 reset_frame_color
Reset the frame color of the marker.

See frame_color= for a description of the frame color property of the marker.

The set method has been added in version 0.20.

13.46.18 set(DPolygon polygon)
Set the polygon the marker is to display.

A synonym for: set_polygon(DPolygon polygon).

The set method has been added in version 0.20.

Input: polygon Makes the marker show a polygon which must be given in micron units.

13.46.19 set(DPath path)
Set the path the marker is to display.

A synonym for: set_path(DPath path).

The set method has been added in version 0.20.

Input: path Makes the marker show a path which must be given in micron units.

13.46.20 set(DBox box)
Set the box the marker is to display.

A synonym for: set_box(DBox box).

The set method has been added in version 0.20.

Input: box Makes the marker show a box which must be given in micron units. In case the box is empty, no marker is drawn.

13.46.21 set(DEdge edge)
Set the edge the marker is to display.

A synonym for: set_edge(DEdge edge).

The set method has been added in version 0.20.

Input: edge Makes the marker show an edge which must be given in micron units.

13.46.22 set(DText text)
Set the text the marker is to display.

A synonym for: set_text(DText text).

The set method has been added in version 0.20.

Input: text Makes the marker show a text which must be given in micron units.

13.46.23 vertex_size
Get the vertex size of the marker.

See vertex_size= for a description.

13.46.24 vertex_size=(integer)
Set the vertex size of the marker.

Input: integer The size of the rectangles drawn for the vertices object.

13.47 Class MessageBox (version 0.21) Various methods to display message boxes.

Method Overview

b_...	Various "b_..." constant describing the respective button label.
b_ok	"b_ok" constant describing the respective button label.
b_cancel	"b_cancel" constant describing the respective button label.
b_yes	"b_yes" constant describing the respective button label.
b_no	"b_no" constant describing the respective button label.
b_abort	"b_abort" constant describing the respective button label.
b_retry	"b_retry" constant describing the respective button label.
b_ignore	"b_ignore" constant describing the respective button label.
warning	Open a warning message box.
question	Open a question message box.
info	Open a information message box.
critical	Open a critical (error) message box.
assign	Assign the contents of another object to self.
dup	Creates a copy of self..
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.47.1 assign(MessageBox other) Assign the contents of another object to self.

This method assigns the contents of another object to self. This is a deep copy that does not only copy the reference but the actual content.

13.47.2 [static] integer b_... Various "b_..." constant describing the respective button label.

13.47.2.1 [static] integer b_abort – "b_abort" constant describing the respective button label.

Return: integer The "b_abort" constant.

13.47.2.2 [static] integer b_cancel – "b_cancel" constant describing the respective button label.

Return: integer The "b_cancel" constant.

13.47.2.3 [static] integer b_ignore – "b_ignore" constant describing the respective button label.

Return: integer The "b_ignore" constant.

13.47.2.4 [static] integer b_no – "b_no" constant describing the respective button label.

Return: integer The "b_no" constant.

13.47.2.5 [static] integer b_ok – "b_ok "constant describing the respective button label.

Return: integer The "b_ok" constant.

13.47.2.6 [static] integer b_retry – "b_retry" constant describing the respective button label.

Return: integer The "b_retry" constant.

13.47.2.7 [static] integer b_yes – "b_yes" constant describing the respective button label.

Return: integer The "b_yes" constant.

13.47.3 [static] integer critical(title, text, buttons)
Open a critical (error) message box.

Input: title The title of the window.
text The text to show.
buttons A combination (+) of "b_..." constants describing the buttons to show for the message box.
Return: integer The "b_..." constant describing the button that was pressed.

13.47.4 destroy
Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.47.5 [const] boolean destroyed
Tell, if the object was destroyed.

Return: true The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
false The object is still alive.

13.47.6 [const] MessageBox dup
Creates a copy of self.

Return: MessageBox The copy of self.

13.47.7 [static] integer info(title, text, buttons)
Open an information message box.

Input: title The title of the window.
text The text to show.
buttons A combination (+) of "b_..." constants describing the buttons to show for the message box.
Return: integer The "b_..." constant describing the button that was pressed.

13.47.8 [static] integer question(title, text, buttons)
Open a question message box.

Input:	title	The title of the window.
	text	The text to show.
	buttons	A combination (+) of "b_..." constants describing the buttons to show for the message box.
Return:	integer	The "b_..." constant describing the button that was pressed.

13.47.9 [static] integer warning(title, text, buttons)
Open a warning message box.

Input:	title	The title of the window.
	text	The text to show.
	buttons	A combination (+) of "b_..." constants describing the buttons to show for the message box.
Return:	integer	The "b_..." constant describing the button that was pressed.

13.48 Class Method (version 0.21)
The interface to a method declaration.

Method Overview

each_argument	Iterate over all arguments of this method.
ret_type	The return type of this method.
is_const?	True, if this method does not alter the object.
is_static?	True, if this method is static (a class method).
is_event?	True, if this method is an event.
name	The name of the class.
doc	The documentation string for this method.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.48.1 destroy
Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.48.2 [const] boolean destroyed
Tell, if the object was destroyed.

Return: true — The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
false — The object is still alive.

13.48.3 [const] string doc
The documentation string for this method.

Return: string — The documentation string for this method.

13.48.4 [const] yield const ref ArgType each_argument
Iterate over all arguments of this method.

13.48.5 [const] boolean is_const?
True, if this method does not alter the object.

Return: true — This method does not alter the object.
false — This method alters the object.

13.48.6 [const] boolean is_event?
True, if this method is an event.

Return: true — This method is an event.
false — This method is not an event.

13.48.7 is_static?

True, if this method is static (a class method).

Return:	true	This method is static (a class method).
	false	This method is not static.

13.48.8 [const] string name

The name of the class.

Return:	string	The name of the class.

13.48.9 [const] const ref ArgType ret_type

The return type of this method.

13.49 Class ObjectInstPath (version 0.21) A class describing a selected shape or instance.

A shape or instance is addressed by a path which describes all instances leading to the specified object. These instances are described through InstElement objects, which describe the instance and, in case of array instances, the specific array member. For shapes, additionally the layer and the shape itself is specified. The ObjectInstPath objects encapsulates both forms, which can be distinguished with the is_cell_inst? attribute.

Method Overview

cv_index	Accessor to the cell view index that describes which cell view the shape or instance is located in.
cell_index	Accessor to the cell index of the cell that the selection applies to..
source	Returns to the cell index of the cell that the selected element resides inside..
trans	Accessor to the transformation applicable for the shape.
source_trans	Accessor to the transformation applicable for an instance and shape..
layer	Accessor to the layer index that describes which layer the selected shape is on.
shape	Accessor to the shape object that describes the selected shape geometrically.
inst	Deliver the instance represented by this selection.
is_cell_inst?	True, if this selection represents a cell instance.
seq	The sequence number.
path_length	Returns the length of the path (number of elements delivered by each_inst).
path_nth	Returns the nth element of the path (similar to each_inst but with direct access through the index).
each_inst	Yield the instantiation path.
assign	Assign the contents of another object to self.
dup	Creates a copy of self.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.49.1 assign(ObjectInstPath other) Assign the contents of another object to self.

This method assigns the contents of another object to self. This is a deep copy that does not only copy the reference but the actual content.

13.49.2 [const] unsigned cell_index Accessor to the cell index of the cell that the selection applies to.

This method returns the cell index that describes which cell the selected shape is located in or the cell whose instance is selected if is_cell_inst? is true.

13.49.3 [const] unsigned cv_index

Accessor to the cell view index that describes which cell view the shape or instance is located in.

13.49.4 destroy

Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.49.5 [const] boolean destroyed

Tell, if the object was destroyed.

Return:	true	The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
	false	The object is still alive.

13.49.6 [const] ObjectInstPath dup

Creates a copy of self.

Return:	ObjectInstPath	The copy of self.

13.49.7 [const] yield const ref InstElement each_inst

Yield the instantiation path.

The instantiation path describes by a sequence of InstElement objects the path by which the cell containing the selected shape is found from the cell view's current cell. If this object represents an instance, the path will contain the selected instance as the last element. The elements are delivered top down.

13.49.8 [const] const ref Instance inst

Deliver the instance represented by this selection.

This method delivers valid results only if is_cell_inst? is true. It returns the instance reference (an Instance object) that this selection represents.

This method has been added in version 0.16.

13.49.9 [const] boolean is_cell_inst?

True, if this selection represents a cell instance.

If this attribute is true, the shape reference and layer are not valid.

13.49.10 [const] unsigned layer

Accessor to the layer index that describes which layer the selected shape is on.

This method delivers valid results only for object selections that represent shapes, i.e for which is_cell_inst? is false.

13.49.11 [const] unsigned path_length
Returns the length of the path (number of elements delivered by each_inst).

This method has been added in version 0.16.

13.49.12 [const] const ref InstElement path_nth(unsigned n)
Returns the nth element of the path (similar to each_inst but with direct access through the index).

This method has been added in version 0.16.

Input: unsigned n The index of the element to retrieve (0...path_length-1).

13.49.13 [const] unsigned long seq
The sequence number.

The sequence number describes when the item was selected. A sequence number of 0 indicates that the item was selected in the first selection action (without 'Shift' pressed).

13.49.14 [const] const ref Shape shape
Accessor to the shape object that describes the selected shape geometrically.

This method delivers valid results only for object selections that represent shapes, i.e for which is_cell_inst? is false.

13.49.15 [const] unsigned source
Returns the cell index of the cell that the selected element resides inside.

If this reference represents a cell instance, this method delivers the index of the cell in which the cell instance resides. Otherwise, this method returns the same value than cell_index.

This method has been added in version 0.16.

13.49.16 [const] CplxTrans source_trans
Accessor to the transformation applicable for an instance and shape.

If this object represents a shape, this transformation describes how the selected shape is transformed into the current cell of the cell view. If this object represents an instance, this transformation describes how the selected instance is transformed into the current cell of the cell view. This method is similar to trans, except that the resulting transformation does not include the instance transformation if the object represents an instance.

This method has been added in version 0.16.

13.49.17 [const] CplxTrans trans
Accessor to the transformation applicable for the shape.

If this object represents a shape, this transformation describes how the selected shape is transformed into the current cell of the cell view. Basically, this transformation is the accumulated transformation over the instantiation path. If the ObjectInstPath represents a cell instance, this includes the transformation of the selected instance as well.

13.50 Class Observer (version 0.21)
This class implements an event handler for use with "observer" interfaces.

Some classes provide callbacks by attaching 'observer' objects to certain events. Specific actions can be implemented by reimplementing the "signal..." methods of this class.

Method Overview

signal	This method is called when the event without value is issued.
signal_int	This method is called when an event associated with an integer is issued.
assign	Assign the contents of another object to self.
dup	Creates a copy of self.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.50.1 assign(Observer other)
Assign the contents of another object to self.

This method assigns the contents of another object to self. This is a deep copy that does not only copy the reference but the actual content.

13.50.2 destroy
Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.50.3 [const] boolean destroyed
Tell, if the object was destroyed.

Return: true — The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
false — The object is still alive.

13.50.4 [const] Observer dup
Creates a copy of self.

Return: Observer — The copy of self.

13.50.5 signal
This method is called when the event without value is issued.

13.50.6 signal_int(integer)
This method is called when an event associated with an integer is issued.

Input: integer — The integer value to associate to the event.

13.51 Class ObserverBase (version 0.21) This class implements an event handler for use with 'observer' interfaces.

Some classes provide callbacks by attaching Observer objects to certain events. Specific actions can be implemented by reimplementing the "signal..." methods of this class.

Method Overview

assign	Assign the contents of another object to self.
dup	Creates a copy of self.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.51.1 assign(ObserverBase other) Assign the contents of another object to self.

This method assigns the contents of another object to self. This is a deep copy that does not only copy the reference but the actual content.

13.51.2 destroy Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.51.3 [const] boolean destroyed Tell, if the object was destroyed.

Return:	true	The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
	false	The object is still alive.

13.51.4 [const] ObserverBase dup Creates a copy of self.

Return:	ObserverBase	The copy of self.

13.52 Class ParentInstArray (version 0.21)
This class implements an event handler for use with 'observer' interfaces.

Some classes provide callbacks by attaching Observer objects to certain events. Specific actions can be implemented by reimplementing the "signal..." methods of this class.

Method Overview

parent_cell_index	Retrieve the reference to the parent cell.
child_inst	Retrieve the child instance associated with this parent instance.
inst	Compute the inverse instance by which the parent is seen from the child.
assign	Assign the contents of another object to self.
dup	Creates a copy of self.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.52.1 assign(ParentInstArray other)
Assign the contents of another object to self.

This method assigns the contents of another object to self. This is a deep copy that does not only copy the reference but the actual content.

13.52.2 [const] Instance child_inst
Retrieve the child instance associated with this parent instance.

Starting with version 0.15, this method returns an Instance object rather than a CellInstArray reference.

13.52.3 destroy
Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.52.4 [const] boolean destroyed
Tell, if the object was destroyed.

Return:	true	The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
	false	The object is still alive.

13.52.5 [const] ParentInstArray dup
Creates a copy of self.

Return: ParentInstArray The copy of self.

13.52.6 [const] CellInstArray inst

Compute the inverse instance by which the parent is seen from the child.

13.52.7 [const] unsigned parent_cell_index

Retrieve the reference to the parent cell.

13.53 Class Path (version 0.21)
An path class with integer coordinates.

A path consists of an sequence of line segments forming the "spine" of the path and a width. In addition, the starting point can be drawn back by a certain extent (the "begin extension") and the end point can be pulled forward somewhat (by the "end extension"). A path may have round ends for special purposes.

Method Overview

new	Default constructor: creates an empty (invalid) path with width 0.
new	Constructor given the points of the path's spine and the width.
new	Constructor given the points of the path's spine, the width and the extensions.
new	Constructor given the points of the path's spine, the width, the extensions and the round end flag.
<	Less operator.
==	Equality test.
!=	Inequality test.
points=	Set the points of the path.
each_point	Get the points that make up the path's spine.
points	Get the number of points.
width=	Set the width.
width	Get the width.
bgn_ext=	Set the begin extension.
bgn_ext	Get the begin extension.
end_ext=	Set the end extension.
end_ext	Get the end extension.
round=	Set the 'round ends' flag.
is_round?	Tell, if the path has round ends.
move	Moves the path.
moved	Returns the moved path.
transformed	Transform the path.
transformed_cplx	Transform the path.
transformed_cplx	Transform the path.
to_s	Convert to a string.
simple_polygon	Convert the path to a simple polygon.
polygon	Convert the path to a polygon.
bbox	Return the bounding box of the path.
from_dpath	Construct an integer coordinate path from a floating-point coordinate one.
assign	Assign the contents of another object to self.
dup	Creates a copy of self.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.53.1 [const] boolean !=(Path p)
Inequality test.

Input:	p	The object to compare against.
Return:	true	Inequality.
	false	???.

13.53.2 [const] boolean <(Path p)
Less operator.

This operator is provided to establish some, not necessarily a certain sorting order.

Input:	p	The object to compare against.
Return:	true	The path is less then the argument path.
	false	The path is greater then the argument path.

13.53.3 [const] boolean ==(Path p)
Equality test.

Input:	p	The object to compare against.
Return:	true	Equality.
	false	???.

13.53.4 assign(Path other)
Assign the contents of another object to self.

This method assigns the contents of another object to self. This is a deep copy that does not only copy the reference but the actual content.

13.53.5 [const] Box bbox
Return the bounding box of the path.

Return:	Box	The bounding box.

13.53.6 [const] integer bgn_ext
Get the begin extension.

Return:	integer	The begin extension.

13.53.7 bgn_ext=(integer)
Set the begin extension.

Input:	integer	The begin extension.

13.53.8 destroy
Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.53.9 [const] boolean destroyed
Tell, if the object was destroyed.

Return:	true	The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
	false	The object is still alive.

13.53.10 [const] Path dup
Creates a copy of self.

Return: Path The copy of self.

13.53.11 [const] yield DPoint each_point
Get the points that make up the path's spine.

Return: yield DPoint An array of points.

13.53.12 [const] integer end_ext
Get the end extension.

Return: integer The end extension.

13.53.13 end_ext=(integer)
Set the end extension.

Input: integer The end extension.

13.53.14 [static] Path from_dpath(DPath double_path)
Construct a floating-point coordinate path from an integer coordinate one.

This method has been added in version 0.15.

13.53.15 [const] boolean is_round?
Tell, if the path has round ends.

Return: true The path has round ends.
false The path has other ends.

13.53.16 refPath move(DPoint p)
Moves the path.

Moves the path by the given offset and returns the reference of the moved path. The path is overwritten.

Input: p The distance to move the path.
Return: ref The reference of the moved path.

13.53.17 [const] Path moved(DPoint p)
Returns the moved path.

Moves the path by the given offset and returns the reference of the moved path. The path is not modified.

Input: p The distance to move the path.
Return: Path The moved path.

13.53.18 [static] Path new
Default constructor: creates an empty (invalid) path with width 0.

Return: Path The empty (invalid) path.

13.53.19 [static] Path new(Point pts[], width)
Constructor given the points of the path's spine and the width.

A synonym for: [static] Path new_pw(Point pts[], width).

Input:	pts[]	The points forming the spine of the path.
	width	The width of the path.
Return:	Path	The resulting path.

13.53.20 [static] Path new(DPoint pts[], width, bgn_ext, end_ext)
Constructor given the points of the path's spine, the width and the extensions.

A synonym for: [static] Path new_pwx(DPoint pts[], width, bgn_ext, end_ext).

Input:	pts[]	The points forming the spine of the path.
	width	The width of the path.
	bgn_ext	The begin extension of the path.
	end_ext	The end extension of the path.
Return:	Path	The resulting path.

13.53.21 [static] Path new(Point pts[], width, bgn_ext, end_ext, boolean round)
Constructor given the points of the path's spine, the width, the extensions and the round end flag.

A synonym for: [static] Path new_pwxr(Point pts[], width, bgn_ext, end_ext, boolean round).

Input:	pts[]	The points forming the spine of the path.
	width	The width of the path.
	bgn_ext	The begin extension of the path.
	end_ext	The end extension of the path.
	boolean round	If this flag is true, the path will get rounded ends.
Return:	Path	The resulting path.

13.53.22 [const] unsigned points
Get the number of points.

Return:	unsigned	The number of points.

13.53.23 points=(Point pts[])
Set the points of the path.

Input:	pts[]	An area of points forming the spine of the path.

13.53.24 [const] Polygon polygon
Convert the path to a polygon.

The returned polygon is not guaranteed to be non-selfoverlapping. This may happen if the path overlaps itself or contains very short segments.

Return:	Polygon	The resulting polygon.

13.53.25 round=(boolean)
Set the "round ends" flag.

Input:	true	"round ends".
	false	Other ends.

13.53.26 [const] SimplePolygon simple_polygon
Convert the path to a simple polygon.

The returned polygon is not guaranteed to be non-selfoverlapping. This may happen if the path overlaps itself or contains very short segments.

Return:	SimplePolygon	The resulting polygon.

13.53.27 [const] string to_s
Convert to a string.

Return:	string	The resulting string.

13.53.28 [const] Path transformed(Trans t)
Transform the path.

Transforms the path with the given transformation. Does not modify the path but returns the transformed path.

Input:	t	The transformation to apply.
Return:	Path	The transformed path.

13.53.29 [const] Path transformed_cplx(ICplxTrans t)
Transform the path.

Transforms the path with the given complex transformation. Does not modify the path but returns the transformed path.

This method has been introduced in version 0.18.

Input:	t	The transformation to apply.
Return:	Path	The transformed path (in this case an integer coordinate path).

13.53.30 [const] DPath transformed_cplx(CplxTrans t)
Transform the path.

Transforms the path with the given complex transformation. Does not modify the path but returns the transformed path.

Input:	t	The transformation to apply.
Return:	Path	The transformed path.

13.53.31 [const] integer width
Get the width.

Return:	integer	The width of the path.

13.53.32 width=(integer)
Set the width.

Input: integer The width of the path.

13.54 Class Point (version 0.21)
A integer point class with integer coordinates.

Method Overview

from_dpoint	Create an integer coordinate point from a floating-point coordinate point.
new	Default constructor: creates a point at 0,0.
new	Constructor for a point from two coordinate values.
+	Add one point to another.
–	Subtract one point from another.
<	"less" comparison operator.
==	Equality test operator.
!=	Inequality test operator.
x	Accessor to the x coordinate.
y	Accessor to the y coordinate.
x=	Write accessor to the x coordinate.
y=	Write accessor to the y coordinate.
*****	Scaling by some factor.
distance	The euclidean distance to another point.
sq_distance	The square euclidean distance to another point.
to_s	String conversion.
assign	Assign the contents of another object to self.
dup	Creates a copy of self.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.54.1 [const] boolean !=(Point p)
Inequality test operator.

Input:	p	The given integer coordinate point.
Return:	true	This and the given point are unequal.
	false	This and the given point are equal.

13.54.2 [const] Point *(double f)
Scaling by some factor.

Scaling may involve rounding for integer coordinate points.

Input:	double f	The given floating-point scaling factor.
Return:	Point	The scaled integer coordinate point.

13.54.3 [const] Point +(Point p)
Add one point to another.

Add point p to self by adding the coordinates.

Input:	p	The given integer coordinate point.
Return:	Point	The resulting integer coordinate point.

13.54.4 [const] Point –(Point p)
Subtract one point to another.

Subtract point p from self by subtracting the coordinates.

Input:	p	The given integer coordinate point.
Return:	Point	The resulting integer coordinate point.

13.54.5 [const] boolean <(Point p)
"less" comparison operator.

This operator is provided to establish a sorting order.

Input:	p	The given integer coordinate point.
Return:	true	This point is 'less' than the given one.
	false	This point is 'greater' than the given one.

13.54.6 [const] boolean ==(Point p)
Equality test operator.

Input:	p	The given integer coordinate point.
Return:	true	This point and the given point are equal.
	false	This point and the given point are unequal.

13.54.7 assign(Point other)
Assign the contents of another object to self.

This method assigns the contents of another object to self. This is a deep copy that does not only copy the reference but the actual content.

13.54.8 destroy
Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.54.9 [const] boolean destroyed
Tell, if the object was destroyed.

Return:	true	The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
	false	The object is still alive.

13.54.10 [const] double distance(Point d)
The euclidean distance to another point.

Input:	d	The other point to compute the distance to.
Return:	double	The euclidean distance.

13.54.11 [const] Point dup

Creates a copy of self.

Return: Point — The copy of self.

13.54.12 [static] Point from_dpoint(DPoint p)

Create an integer coordinate point from a floating-point coordinate point.

Input: p — The given floating-point coordinate point.
Return: Point — The created integer coordinate point.

13.54.13 [static] Point new

Default constructor: creates a point at 0,0.

Return: Point — The created integer coordinate point at coordinate 0,0.

13.54.14 [static] Point new(x, y)

Constructor for a point from two coordinate values.

Input: x — The given x part of the coordinate.
y — the given y part of the coordinate.
Return: Point — The created integer coordinate point.

13.54.15 [const] double sq_distance(Point d)

The square euclidean distance to another point.

Input: d — The other point to compute the distance to.
Return: double — The square euclidean distance.

13.54.16 [const] string to_s

String conversion.

Return: string — The point as string.

13.54.17 [const] integer x

Accessor to the x part of the coordinate.

Return: integer — The x part of the integer coordinate point.

13.54.18 x=(integer)

Write accessor to the x part of the coordinate.

Input: integer — The x part of the integer coordinate point.

13.54.19 [const] integer y

Accessor to the y part of the coordinate.

Return: integer — The y part of the integer coordinate point.

13.54.20 y=(integer)
Write accessor to the y part of the coordinate.

Input: integer The y part of the integer coordinate point.

13.55 Class Polygon (version 0.21)
A polygon class with integer coordinates.

A polygon consists of an outer hull and zero to many holes. Each contour consists of several points. The point list is normalized such that the leftmost, lowest point is the first one. The orientation is normalized such that the orientation of the hull contour is clockwise, while the orientation of the holes is counter-clockwise.

It is in no way checked that the contours are not over-lapping. This must be ensured by the user of the object when filling the contours.

Method Overview

new	Default constructor: creates an empty (invalid) polygon.
new	Constructor given the points of the polygon hull.
new	Constructor converting a box to a polygon.
<	Less operator.
==	Equality test.
!=	Inequality test.
hull=	Set the points of the hull of polygon.
assign_hole	Set the points of the given hole of the polygon.
points	Get the total number of points (hull plus holes).
point_hull	Get a specific point of the hull@args p.
point_hole	Get a specific point of a hole@args n,p.
points_hull	Get the number of points of the hull.
points_hole	Get the number of points of the given hole.
insert_hole	Insert a hole with the given points.
each_point_hull	Iterate over the points that make up the hull.
each_point_hole	Iterate over the points that make up the nth hole.
size	Sizing (biasing).
size	Sizing (biasing).
holes	Get the number of holes.
each_edge	Iterate over the edges that make up the polygon.
inside	Test, if the given point is inside the polygon.
compress	Compress the polygon.
move	Moves the polygon.
moved	Returns the moved polygon.
transformed	Transform the polygon.
transformed_cplx	Transform the polygon with a complex transformation.
transformed_cplx	Transform the polygon with a complex transformation.
to_s	Convert to a string.
area	The area of the polygon.
bbox	Return the bounding box of the polygon.
from_dpoly	Construct an integer coordinate polygon from a floating-point coordinate one.
round_corners	Round the corners of the polygon.
assign	Assign the contents of another object to self.
dup	Creates a copy of self.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.55.1 [const] boolean !=(Polygon p)
Inequality test.

Input:	p	The object to compare against.
Return:	true	Inequality.
	false	???.

13.55.2 [const] boolean <(Polygon p)
Less operator.

This operator is provided to establish some, not necessarily a certain sorting order.

Input:	p	The object to compare against.
Return:	true	This polygon is less than the given one.
	false	???.

13.55.3 [const] boolean ==(Polygon p)
Equality test.

Input:	p	The object to compare against.
Return:	true	The polygons are equal.
	false	???.

13.55.4 [const] long area
The area of the polygon.

The area is correct only if the polygon is not self-overlapping and oriented clockwise.

Return:	long	The area of the polygon.

13.55.5 assign(Polygon other)
Assign the contents of another object to self.

This method assigns the contents of another object to self. This is a deep copy that does not only copy the reference but the actual content.

13.55.6 assign_hole(unsigned, Point p[])
Set the points of the given hole of the polygon.

If the hole index is not valid, this method does nothing.

This method was introduced in version 0.18.

Input:	unsigned	The index of the hole to which the points should be assigned.
	p[]	An array of points to assign to the polygon's hole.

13.55.7 [const] const refBox bbox
Return the bounding box of the polygon.

13.55.8 compress(boolean)
Compress the polygon.

Removes redundant points from the polygon, such as points being on a line formed by two other points.

Input: true — Additionally removes points if the two adjacent edges form a spike.
false — Basic behavior.

13.55.9 destroy
Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.55.10 [const] boolean destroyed
Tell, if the object was destroyed.

Return: true — The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
false — The object is still alive.

13.55.11 [const] Polygon dup
Creates a copy of self.

Return: Polygon — The copy of self.

13.55.12 yield Edge each_edge
Iterate over the edges that make up the polygon.

Return: yield The array of the edges that make up the polygon.

13.55.13 [const] yield Point each_point_hole(unsigned)
Iterate over the points that make up the nth hole.

Input: unsigned — The hole number, which must be equal or less than the number of holes (see holes)

13.55.14 [const] yield Point each_point_hull
Iterate over the points that make up the hull.

Return: yield The array of the points that make up the hull.

13.55.15 [static] Polygon from_dpoly(DPolygon double_poly)
Construct a floating-point coordinate polygon from an integer coordinate one.

This method has been added in version 0.15.

13.55.16 [const] unsigned holes
Get the number of holes.

Return: unsigned The number of holes.

13.55.17 hull=(Point p[])
Set the points of the hull of polygon.

A synonym for: assign_hull(Point p[]).

The 'assign_hull' variant is provided in analogy to 'assign_hole'.

Input: p[] An array of points to assign to the polygon's hull.

13.55.18 insert_hole(Point p[])
Insert a hole with the given points.

Input: p[] An array of points to insert as a new hole.

13.55.19 [const] boolean inside(Point p)
Test, if the given point is inside the polygon.

This tests works well only if the polygon is not self-overlapping and oriented clockwise.

Input: true The given point is inside the polygon.
false The given point is outside the polygon.

13.55.20 ref Polygon move(Point p)
Moves the polygon.

Moves the polygon by the given offset and returns the reference of the moved polygon. The polygon is overwritten.

Input: p The distance to move the polygon.
Return: ref The reference of the moved polygon.

13.55.21 [const] Polygon moved(Point p)
Returns the moved polygon.

Moves the polygon by the given offset and returns the moved polygon. The polygon is not modified.

Input: p The distance to move the polygon.
Return: Polygon The moved polygon.

13.55.22 [static] Polygon new
Default constructor: creates an empty (invalid) polygon.

13.55.23 [static] Polygon new(Box box)
Constructor converting a box to a polygon.

A synonym for: [static] Polygon new_b(Box box).

Input: box The box to convert to a polygon.

13.55.24 [static] Polygon new(Point p[])
Constructor given the points of the polygon hull.

A synonym for: [static] Polygon new_p(Point p[]).

Input: p[] An array of points to insert as a new polygon hull.

13.55.25 Point point_hole(unsigned n, unsigned p)
Get a specific point of a hole@args n,p.

This method was introduced in version 0.18.

Input: unsigned n — The index of the hole to which the points should be assigned.
unsigned p — The index of the point to get. If the index of the point or of the hole is not valid, a default value is returned.
Return: Point — The specific hole point.

13.55.26 Point point_hull(unsigned p)
Get a specific point of a hull@args p.

This method was introduced in version 0.18.

Input: unsigned p — The index of the point to get. If the index of the point is not a valid index, a default value is returned.
Return: Point — The specific hull point.

13.55.27 unsigned points
Get the total number of points (hull plus holes).

This method was introduced in version 0.18.

13.55.28 unsigned points_hole(unsigned n)
Get the number of points of the given hole.

The argument gives the index of the hole of which the number of points are requested. The index must be less than the number of holes, see holes.

Input: unsigned n The given hole.
Return: unsigned The number of holes.

13.55.29 unsigned points_hull
Get the number of points of the hull.

Return: unsigned The number of points of the hull.

13.55.30 [const] Polygon round_corners(double rinner, double router, unsigned n)
Round the corners of the polygon.

Replaces the corners of the polygon with circle segments.

This method was introduced in version 0.20.

Input: double rinner — The circle radius of inner corners (in database units).
double router — The circle radius of outer corners (in database units).
unsigned n — The number of points per full circle.
Return: Polygon — The new polygon.

13.55.31 size(d, unsigned mode)
Sizing (biasing).

Shifts the contour outwards (d>0) or inwards (d<0). May create invalid (self-overlapping, reverse oriented) contours.

Input:	double d	The distance to shift the contour in x and y direction.
	0	Bending angle cutoff occurs at greater than 0 degree.
	1	Bending angle cutoff occurs at greater than 45 degree.
	2	Bending angle cutoff occurs at greater than 90 degree.
	3	Bending angle cutoff occurs at greater than 135 degree.
	4	Bending angle cutoff occurs at greater than approximately 168 degree.
	other	Bending angle cutoff occurs at greater than approximately 179 degree.

13.55.32 size(dx, dy, unsigned mode)
Sizing (biasing).

Shifts the contour outwards (dx,dy>0) or inwards (dx,dy<0). May create invalid (self-overlapping, reverse oriented) contours. The sign of dx and dy should be identical.

Input:	double dx	The x value to shift the contour.
	double dy	The y value to shift the contour.
	0	Bending angle cutoff occurs at greater than 0 degree.
	1	Bending angle cutoff occurs at greater than 45 degree.
	2	Bending angle cutoff occurs at greater than 90 degree.
	3	Bending angle cutoff occurs at greater than 135 degree.
	4	Bending angle cutoff occurs at greater than approximately 168 degree.
	other	Bending angle cutoff occurs at greater than approximately 179 degree.

13.55.33 string to_s
Convert to a string.

Return:	string	The string.

13.55.34 [const] Polygon transformed(Trans t)
Transform the polygon.

Transforms the polygon with the given transformation. Does not modify the polygon but returns the transformed polygon.

Input:	t	The transformation to apply.
Return:	Polygon	The transformed polygon.

13.55.35 [const] Polygon transformed_cplx(CplxTrans t)
Transform the polygon.

Transforms the polygon with the given transformation. Does not modify the polygon but returns the transformed polygon.

Input:	t	The transformation to apply.
Return:	Polygon	The transformed polygon.

13.55.36 [const] Polygon transformed_cplx(ICplxTrans t)
Transform the polygon.

Transforms the polygon with the given transformation. Does not modify the polygon but returns the transformed polygon.

This method was introduced in version 0.18.

Input:	t	The transformation to apply.
Return:	Polygon	The transformed polygon (in this case an integer coordinate polygon).

13.56 Class RdbCategory (version 0.21) The report database category.

Every item in the report database is assigned to a category. A category is a DRC rule check for example. Categories can be organized hierarchically, i.e. a category may have sub-categories. Item counts are summarized for categories and items belonging to sub-categories of one category can be browsed together for example. As a general rule, categories not being leaf categories (having child categories) may not have items.

Method Overview

rdb_id	Get the category ID.
name	Get the category name.
path	Get the category path.
description	Get the category description.
description=	Set the category description.
each_sub_category	Iterate over all sub-categories.
parent	Get the parent category of this category.
num_items	Get the number of items in this category.
num_items_visited	Gets the number of visited items in this category.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.56.1 [const] description description
Get the category description.

Return: description — The description string.

13.56.2 description=(description)
Set the category description.

Input: description — The description string.

13.56.3 destroy
Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.56.4 [const] boolean destroyed
Tell, if the object was destroyed.

Return: true — The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.

false — The object is still alive.

13.56.5 yield ref RdbCategory each_sub_category
Iterate over all sub-categories.

13.56.6 [const] string name
Get the category name.

The category name is an string that identifies the category in the context of a parent category or inside the database when it is a top level category. The name is not the path name which is a path to a child category and incorporates all names of parent categories.

Return: string The category name.

13.56.7 [const] unsigned num_items
Get the number of items in this category.

Return: unsigned The number of items includes the items in sub-categories of this category.

13.56.8 [const] unsigned num_items_visited
Get the number of visited items in this category.

Return: unsigned The number of visited items includes the visited items in sub-categories of this category.

13.56.9 ref RdbCategory parent
Get the parent category of this category.

Return: ref A reference representing the parent category or nil if this category is a top-level category.

13.56.10 [const] string path
Get the category path.

The category path is the category name for top level categories. For child categories, the path contains the names of all parent categories separated by a dot.

Return: string The path for this category.

13.56.11 [const] unsigned rdb_id
Get the category ID.

The category ID is an integer that uniquely identifies the category. It is used for referring to a category in RdbItem for example.

Return: unsigned The category ID.

13.57 Class RdbCell (version 0.21) A report database cell representation.

This class represents a cell in the report database. There is not necessarily a 1:1 correspondence of RDB cells and layout database cells. Cells have an ID, a name, optionally a variant name and a set of references which describe at least one example instantiation in some parent cell. The references do not necessarily map to references or cover all references in the layout database.

Method Overview

rdb_id	Get the cell ID.
name	Get the cell name.
variant	Get the cell variant name.
qname	Get the cell's qualified name.
num_items	Get the number of items for this cell.
num_items_visited	Get the number of visited items for this cell.
add_reference	Add a reference to the references of this cell.
clear_references	Remove all references from this cell.
each_reference	Iterate over all references.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.57.1 add_reference(RdbReference ref) Adds a reference to the references of this cell.

Input: ref — The reference to add.

13.57.2 clear_references Remove all references from this cell.

13.57.3 destroy Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.57.4 [const] boolean destroyed Tell, if the object was destroyed.

Return: true — The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.

false — The object is still alive.

13.57.5 yield ref RdbReference each_reference Iterate over all references.

13.57.6 [const] string name Get the cell name.

The cell name is a string that identifies the category in the database. Additionally, a cell may carry a variant identifier which is a string that uniquely identifies a cell in the context of it's variants. The "qualified name"

contains both the cell name and the variant name. Cell names are also used to identify report database cell's with layout cells.

Return: string The cell name.

13.57.7 [const] unsigned num_items
Get the number of items for this cell.

Return: unsigned The number of items for this cell.

13.57.8 [const] unsigned num_items_visited
Get the number of visited items for this cell.

Return: unsigned The number of visited items for this cell.

13.57.9 [const] string qname
Get the cell's qualified name.

The qualified name is a combination of the cell name and optionally the variant name. It is used to identify the cell by name in a unique way.

Return: string The qualified cell name.

13.57.10 [const] unsigned rdb_id
Get the cell ID.

The cell ID is an integer that uniquely identifies the cell. It is used for referring to a cell in RdbItem for example.

Return: unsigned The cell ID.

13.57.11 [const] string variant
Get the cell variant name.

A variant name additionally identifies the cell when multiple cells with the same name are present. A variant name is either assigned automatically or set when creating a cell.

Return: string The cell variant name.

13.58 Class RdbItem (version 0.21) A RDB item.

An item is the basic information entity in the RDB. It is associated with a cell and a category. It can be assigned values which encapsulate other objects such as strings and geometrical objects. In addition, items can be assigned an image (i.e. a screen shot image) and tags which are basically Boolean flags that can be defined freely.

Method Overview

cell_id	Get the cell ID.
category_id	Get the category ID.
is_visited?	Get a value indicating whether the item was already visited.
add_tag	Add a tag with the given id to the item.
remove_tag	Remove the tag with the given id from the item.
has_tag?	Return a value indicating whether the item has a tag with the given ID.
tags_str	Return a string listing all tags of this item.
tags_str=	Set the tags from a string.
image_str	Get the image associated with this item as a string.
image_str=	Set the image from a string.
add_value	Add a value object to the values of this item.
clear_values	Remove all values from this item.
each_value	Iterate over all values.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.58.1 add_tag(unsigned) Add a tag with the given id to the item.

Each tag can be added once to the item. The tags of an item thus form a set. If a tag with that ID already exists, this method does nothing.

Input: unsigned The tag ID.

13.58.2 add_value(RdbItemValue value) Add a value object to the values of this item.

Input: value The value to add.

13.58.3 [const] unsigned category_id Get the category ID.

Return: unsigned The ID of the category that this item is associated with.

13.58.4 [const] unsigned cell_id Get the cell ID.

Return: unsigned The ID of the cell that this item is associated with.

13.58.5 clear_values
Removes all values from this item.

13.58.6 destroy
Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.58.7 [const] boolean destroyed
Tell, if the object was destroyed.

Return:	true	The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
	false	The object is still alive.

13.58.8 [const] yield const ref RdbItemValue each_value
Iterate over all values.

13.58.9 [const] boolean has_tag?(unsigned)
Return a value indicating whether the item has a tag with the given ID.

Input:	unsigned	The given ID.
Return:	true	The item has a tag with the given ID.
	false	The item has no tag with the given ID.

13.58.10 [const] string image_str
Get the image associated with this item as a string.

Return:	string	A base64-encoded image file (usually in PNG format).

13.58.11 image_str=(string)
Set the image from a string.

Input:	string	A base64-encoded image file (usually in PNG format).

13.58.12 [const] boolean is_visited?
Get a value indicating whether the item was already visited.

Return:	true	The item has been visited already.
	false	The item has not been visited already.

13.58.13 remove_tag(unsigned)
Remove the tag with the given id from the item.

If a tag with that ID does not exists on this item, this method does nothing.

Input:	unsigned	The given ID.

13.58.14 [const] string tags_str

Return a string listing all tags of this item.

Return: string A comma-separated list of tags.

13.58.15 tags_str=(string)

Set the tags from a string.

Input: string A comma-separated list of tags.

13.59 Class RdbItemValue (version 0.21) A RDB value object.

Value objects are attached to items to provide markers. An arbitrary number of such value objects can be attached to an item. Currently, a value can represent a box, a polygon or an edge. Geometrical objects are represented in micron units and are therefore "D" type objects (DPolygon, DEdge and DBox).

Method Overview

from_s	Create a value object from a string.
new	Create a value representing a string.
new	Create a value representing a DPolygon object.
new	Create a value representing a DEdge object.
new	Create a value representing a DBox object.
to_s	Convert a value to a string.
is_string?	Return true if the value object represents a string.
string	Get the string if the value represents one or nil if it does not.
is_polygon?	Return true if the value object represents a polygon.
polygon	Get the polygon if the value represents one or nil if it does not.
is_edge?	Return true if the value object represents an edge.
edge	Get the edge if the value represents one or nil if it does not.
is_box?	Return true if the value object represents a box.
box	Get the box if the value represents one or nil if it does not.
assign	Assign the contents of another object to self.
dup	Creates a copy of self.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.59.1 assign(RdbItemValue other) Assign the contents of another object to self.

This method assigns the contents of another object to self. This is a deep copy that does not only copy the reference but the actual content.

13.59.2 [const] DBox box Get the box if the value represents one or nil if it does not.

Return: DBox — The DBox object or nil.

13.59.3 destroy Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.59.4 [const] boolean destroyed Tell, if the object was destroyed.

Return: true — The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.

false — The object is still alive.

13.59.5 [const] RdbItemValue dup
Creates a copy of self.

Return: RdbItemValue The copy of self.

13.59.6 [const] DEdge edge
Get the edge if the value represents one or nil if it does not.

Return: DEdge The copy of self.

13.59.7 [static] RdbItemValue from_s(string)
Create a value object from a string.

Input: string The given string. The string format is the same than obtained by the to_s method.
Return: RdbItemValue The created value object.

13.59.8 [const] boolean is_box?
Returns true if the value object represents a box.

Return: true The value object represents a box.
false The value object represents not a box.

13.59.9 [const] boolean is_edge?
Returns true if the value object represents an edge.

Return: true The value object represents an edge.
false The value object represents not an edge.

13.59.10 [const] boolean is_polygon?
Returns true if the value object represents a polygon.

Return: true The value object represents a polygon.
false The value object represents not a polygon.

13.59.11 [const] boolean is_string?
Returns true if the value object represents a string.

Return: true The value object represents a string.
false The value object represents not a string.

13.59.12 [static] ref RdbItemValue new(string)
Create a value representing a string.

Input: string The given string.
Return: ref A reference representing a string.

13.59.13 [static] ref RdbItemValue new(DPolygon)
Create a value representing a DPolygon object.

Input: DPolygon The given object.
Return: ref A reference representing a DPolygon object.

13.59.14 [static] ref RdbItemValue new(DBox)
Create a value representing a DBox object.

Input: DBox The given object.
Return: ref A reference representing a DBox object.

13.59.15 [static] ref RdbItemValue new(DEdge)
Create a value representing a DEdge object.

Input: DEdge The given object.
Return: ref A reference representing an DEdge object.

13.59.16 [const] DPolygon polygon
Get the polygon if the value represents one or nil if it does not.

Return: DPolygon The DPolygon object or nil.

13.59.17 [const] string string
Get the string if the value represents one or nil if it does not.

Return: string The string object or nil.

13.59.18 [const] string to_s
Convert a value to a string.

The string can be used by the string constructor to create another object from it.

Return: string The string converted from a value.

13.60 Class RdbReference (version 0.21) A cell reference.

This class describes a cell reference. Such reference object can be attached to cells to describe instantiations of them in parent cells. Not necessarily all instantiations of a cell in the layout database are represented by references and in some cases there might even be no references at all. The references are merely a hint how a marker must be displayed in the context of any other, potentially parent, cell in the layout database.

Method Overview

new	Create a reference with a given transformation and parent cell ID.
trans	Gets the transformation for this reference.
trans=	Sets the transformation for this reference.
parent_cell_id	Gets parent cell ID for this reference.
parent_cell_id=	Sets the parent cell ID for this reference.
assign	Assign the contents of another object to self.
dup	Creates a copy of self.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.60.1 assign(RdbReference other) Assign the contents of another object to self.

This method assigns the contents of another object to self. This is a deep copy that does not only copy the reference but the actual content.

13.60.2 destroy Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.60.3 [const] boolean destroyed Tell, if the object was destroyed.

Return:	true	The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
	false	The object is still alive.

13.60.4 [const] RdbReference dup Creates a copy of self.

Return:	RdbReference	The copy of self.

13.60.5 [static] RdbReference new(DCplxTrans t, unsigned) Create a reference with a given transformation and parent cell ID.

Input:	unsigned	The parent cell ID.
	t	The given transformation.
Return:	RdbReference	The created reference.

13.60.6 [const] unsigned parent_cell_id
Get parent cell ID for this reference.

Return: unsigned The parent cell ID.

13.60.7 parent_cell_id=(unsigned)
Set the parent cell ID for this reference.

Input: unsigned The parent cell ID.

13.60.8 [const] const ref DCplxTrans trans
Get the transformation for this reference.

The transformation describes the transformation of the child cell into the parent cell. In that sense that is the usual transformation of a cell reference.

Return: ref The transformation for this reference.
Comment: Return value(s) not clear.

13.60.9 trans=(DCplxTrans t)
Set the transformation for this reference.

Input: t The transformation for this reference.

13.61 Class RecursiveShapeIterator (version 0.21) This class implements an event handler for use with 'observer' interfaces.

Some classes provide callbacks by attaching Observer objects to certain events. Specific actions can be implemented by reimplementing the "signal..." methods of this class.

Method Overview

max_depth=	Specify the maximum hierarchy depth to look into.
shape_flags=	Specify the shape selection flags.
trans	Get the current transformation by which the shapes must be transformed into the initial cell.
itrans	Get the current transformation by which the shapes must be transformed into the initial cell.
shape	Get the current shape.
at_end?	End of iterator predicate.
cell_index	Get the current cell's index.
next	Increment the iterator.
==	Comparison of iterators - equality test.
!=	Comparison of iterators - inequality test.
assign	Assign the contents of another object to self.
dup	Creates a copy of self.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.61.1 [const] boolean !=(RecursiveShapeIterator p) Inequality test.

Two iterators are not equal if they do not point to the same shape.

Input:	p	The object to compare against.
Return:	true	Inequality.
	false	???.

13.61.2 [const] boolean ==(RecursiveShapeIterator p) Equality test.

Two iterators are equal if they point to the same shape.

Input:	p	The object to compare against.
Return:	true	Equality.
	false	???.

13.61.3 assign(RecursiveShapeIterator other) Assign the contents of another object to self.

This method assigns the contents of another object to self. This is a deep copy that does not only copy the reference but the actual content.

13.61.4 [const] boolean at_end?
End of iterator predicate.

A synonym for: [const] boolean at_end.

Return: true — The iterator is at the end of the sequence.
false — The iterator is in between the sequence.

13.61.5 [const] unsigned cell_index
Get the current cell's index.

Return: unsigned — The cell index of the current cell.

13.61.6 destroy
Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.61.7 [const] boolean destroyed
Tell, if the object was destroyed.

Return: true — The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
false — The object is still alive.

13.61.8 [const] RecursiveShapeIterator dup
Creates a copy of self.

Return: RecursiveShapeIterator The copy of self.

13.61.9 [const] ICplxTrans itrans
Get the current transformation by which the shapes must be transformed into the initial cell.

The shapes delivered are not transformed. Instead, this transformation must be applied to get the shape in the coordinate system of the top cell. This method delivers the integer version which is not accurate in the strict sense but delivers integer coordinate shapes. This method is somewhat slower than the 'trans' method.

Return: ??? — The integer version of the shapes in the coordinate system of the top cell.

13.61.10 max_depth=(integer)
Specify the maximum hierarchy depth to look into.

A depth of 0 instructs the iterator to deliver only shapes from the initial cell. The depth must be specified before the shapes are being retrieved.

Input: integer — The maximum hierarchy depth to look into.

13.61.11 next
Increment the iterator.

This moves the iterator to the next shape inside the search scope.

13.61.12 [const] Shape shape
Get the current shape.

Returns the shape currently referred to by the recursive iterator. This shape is not transformed yet and is located in the current cell.

13.61.13 shape_flags=(unsigned)
Specify the shape selection flags.

The flags are the same then being defined in Shapes (the default is Shapes.s_all). The flags must be specified before the shapes are being retrieved.

Input: unsigned The shape selection flags.

13.61.14 [const] const ref CplxTrans trans
Get the current transformation by which the shapes must be transformed into the initial cell.

The shapes delivered are not transformed. Instead, this transformation must be applied to get the shape in the coordinate system of the top cell.

13.62 Class ReportDatabase (version 0.21) The report database object.

A report database is organized around a set of items which are associated with cells and categories. Categories can be organized hierarchically by created sub-categories of other categories. Cells are associated with layout database cells and can come with an example instantiation if the layout database does not allow a unique association of the cells. Items in the database can have a variety of attributes: values, tags and an image object. Values are geometrical objects for example. Tags are a set of boolean flags and an image can be attached to an item to provide a screen shot for visualization for example. This is the main report database object. The basic use case of this object is to create one inside a LayoutView and populate it with items, cell and categories or load it from a file. Another use case is to create a standalone ReportDatabase object and use the methods provided to perform queries or to populate it.

Method Overview

new	Create a report database.
description	Get the databases description.
description=	Set the databases description.
generator	Get the databases generator.
generator=	Set the generator string.
filename	Get the file name and path where the report database is stored.
name	Get the database name.
top_cell_name	Get the top cell name.
top_cell_name=	Set the top cell name string.
original_file	Get the original file name and path.
original_file=	Set the original file name and path.
tag_id	Get the tag ID for a given tag name.
set_tag_description	Set the tag description for the given tag ID.
tag_description	Get the tag description for the given tag ID.
each_category	Iterate over all top-level categories.
create_category	Create a new top level category.
create_category	Create a new sub-category.
category_by_path	Get a category by path.
category_by_id	Get a category by ID.
create_cell	Create a new cell.
create_cell	Create a new cell, potentially as a variant for a cell with the same name.
variants	Get the variants for a given cell name.
cell_by_qname	Return the cell for a given qualified name.
cell_by_id	Return the cell for a given ID.
each_cell	Iterate over all cells.
num_items	Return the number of items inside the database.
num_items_visited	Return the number of items already visited inside the database.
num_items	Return the number of items inside the database for a given cell/category combination.
num_items_visited	Return the number of items visited already for a given cell/category combination.
create_item	Create a new item for the given cell/category combination.
is_modified?	Return a value indicating whether the database has been modified.
reset_modified	Reset the modified flag.
each_item	Iterates over all items inside the database.
each_item_per_cell	Iterate over all items inside the database which are associated with the given cell.
each_item_per_category	Iterate over all items inside the database which are associated with the given

	category.
each_item_per_cell_and_category	Iterate over all items inside the database which are associated with the given cell and category.
set_item_visited	Modify the visited state of an item.
load	Load the database from the given file.
save	Save the database to the given file.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.62.1 [const] const ref RdbCategory category_by_id(unsigned)
Get a category by ID.

Input:	unsigned	The ID of the category.
Return:	RdbCategory	The (const) category object or nil if the ID is not valid.

13.62.2 [const] const ref RdbCategory category_by_path(path)
Get a category by path.

Input:	path	The full path to the category starting from the top level (subcategories separated by dots).
Return:	RdbCategory	The (const) category object or nil if the name is not valid.

13.62.3 [const] const ref RdbCell cell_by_id(unsigned)
Return the cell for a given ID.

Input:	unsigned	The ID of the cell.
Return:	RdbCell	The (const) cell object or nil if the ID is not valid.

13.62.4 [const] const ref RdbCell cell_by_qname(qname)
Return the cell for a given qualified name.

Input:	qname	The qualified name of the cell (name plus variant name optionally).
Return:	RdbCell	The (const) category object or nil if the name is not valid.

13.62.5 ref RdbCategory create_category(name)
Create a new top level category.

Input:	name	The name of the category.

13.62.6 ref RdbCategory create_category(ref RdbCategory parent, name)
Create a new sub-category.

Input:	parent	The category under which the category should be created.
	name	The name of the category.

13.62.7 ref RdbCell create_cell(name, variant)

Create a new cell, potentially as a variant for a cell with the same name.

Input:	name	The name of the cell.
	parent	The variant name of the cell.

13.62.8 ref RdbCell create_cell(name)

Create a new cell.

Input:	name	The name of the cell.

13.62.9 ref RdbItem create_item(unsigned cell_id, unsigned category_id)

Create a new item for the given cell/category combination.

Input:	unsigned cell_id	The ID of the cell to which the item is associated.
	unsigned category_id	The ID of the category to which the item is associated.

13.62.10 [const] string description

Get the database description.

The description is a general purpose string that is supposed to further describe the database and it's content in a human-readable form.

Return:	string	The description string.

13.62.11 description=(string)

Set the databases description.

Input:	string	The description string.

13.62.12 destroy

Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.62.13 [const] boolean destroyed

Tell, if the object was destroyed.

Return:	true	The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
	false	The object is still alive.

13.62.14 [const] yield const ref RdbCategory each_category

Iterate over all top-level categories.

13.62.15 [const] yield const ref RdbCell each_cell

Iterate over all cells.

13.62.16 [const] yield const ref RdbItem each_item

Iterate over all item inside the database.

13.62.17 [const] yield const ref RdbItem each_item_per_category(unsigned category_id)

Iterate over all items inside the database which are associated with the given category.

Input: unsigned category_id — The ID of the category for which all associated items should be retrieved.

13.62.18 [const] yield const ref RdbItem each_item_per_cell(unsigned cell_id)

Iterate over all items inside the database which are associated with the given cell.

Input: unsigned cell_id — The ID of the cell for which all associated items should be retrieved.

13.62.19 [const] yield const ref RdbItem each_item_per_cell_and_category(unsigned cell_id, unsigned category_id)

Iterate over all items inside the database which are associated with the given cell and category.

Input: unsigned cell_id — The ID of the cell for which all associated items should be retrieved.
unsigned category_id — The ID of the category for which all associated items should be retrieved.

13.62.20 [const] string filename

Get the file name and path where the report database is stored.

This property is set when a database is saved or loaded. It cannot be set manually.

Return: string — The file name and path.

13.62.21 [const] string generator

Get the database generator.

The generator string describes how the database was created, i.e. DRC tool name and tool options. In a later version this should allow to rerun the tool that created the report.

Return: string — The generator string.

13.62.22 generator=(string)

Set the generator string.

Input: string — The generator string.

13.62.23 [const] boolean is_modified?

Return a value indicating whether the database has been modified.

Return:	true	The database has been modified.
	false	The database is unmodified.

13.62.24 load(string)

Load the database from the given file.

The reader recognizes the format automatically and will choose the appropriate decoder. "gzip" compressed files are uncompressed automatically.

Input:	string	The file name and path.

13.62.25 [const] string name

Get the database name.

The name of the database is supposed to identify the database within a layout view context. The name is modified to be unique when a database is entered into a layout view.

Return:	string	The database name.

13.62.26 [static] ReportDatabase new(string)

Create a report database.

The name of the database will be used in the user interface to refer to a certain database.

Input:	string	The database name.

13.62.27 [const] unsigned num_items

Return the number of items inside the database.

Return:	unsigned	The total number of items.

13.62.28 [const] unsigned num_items(unsigned cell_id, unsigned category_id)

Return the number of items inside the database for a given cell/category combination.

Input:	unsigned cell_id	The ID of the cell for which to retrieve the number.
	unsigned category_id	The ID of the category for which to retrieve the number.
Return:	unsigned	The total number of items.

13.62.29 [const] unsigned num_items_visited

Return the number of items already visited inside the database.

Return:	unsigned	The total number of items already visited.

13.62.30 [const] unsigned num_items_visited(unsigned cell_id, unsigned category_id)

Return the number of items already visited inside the database for a given cell/category combination.

Input:	unsigned cell_id	The ID of the cell for which to retrieve the number.
	unsigned category_id	The ID of the category for which to retrieve the number.
Return:	unsigned	The total number of items already visited.

13.62.31 [const] string original_file

Get the original file name and path.

The original file name is supposed to describe the file from which this report database was generated.

Return:	string	The original file name and path.

13.62.32 original_file=(string)

Set the original file name and path.

Input:	string	The original file name and path.

13.62.33 reset_modified

Reset the modified flag.

13.62.34 save(string)

Saves the database to the given file.

The database is always saved in KLayout's XML-based format.

Input:	string	The file name and path to which to save the database.

13.62.35 set_item_visited(RdbItem item, boolean)

Modify the visited state of an item.

Input:	item	The item to modify.
Return:	true	Set the item to visited state.
	false	Set the item to none visited state.

13.62.36 set_tag_description(unsigned tag_id, string)

Set the tag description for the given tag ID.

Input:	unsigned tag_id	The ID of the tag.
	string	The description string.

13.62.37 [const] string tag_description(unsigned tag_id)

Get the tag description for the given tag ID.

Input:	unsigned tag_id	The ID of the tag.
Return:	string	The description string.

13.62.38 [const] unsigned tag_id tag_id(string)
Get the tag ID for a given tag name.

This method will always succeed and the tag will be created if it does not exist yet.

Input: string The description string.
Return: unsigned tag_id The ID of the tag.

13.62.39 [const] string top_cell_name
Get the top cell name.

The top cell name identifies the top cell of the design for which the report was generated. This property must be set to establish a proper hierarchical context for a hierarchical report database.

Return: string The top cell name.

13.62.40 top_cell_name=(string)
Set the top cell name string.

Input: string The top cell name.

13.62.41 unsigned[] variants(string)
Get the variants for a given cell name.

Input: (string) The basic name of the cell.
Return: unsigned[] An array of ID's representing cells that are variants for the given base name.

13.63 Class SaveLayoutOptions (version 0.21) Options for saving layout.

This class describes the various options for saving a layout to a stream file (GDS2, OASIS). There are: layers to be saved, cell or cells to be saved, scale factor, format, database unit and format specific options. Usually the default constructor provides a suitable object. The layers are specified by either selecting all layers or by defining layer by layer using the add_layer method. select_all_layers will explicitly select all layers for saving, deselect_all_layers will explicitly clear the list of layers.

Cells are selected in a similar fashion: by default, all cells are selected. Using add_cell, specific cells can be selected for saving. All these cells plus their hierarchy will then be written to the stream file.

Method Overview

new	Default constructor
format=	Select a format.
format	Get the format name.
add_layer	Add a layer to be saved.
select_all_layers	Select all layers to be saved.
deselect_all_layers	Deselect all layers: no layer will be saved.
add_cell	Add a cell (plus hierarchy) to be saved.
select_all_cells	Select all cells to save.
dbu=	Set the database unit to be used in the stream file.
dbu	Get the explicit database unit if one is set.
no_empty_cells=	Don't write empty cells if this flag is set.
no_empty_cells	Returns a flag indicating whether empty cells are not written..
scale_factor=	Set the scaling factor for the saving.
scale_factor	Get the scaling factor currently set.
dxf_dbu=	Specifies the database unit which the reader uses and produces.
dxf_dbu	Specifies the database unit which the reader uses and produces.
cif_wire_mode=	How to read "W" objects.
wire_mode	Specifies how to read "W" objects.
cif_dbu=	Specifies the database unit which the reader uses and produces.
cif_dbu	Specifies the database unit which the reader uses and produces.
gds2_max_vertex_count=	Set the maximum number of vertices for polygons to write.
gds2_max_vertex_count	Get the maximum number of vertices for polygons to write.
gds2_multi_xy_records=	Use multiple XY records in BOUNDARY elements for unlimited large polygons.
gds2_multi_xy_records	Get the property enabling multiple XY records for BOUNDARY elements.
gds2_max_cellname_length=	Maximum length of cell names.
gds2_max_cellname_length	Get the maximum length of cell names.
gds2_libname=	Set the library name.
gds2_libname	Get the library name.
gds2_user_units=	Set the users units to write into the GDS file.
gds2_user_units	Get the user units.
gds2_box_mode=	Specify how to treat BOX records.
box_mode	Specifies how to treat BOX records.
gds2_allow_multi_xy_records=	Allows the use of multiple XY records in BOUNDARY elements for unlimited large polygons.
gds2_allow_multi_xy_records	Specifies whether to allow big polygons with multiple XY records..
gds2_allow_big_records=	Allow big records with more than 32767 bytes.
gds2_allow_big_records	Specifies whether to allow big records with a length of 32768 to 65535 bytes.
oasis_compression_level=	Set the OASIS compression level.
oasis_compression_level	Get the OASIS compression level.

assign	Assign the contents of another object to self.
dup	Creates a copy of self.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.63.1 add_cell(unsigned)
Add a cell (plus hierarchy) to be saved.

This method clears the select_all_cells flag.

Input: unsigned — The index of the cell. It must be a valid index in the context of the layout to be saved.

13.63.2 add_layer(unsigned, LayerInfo properties)
Add a layer to be saved .

Input: unsigned — The index of the layer to add to the layer list that will be written. If all layers have been selected previously, this state will be cleared.

properties — The properties argument can be used to assign different layer properties than the ones present in the layout. Pass a default LayerInfo object to this argument to use the properties from the layout object. Construct a valid LayerInfo object with explicit layer, data type and possibly a name to override the properties stored in the layout.

13.63.3 assign(SaveLayoutOptions other)
Assign the contents of another object to self.

This method assigns the contents of another object to self. This is a deep copy that does not only copy the reference but the actual content.

13.63.4 [const] unsigned box_mode
Specifies how to treat BOX records.

See gds2_box_mode= method for a description of this mode.

This property has been added in version 0.18.

Return: unsigned — The box mode property.

13.63.5 [const] double cif_dbu
Specifies the database unit which the reader uses and produces.

See cif_dbu= method for a description of this property.

This property has been added in version 0.21.

Return: double — The database unit.

13.63.6 cif_dbu=(double)
Specifies the database unit which the reader uses and produces.

This property has been added in version 0.21.

Input: double The database unit.

13.63.7 cif_wire_mode=(unsigned)
How to read "W" objects.

This property has been added in version 0.21.

This property specifies how to read "W" (wire) objects. Allowed values are:

Input: 0 Read wire objects as square ended paths.
1 Read wire objects as flush ended paths.
2 Read wire objects as round ended paths.

13.63.8 [const] double dbu
Get the explicit database unit if one is set.

Return: double The database unit.

13.63.9 dbu=(double)
Set the database unit to be used in the stream file.

By default, the database unit of the layout is used. This method allows to explicitly use a different database unit. This effectively scales the layout.

Input: double The database unit.

13.63.10 deselect_all_layers
Deselect all layers: no layer will be saved.

This method will clear all layers selected with add_layer so far and clear the select_all_layers flag. Using this method is the only way to save a layout without any layers.

13.63.11 destroy
Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.63.12 [const] boolean destroyed
Tell, if the object was destroyed.

Return: true The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
false The object is still alive.

13.63.13 [const] SaveLayoutOptions dup
Creates a copy of self.

Return: SaveLayoutOptions The copy of self.

13.63.14 [const] double dxf_dbu
Specifies the database unit which the reader uses and produces.

See dxf_dbu= method for a description of this property.

This property has been added in version 0.21.

Return: double The database unit used by the reader.

13.63.15 dxf_dbu=(double)
Specifies the database unit which the reader uses and produces.

This property has been added in version 0.21.

Input: double The database unit to be used by the reader.

13.63.16 [const] string format
Get the format name.

Return:
- GDS2 String for GDS format.
- OASIS String for OASIS format.
- other Other formats may be available if a suitable plug-in is installed.

13.63.17 format=
Select a format.

Input:
- GDS2 String for GDS format.
- OASIS String for OASIS format.
- other Other formats may be available if a suitable plug-in is installed.

13.63.18 [const] boolean gds2_allow_big_records
Specifies whether to allow big records with a length of 32768 to 65535 bytes.

See gds2_allow_big_records= method for a description of this property.

This property has been added in version 0.18.

Return:
- true Records with more than 32767 bytes allowed.
- false Records uses less than 32767 bytes.

13.63.19 gds2_allow_big_records=(boolean)
Allow big records with more than 32767 bytes.

This property has been added in version 0.18.

Input:
- true The default allows the use of larger records by treating the record length as unsigned short, which for example allows larger polygons (8000 points rather than 4000 points) without using multiple XY records.
- false For strict standard compatibility the use of larger records is forbidden.

13.63.20 [const] boolean gds2_allow_multi_xy_records
Specifies whether to allow big polygons with multiple XY records..

See gds2_allow_multi_xy_records= method for a description of this property.

This property has been added in version 0.18.

Return: true — The use of big polygons is allowed.
false — The use of big polygons is forbidden.

13.63.21 gds2_allow_multi_xy_records=(boolean)
Allows the use of multiple XY records in BOUNDARY elements for unlimited large polygons.

This property has been added in version 0.18.

Input: true — The default allows the use of big polygons that span over multiple XY records.
false — For strict standard compatibility the use of big polygons is forbidden.

13.63.22 gds2_box_mode=(unsigned)
Specify how to treat BOX records.

This property has been added in version 0.18.

This property specifies how to treat BOX records. Allowed values are:

Input: 0 — Ignore BOX records.
1 — Treat BOX records as rectangles. The default.
2 — Treat BOX records as boundaries.
3 — Treat BOX records as errors.

13.63.23 [const] string gds2_libname
Get the library name.

See gds2_libname= method for a description of the library name.

This property has been added in version 0.18.

Return: string — The GDS lib name.

13.63.24 gds2_libname=(string)
Set the library name.

The library name is the string written into the LIBNAME records of the GDS file. The library name should not be an empty string and is subject to certain limitations in the character choice.

This property has been added in version 0.18.

Input: string — The GDS lib name.

13.63.25 [const] unsigned gds2_max_cellname_length
Get the maximum length of cell names.

See gds2_max_cellname_length= method for a description of the maximum cell name length.

This property has been added in version 0.18.

Return: unsigned — The maximum number of characters for cell names.

13.63.26 gds2_max_cellname_length=(unsigned)
Maximum length of cell names.

This property has been added in version 0.18.

Input: unsigned — The maximum number of characters for cell names. Longer cell names will be shortened.

13.63.27 [const] unsigned gds2_max_vertex_count
Get the maximum number of vertices for polygons to write.

See gds2_max_vertex_count= method for a description of the maximum vertex count.

This property has been added in version 0.18.

Return: unsigned — The maximum number of vertices for polygons to write.

13.63.28 gds2_max_vertex_count=(unsigned)
Set the maximum number of vertices for polygons to write.

This property describes the maximum number of points for polygons in GDS2 files. Polygons with more points will be split. The minimum value for this property is 4. The maximum allowed value is about 4000 or 8000, depending on the GDS2 interpretation. If gds2_multi_xy_records is true, this property is not used. Instead, the number of points is unlimited.

This property has been added in version 0.18.

Input: unsigned — The maximum number of vertices for polygons to write.

13.63.29 [const] boolean gds2_multi_xy_records
Get the property enabling multiple XY records for BOUNDARY elements.

See gds2_multi_xy_records= method for a description of this property.

This property has been added in version 0.18.

Return: true — Use of unlimited large polygons is allowed.
false — Use of unlimited large polygons is forbidden.

13.63.30 gds2_multi_xy_records=(boolean)
Use multiple XY records in BOUNDARY elements for unlimited large polygons.

This property has been added in version 0.18.

Input: true — Allows to produce unlimited large polygons at the cost of incompatible formats and disables the gds2_max_vertex_count setting.
false — For strict standard compatibility the use of unlimited large polygons is forbidden.

13.63.31 [const] double gds2_user_units
Get the user units.

See gds2_user_units= method for a description of the user units.

This property has been added in version 0.18.

Return: double The users units.

13.63.32 gds2_user_units=(double)
Set the users units to write into the GDS file.

The user units of a GDS file are rarely used and usually are set to 1 (micron). The intention of the user units is to specify the display units. **KLayout** ignores the user unit and uses microns as the display unit. The user unit must be larger than zero.

This property has been added in version 0.18.

Input: double The users units.

13.63.33 [static] SaveLayoutOptionsnew
Default constructor.

By default, the scale factor will be 1.0, the database unit is set to "same as original" and all layers are selected as well as all cells. The default format is GDS2.

13.63.34 [const] boolean no_empty_cells
Returns a flag indicating whether empty cells are not written.

Return: true Write all cells, even if they are empty.
false Write none empty cells only.

13.63.35 no_empty_cells=(boolean)
Don't write empty cells if this flag is set.

By default, all cells are written (no_empty_cells is false). This applies to empty cells which do not contain shapes for the specified layers as well as cells which are empty because they reference empty cells only.

Input: true Write none empty cells only.
false Write all cells. The default.

13.63.36 [const] integer oasis_compression_level
Get the OASIS compression level.

See oasis_compression_level= method for a description of the OASIS compression level.

Return: integer The OASIS compression level.

13.63.37 oasis_compression_level=(integer)
Set the OASIS compression level.

The OASIS compression level is an integer number between 0 and 10. 0 basically is no compression, 1 produces shape arrays in a simple fashion. 2 and higher compression levels will use a more elaborate algorithm to find shape arrays which uses 2nd and further neighbor distances. The higher the level, the higher the memory requirements and run times. Setting this property clears all format specific options for other formats such as GDS.

Input: integer The OASIS compression level.

13.63.38 [const] double scale_factor
Get the scaling factor currently set.

Return: double The current scaling factor.

13.63.39 scale_factor=(double)
Set the scaling factor for the saving .

Using a scaling factor will scale all objects accordingly. Using a scaling factor can compensate implicit scaling by an explicit database unit specification. Setting and scale factor plus an explicit database unit thus allows to transcribe a layout to a different database unit without changing the layout's physical dimensions (beside potential grid snapping effects).

Be aware that rounding effects may occur if fractional scaling factors are used which are not compliant with any implicit layout grid.

By default, no scaling is applied.

Input: double The current scaling factor.

13.63.40 select_all_cells
Select all cells to save.

This method will clear all cells specified with add_cell so far and set the select_all_cells flag.

This is the default.

13.63.41 select_all_layers
Select all layers to be saved.

This method will clear all layers selected with add_layer so far and set the select_all_cells flag.

This is the default.

13.63.42 [const] unsigned wire_mode
Specifies how to read "W" objects.

See cif_wire_mode= method for a description of this mode.

This property has been added in version 0.21.

13.64 Class Shape (version 0.21) A shape proxy .

The shape proxy is basically a pointer to a shape of different kinds. No copy of the shape is created: if the shape proxy is copied the copy still points to the original shape. If the original shape is modified or deleted, the shape proxy will also point to a modified or invalid shape. The proxy can be "null" which describes an invalid reference.

Method Overview

prop_id	Get the properties Id associated with the shape.
has_prop_id?	Check, if the shape is associated with a properties Id.
each_point	Iterate over all points of the object.
each_point_hull	Iterate over the hull contour of the object.
each_point_hole	Iterate over the points of a hole contour.
holes	Return the number of holes.
each_edge	Iterate over the edges of the object.
type	Return the type of the shape reference.
is_null?	Test if the shape proxy is a null object.
is_polygon?	Test if the shape proxy points to a polygon.
polygon	Instantiate the polygon object.
is_simple_polygon?	Test if the shape proxy points to a simple polygon.
simple_polygon	Instantiate the simple polygon object.
is_path?	Test if the shape proxy points to a path.
path_width	Obtain the path width.
round_path?	Returns true, if the path has round ends.
path_bgnext	Obtain the path's "begin" extension.
path_endext	Obtain the path's "end" extension.
path	Instantiate the path object.
is_edge?	Test if the shape proxy points to a edge.
edge	Instantiate the edge object.
is_text?	Test if the shape proxy points to a text.
text	Instantiate the text object.
text_string	Obtain the text string.
text_trans	Obtain the text transformation.
text_size	Obtain the text size.
text_font	Obtain the text's font.
is_box?	Test if the shape proxy points to a box.
box	Instantiate the box object.
is_user_object?	Test if the shape proxy points to a user object.
is_array_member?	Returns true, if the shape referenced is a member of a shape array.
array_trans	Get the array instance member transformation.
bbox	Compute the bounding box of the shape that is referenced.
!=	Inequality.
==	Equality.
to_s	Create a string showing the contents of the reference.
t_...	Various type constant.
t_null	"t_null" constant.
t_polygon	"t_polygon" constant.
t_polygon_ref	"t_polygon_ref" constant.
t_polygon_ptr_array	"t_polygon_ptr_array" constant.
t_polygon_ptr_array_member	"t_polygon_ptr_array_member" constant.
t_simple_polygon	"t_simple_polygon" constant.

t_simple_polygon_ref	"t_simple_polygon_ref" constant.
t_simple_polygon_ptr_array	"t_simple_polygon_ptr_array_member" constant.
t_simple_polygon_ptr_array_member	"t_simple_polygon_ptr_array_member" constant.
t_edge	"t_edge" constant.
t_path	"t_path" constant.
t_path_ref	"t_path_ref" constant.
t_path_ptr_array	"t_path_ptr_array" constant.
t_path_ptr_array_member	"t_path_ptr_array_member" constant.
t_box	"t_box" constant.
t_box_array	"t_box_array" constant.
t_box_array_member	"t_box_array_member" constant.
t_short_box	"t_shor"t_box" constant.
t_short_box_array	"t_shor"t_box_array" constant.
t_short_box_array_member	"t_shor"t_box_array_member" constant.
t_text	"t_text" constant.
t_text_ref	"t_text_ref" constant.
t_text_ptr_array	"t_text_ptr_array" constant.
t_text_ptr_array_member	"t_text_ptr_array_member" constant.
t_user_object	"t_user_object" constant.
assign	Assign the contents of another object to self.
dup	Creates a copy of self.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.64.1 [const] boolean !=
Inequality test.

Return: true Inequality.
false ???.

13.64.2 [const] boolean ==
Equality test.

Equality of shapes is not specified by the identity of the objects but by the identity of the pointers - both shapes must reference the same object.

Return: true Equality.
false ???.

13.64.3 [const] const ref Referencessec:Trans array_trans
Get the array instance member transformation.

This attribute is valid only if Referencessec:Shapeis-array-member? is true. The transformation returned describes the relative transformation of the array member addressed.

13.64.4 assign(Shape other)
Assign the contents of another object to self.

This method assigns the contents of another object to self. This is a deep copy that does not only copy the reference but the actual content.

13.64.5 [const] Box bbox
Compute the bounding box of the shape that is referenced.

Return: Box The bounding box.

13.64.6 [const] Box box
Instantiate the box object.

If a box is referenced, this object is instantiated by this method.

13.64.7 destroy
Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.64.8 [const] boolean destroyed
Tell, if the object was destroyed.

Return: true The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
false The object is still alive.

13.64.9 [const] Shape dup
Creates a copy of self.

Return: Shape The copy of self.

13.64.10 yield Edge each_edge
Iterate over the edges of the object.

This method applies to polygons and simple polygons.

Return: yield The array of the edges.

13.64.11 [const] yield Point each_point
Iterate over all points of the object.

This method applies to paths.

Return: yield The array of all points.

13.64.12 [const] yield Point each_point_hole(unsigned)
Iterate over the points of a hole contour.

This method applies to polygons. Simple polygons deliver an empty sequence.

Input: unsigned The hole index. Simple polygons deliver a zero value.

13.64.13 [const] yield Point each_point_hull
Iterate over the hull contour of the object.

This method applies to polygons.

Return: yield The array of the hull contour of the object.

13.64.14 [const] Edge edge
Instantiate the edge object.

If an edge is referenced, this object is instantiated by this method.

13.64.15 [const] boolean has_prop_id?
Check, if the shape is associated with a properties Id.

Return:	true	The shape is associated with a properties Id.
	false	Otherwise.

13.64.16 [const] unsigned holes
Return the number of holes.

This method applies to polygons.

Return:	unsigned	The hole index. Simple polygons deliver a zero value.

13.64.17 [const] boolean is_array_member?
Returns true, if the shape referenced is a member of a shape array.

Return:	true	The referenced shape is a member of a shape array.
	false	The referenced shape is not a member of a shape array.

13.64.18 [const] boolean is_box?
Test if the shape proxy points to a box.

Return:	true	The referenced shape points to a box.
	false	The referenced shape points not to a box.

13.64.19 [const] boolean is_edge?
Test if the shape proxy points to a edge.

Return:	true	The referenced shape points to an edge.
	false	The referenced shape points not to an edge.

13.64.20 [const] boolean is_null?
Test if the shape proxy is a null object.

Return:	true	The referenced shape is a null object.
	false	The referenced shape is a not null object.

13.64.21 [const] boolean is_path?
Test if the shape proxy points to a path.

Return: true — The referenced shape points to a path.
false — The referenced shape points not to a path.

13.64.22 [const] boolean is_polygon?
Test if the shape proxy points to a polygon.

Return: true — The referenced shape points to a polygon.
false — The referenced shape points not to a polygon.

13.64.23 [const] boolean is_simple_polygon?
Test if the shape proxy points to a simple polygon.

Return: true — The referenced shape points to a simple polygon.
false — The referenced shape points not to a simple polygon.

13.64.24 [const] boolean is_text?
Test if the shape proxy points to a text.

Return: true — The referenced shape points to a text.
false — The referenced shape points not to a text.

13.64.25 [const] boolean is_user_object?
Test if the shape proxy points to a user object.

Return: true — The referenced shape points to a user object.
false — The referenced shape points not to a user object.

13.64.26 [const] Path path
Instantiate the path object.

If a path is referenced, this object is instantiated by this method.

13.64.27 [const] integer path_bgnext
Obtain the path's "begin" extension.

Applies to paths only.

Return: integer — The "begin" extension of the path.

13.64.28 [const] integer path_endext
Obtain the path's "end" extension.

Applies to paths only.

Return: integer — The "end" extension of the path.

13.64.29 [const] integer path_width
Obtain the path width.

Applies to paths only.

Return: integer The width of the path.

13.64.30 [const] Polygon polygon
Instantiate the polygon object.

If a polygon is referenced, this object is instantiated by this method. Paths and boxes are converted to polygons.

13.64.31 [const] unsigned prop_id
Get the properties Id associated with the shape.

Return: unsigned The properties ID.

13.64.32 [const] boolean round_path?
Returns true, if the path has round ends.

Applies to paths only.

Return: true The path has round ends.
false ???.

13.64.33 [const] SimplePolygon simple_polygon
Instantiate the simple polygon object.

If a simple polygon is referenced, this object is instantiated by this method. Paths and boxes are converted to polygons.

13.64.34 [static] integer t_...
Various type constant.

13.64.34.1 [static] integer t_box – "t_box" constant.

Return: integer The "t_box" constant.

13.64.34.2 [static] integer t_box_array – "t_box_array" constant.

Return: integer The "t_box_array" constant.

13.64.34.3 [static] integer t_box_array_member – "t_box_array_member" constant.

Return: integer The "t_box_array_member" constant.

13.64.34.4 [static] integer t_edge – "t_edge" constant.

Return: integer The "t_edge" constant.

13.64.34.5 [static] integer t_null – "t_null" constant.

Return: integer The "t_null" constant.

13.64.34.6 [static] integer t_path – "t_path" constant.

Return: integer The "t_path" constant.

13.64.34.7 [static] integer t_path_ptr_array – "t_path_ptr_array" constant.

Return: integer The "t_path_ptr_array" constant.

13.64.34.8 [static] integer t_path_ptr_array_member – "t_path_ptr_array_member" constant.

Return: integer The "t_path_ptr_array_member" constant.

13.64.34.9 [static] integer t_path_ref – "t_path_ref" constant.

Return: integer The "t_path_ref" constant.

13.64.34.10 [static] integer t_polygon – "t_polygon" constant.

Return: integer The "t_polygon" constant.

13.64.34.11 [static] integer t_polygon_ptr_array – "t_polygon_ptr_array" constant.

Return: integer The "t_polygon_ptr_array" constant.

13.64.34.12 [static] integer t_polygon_ptr_array_member – "t_polygon_ptr_array_member" constant.

Return: integer The "t_polygon_ptr_array_member" constant.

13.64.34.13 [static] integer t_polygon_ref – "t_polygon_ref" constant.

Return: integer The "t_polygon_ref" constant.

13.64.34.14 [static] integer t_short_box – "t_short_box" constant.

Return: integer The "t_short_box" constant.

13.64.34.15 [static] integer t_short_box_array – "t_short_box_array" constant.

Return: integer The "t_short_box_array" constant.

13.64.34.16 [static] integer t_short_box_array_member – "t_short_box_array_member" constant.

Return: integer The "t_short_box_array_member" constant.

13.64.34.17 [static] integer t_simple_polygon – "t_simple_polygon" constant.

Return: integer The "t_simple_polygon" constant.

13.64.34.18 [static] integer t_simple_polygon_ptr_array – "t_simple_polygon_ptr_array" constant.

Return: integer The "t_simple_polygon_ptr_array" constant.

13.64.34.19 [static] integer t_simple_polygon_ptr_array_member – "t_simple_polygon_ptr_array_member" constant.

Return: integer The "t_simple_polygon_ptr_array_member" constant.

13.64.34.20 [static] integer t_simple_polygon_ref – "t_simple_polygon_ref" constant.

Return: integer The "t_simple_polygon_ref" constant

13.64.34.21 [static] integer t_text – "t_text" constant.

Return: integer The "t_text" constant.

13.64.34.22 [static] integer t_text_ptr_array – "t_text_ptr_array" constant.

Return: integer The "t_text_ptr_array" constant.

13.64.34.23 [static] integer t_text_ptr_array_member – "t_text_ptr_array_member" constant.

Return: integer The "t_text_ptr_array_member" constant.

13.64.34.24 [static] integer t_text_ref – "t_text_ref" constant.

Return: integer The "t_text_ref" constant.

13.64.34.25 [static] integer t_user_object – "t_user_object" constant.

Return: integer The "t_user_object" constant.

13.64.35 [const] Text text
Instantiate the text object.

If a text is referenced, this object is instantiated by this method.

13.64.36 [const] integer text_font
Obtain the text's font.

Applies to texts only. Will throw an exception if not a text.

Return: integer The font of the text object.

13.64.37 [const] integer text_size
Obtain the text size.

Applies to texts only. Will throw an exception if not a text.

Return: integer The size of the text object.

13.64.38 [const] string text_string
Obtain the text string.

Applies to texts only. Will throw an exception if not a text.

Return: string The string of the text object.

13.64.39 [const] Trans text_trans
Obtain the text transformation.

Applies to texts only. Will throw an exception if not a text.

Return: Trans The text transformation.

13.64.40 [const] string to_s
Create a string showing the contents of the reference.

This method has been introduced with version 0.16.

Return: string A string showing the contents of the reference.

13.64.41 [const] integer type
Return the type of the shape reference.

Return: integer The returned values are the "t_..." constants available through the corresponding class members.

13.65 Class ShapeProcessor (version 0.21)
The shape processor (boolean, sizing, merge on shapes).

The shape processor implements the boolean and edge set operations (size, merge). Because the shape processor might allocate resources which can be reused in later operations, it is implemented as an object that can be used several times. The shape processor is similar to the EdgeProcessor. The latter is specialized on handling polygons and edges directly.

Method Overview

merge	Merge the given shapes from a layout into a shapes container.
boolean	Boolean operation on shapes from layouts.
size	Sizing operation on shapes from layouts.
size	Sizing operation on shapes from layouts.
merge	Merge the given shapes.
merge_to_polygon	Merge the given shapes.
merge	Merge the given shapes.
merge_to_polygon	Merge the given shapes.
boolean	Boolean operation on two given shape sets into an edge set.
boolean_to_polygon	Boolean operation on two given shape sets into a polygon set.
boolean	Boolean operation on two given shape sets into an edge set.
boolean_to_polygon	Boolean operation on two given shape sets into a polygon set.
size	Size the given shapes.
size	Size the given shapes.
size_to_polygon	Size the given shapes.
size_to_polygon	Size the given shapes.
size	Size the given shapes.
size	Size the given shapes.
size_to_polygon	Size the given shapes.
size_to_polygon	Size the given shapes.
assign	Assign the contents of another object to self.
dup	Creates a copy of self.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.65.1 assign(ShapeProcessor other)
Assign the contents of another object to self.

This method assigns the contents of another object to self. This is a deep copy that does not only copy the reference but the actual content.

13.65.2 boolean(Layout layout_a, Cell cell_a, unsigned layer_a, Layout layout_b, Cell cell_b, unsigned layer_b, ref Shapes out, mode, hierarchical, resolve_holes, coherence)
Boolean operation on shapes from layouts.

See the EdgeProcessor for a description of the boolean operations. This implementation takes shapes from layout cells (optionally all in hierarchy) and produces new shapes in a shapes container.

Input:	layout_a	The layout from which to take the shapes for input A.
	cell_a	The cell (in "layout") from which to take the shapes for input A.
	layer_a	The layer (in "layout") from which to take the shapes for input A.
	layout_b	The layout from which to take the shapes for input B.
	cell_b	The cell (in "layout") from which to take the shapes for input B.
	layer_b	The layer (in "layout") from which to take the shapes for input B.
	out	The shapes container where to put the shapes into (is cleared before).
	mode	The boolean operation (see EdgeProcessor).
	hierarchical	If true: Collect shapes from sub cells as well.
	resolve_holes	If true: Holes should be resolved into the hull.
	coherence	If true: Minimum polygons should be created for touching corners.

13.65.3 Edge[] boolean(Shape in_a[], CplxTrans trans_a[], Shape in_b[], CplxTrans trans_b[], mode)

Boolean operation on two given shape sets into an edge set with transformation.

See the EdgeProcessor for a description of the boolean operations. This implementation takes shapes rather than polygons for input and produces an edge set.

Input:	in_a[]	The set of shapes to use for input A.
	trans_a[]	A set of transformations to apply before the shapes from input A are used.
	in_b[]	The set of shapes to use for input B.
	trans_b[]	A set of transformations to apply before the shapes from input B are used.
	mode	The boolean operation (see EdgeProcessor).
Return:	Edge[]	The produced edge set.

13.65.4 Edge[] boolean(Shape in_a[], Shape in_b[], mode)

Boolean operation on two given shape sets into an edge set.

See the EdgeProcessor for a description of the boolean operations. This implementation takes shapes rather than polygons for input and produces an edge set.

This version does not allow to specify a transformation for each shape (unity is assumed).

Input:	in_a[]	The set of shapes to use for input A.
	in_b[]	The set of shapes to use for input B.
	mode	The boolean operation (see EdgeProcessor).
Return:	Edge[]	The produced edge set.

13.65.5 Polygon[] boolean_to_polygon(Shape in_a[], CplxTrans trans_a[], Shape in_b[], CplxTrans trans_b[], mode)

Boolean operation on two given shape sets into a polygon set with transformation.

See the EdgeProcessor for a description of the boolean operations. This implementation takes shapes rather than polygons for input and produces a polygon set.

Input:	in_a[]	The set of shapes to use for input A.
	trans_a[]	A set of transformations to apply before the shapes from input A are used.
	in_b[]	The set of shapes to use for input B.
	trans_b[]	A set of transformations to apply before the shapes from input B are used.
	mode	The boolean operation (see EdgeProcessor).
Return:	Polygon[]	The produced polygon set.

13.65.6 Polygon[] boolean_to_polygon(Shape in_a[], Shape in_b[], mode)
Boolean operation on two given shape sets into an edge set.

See the EdgeProcessor for a description of the boolean operations. This implementation takes shapes rather than polygons for input and produces an edge set.

This version does not allow to specify a transformation for each shape (unity is assumed).

Input:	in_a[]	The set of shapes to use for input A.
	in_b[]	The set of shapes to use for input B.
	mode	The boolean operation (see EdgeProcessor).
Return:	Polygon[]	The produced polygon set.

13.65.7 destroy
Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.65.8 [const] boolean destroyed
Tell, if the object was destroyed.

Return:	true	The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
	false	The object is still alive.

13.65.9 [const] ShapeProcessor dup
Creates a copy of self.

Return:	ShapeProcessor	The copy of self.

13.65.10 Edge[] merge(Shape in[], CplxTrans trans[], unsigned min_wc)
Merge the given shapes.

See the EdgeProcessor for a description of the boolean operations. This implementation takes shapes rather than polygons for input and produces an edge set.

Input:	in[]	The set of shapes to merge.
	trans[]	A set of transformations to apply before the shapes are used.
	min_wc	The minimum wrap count for output (0: all polygons, 1: at least two overlapping).
Return:	Edge[]	The produced edge set.

13.65.11 Edge[] merge(Shape in[], unsigned min_wc)
Merge the given shapes.

See the EdgeProcessor for a description of the boolean operations. This implementation takes shapes rather than polygons for input and produces an edge set.

This version does not allow to specify a transformation for each shape (unity is assumed).

Input:	in[]	The set of shapes to merge.
	min_wc	The minimum wrap count for output (0: all polygons, 1: at least two overlapping).
Return:	Edge[]	The produced edge set.

13.65.12 merge(Layout layout, Cell cell, unsigned layer, ref Shapes out, hierarchical, unsigned min_wc, resolve_holes, coherence)
Boolean operation on shapes from layouts.

See the EdgeProcessor for a description of the boolean operations. This implementation takes shapes from layout cells (optionally all in hierarchy) and produces new shapes in a shapes container.

Input:	layout	The layout from which to take the shapes for input A.
	cell	The cell (in "layout") from which to take the shapes for input A.
	layer	The layer (in "layout") from which to take the shapes for input A.
	out	The shapes container where to put the shapes into (is cleared before).
	hierarchical	If true: Collect shapes from sub cells as well.
	min_wc	The minimum wrap count for output (0: all polygons, 1: at least two overlapping).
	resolve_holes	If true: Holes should be resolved into the hull.
	coherence	If true: Minimum polygons should be created for touching corners.

13.65.13 Polygon[] merge_to_polygon(Shape in[], CplxTrans trans[], unsigned min_wc, resolve_holes, coherence)
Merge the given shapes with transformation.

See the EdgeProcessor for a description of the boolean operations. This implementation takes shapes rather than polygons for input and produces a polygon set.

Input:	in[]	The set of shapes to merge.
	trans[]	A set of transformations to apply before the shapes are used.
	min_wc	The minimum wrap count for output (0: all polygons, 1: at least two overlapping).
	resolve_holes	If true: Holes should be resolved into the hull.
	coherence	If true: Minimum polygons should be created for touching corners.
Return:	Polygon[]	The produced polygon set.

13.65.14 Polygon[] merge_to_polygon(Shape in[], CplxTrans trans[], unsigned min_wc, resolve_holes, coherence)
Merge the given shapes.

See the EdgeProcessor for a description of the boolean operations. This implementation takes shapes rather than polygons for input and produces a polygon set.

This version does not allow to specify a transformation for each shape (unity is assumed).

Input:	in[]	The set of shapes to merge.
	min_wc	The minimum wrap count for output (0: all polygons, 1: at least two overlapping).
	resolve_holes	If true: Holes should be resolved into the hull.
	coherence	If true: Minimum polygons should be created for touching corners.
Return:	Polygon[]	The produced polygon set.

13.65.15 size(Layout layout, Cell cell, unsigned layer, ref Shapes out, dx, dy, unsigned mode, hierarchical, resolve_holes, coherence) Sizing operation on shapes from layouts (anisotropic).

See the EdgeProcessor for a description of the boolean operations. This implementation takes shapes from a layout cell (optionally all in hierarchy) and produces new shapes in a shapes container.

Input:	layout	The layout from which to take the shapes for input A.
	cell	The cell (in "layout") from which to take the shapes for input A.
	layer	The layer (in "layout") from which to take the shapes for input A.
	out	The shapes container where to put the shapes into (is cleared before).
	dx	The sizing value in x-direction (see EdgeProcessor).
	dy	The sizing value in y-direction (see EdgeProcessor).
	mode	The boolean operation (see EdgeProcessor).
	hierarchical	If true: Collect shapes from sub cells as well.
	resolve_holes	If true: Holes should be resolved into the hull.
	coherence	If true: Minimum polygons should be created for touching corners.

13.65.16 size(Layout layout, Cell cell, unsigned layer, ref Shapes out, d, unsigned mode, hierarchical, resolve_holes, coherence) Sizing operation on shapes from layouts (isotropic).

See the EdgeProcessor for a description of the boolean operations. This implementation takes This implementation takes shapes from a layout cell (optionally all in hierarchy) and produces new shapes in a shapes container. This is the isotropic version which does not allow to specify different sizing values in x and y-direction.

Input:	layout	The layout from which to take the shapes for input A.
	cell	The cell (in "layout") from which to take the shapes for input A.
	layer	The layer (in "layout") from which to take the shapes for input A.
	out	The shapes container where to put the shapes into (is cleared before).
	d	The sizing value in x-direction (see EdgeProcessor).
	mode	The boolean operation (see EdgeProcessor).
	hierarchical	If true: Collect shapes from sub cells as well.
	resolve_holes	If true: Holes should be resolved into the hull.
	coherence	If true: Minimum polygons should be created for touching corners.

13.65.17 Edge[] size(Shape in[], CplxTrans trans[], dx, dy, unsigned mode) Size the given shapes with transformation (anisotropic).

See the EdgeProcessor for a description of the boolean operations. This implementation takes shapes rather than polygons for input and produces an edge set.

Input:	in[]	The set of shapes to size.
	trans[]	A set of transformations to apply before the shapes are used.
	dx	The sizing value in x-direction (see EdgeProcessor).
	dy	The sizing value in y-direction (see EdgeProcessor).
	mode	The boolean operation (see EdgeProcessor).
Return:	Edge[]	The produced edge set.

13.65.18 Edge[] size(Shape in[], dx, dy, unsigned mode)
Size the given shapes (anisotropic).

See the EdgeProcessor for a description of the boolean operations. This implementation takes shapes rather than polygons for input and produces an edge set.

This version does not allow to specify a transformation for each shape (unity is assumed).

Input:	in[]	The set of shapes to size.
	dx	The sizing value in x-direction (see EdgeProcessor).
	dy	The sizing value in y-direction (see EdgeProcessor).
	mode	The boolean operation (see EdgeProcessor).
Return:	Edge[]	The produced edge set.

13.65.19 Edge[] size(Shape in[], CplxTrans trans[], d, unsigned mode)
Size the given shapes with transformation (isotropic).

See the EdgeProcessor for a description of the boolean operations. This implementation takes shapes rather than polygons for input and produces an edge set.

Input:	in[]	The set of shapes to size.
	trans[]	A set of transformations to apply before the shapes are used.
	d	The sizing value (see EdgeProcessor).
	mode	The boolean operation (see EdgeProcessor).
Return:	Edge[]	The produced edge set.

13.65.20 Edge[] size(Shape in[], d, unsigned mode)
Size the given shapes (isotropic).

See the EdgeProcessor for a description of the boolean operations. This implementation takes shapes rather than polygons for input and produces an edge set. This is isotropic version that does not allow to specify different values in x and y direction.

This version does not allow to specify a transformation for each shape (unity is assumed).

Input:	in[]	The set of shapes to size.
	d	The sizing value (see EdgeProcessor).
	mode	The boolean operation (see EdgeProcessor).
Return:	Edge[]	The produced edge set.

13.65.21 Polygon size_to_polygon(Shape in[], CplxTrans trans[], dx, dy, unsigned mode, resolve_holes, coherence)
Size the given shapes with transformation (anisotropic).

See the EdgeProcessor for a description of the boolean operations. This implementation takes shapes rather than polygons for input and produces an polygon set.

Input:	in[]	The set of shapes to size.
	trans[]	A set of transformations to apply before the shapes are used.
	dx	The sizing value in x-direction (see EdgeProcessor).
	dy	The sizing value in y-direction (see EdgeProcessor).
	mode	The boolean operation (see EdgeProcessor).
	resolve_holes	If true: Holes should be resolved into the hull.
	coherence	If true: Minimum polygons should be created for touching corners.
Return:	Polygon[]	The produced polygon set.

13.65.22 Polygon[] size_to_polygon(Shape in[], dx, dy, unsigned mode, resolve_holes, coherence)
Size the given shapes (anisotropic).

See the EdgeProcessor for a description of the boolean operations. This implementation takes shapes rather than polygons for input and produces an polygon set.

This version does not allow to specify a transformation for each shape (unity is assumed).

Input:	in[]	The set of shapes to size.
	dx	The sizing value in x-direction (see EdgeProcessor).
	dy	The sizing value in y-direction (see EdgeProcessor).
	mode	The boolean operation (see EdgeProcessor).
	resolve_holes	If true: Holes should be resolved into the hull.
	coherence	If true: Minimum polygons should be created for touching corners.
Return:	Polygon[]	The produced polygon set.

13.65.23 Polygon[] size_to_polygon(Shape in[], CplxTrans trans[], d, unsigned mode, resolve_holes, coherence)
Size the given shapes with transformation (isotropic).

See the EdgeProcessor for a description of the boolean operations. This implementation takes shapes rather than polygons for input and produces an polygon set.

Input:	in[]	The set of shapes to size.
	trans[]	A set of transformations to apply before the shapes are used.
	d	The sizing value (see EdgeProcessor).
	mode	The boolean operation (see EdgeProcessor).
	resolve_holes	If true: Holes should be resolved into the hull.
	coherence	If true: Minimum polygons should be created for touching corners.
Return:	Polygon[]	The produced polygon set.

13.65.24 Polygon[] size_to_polygon(Shape in[], d, unsigned mode, resolve_holes, coherence)
Size the given shapes (isotropic).

See the EdgeProcessor for a description of the boolean operations. This implementation takes shapes rather than polygons for input and produces an polygon set. This is isotropic version that does not allow to specify different values in x and y direction.

This version does not allow to specify a transformation for each shape (unity is assumed).

Input:	in[]	The set of shapes to size.
	d	The sizing value (see EdgeProcessor).
	mode	The boolean operation (see EdgeProcessor).
	resolve_holes	If true: Holes should be resolved into the hull.
	coherence	If true: Minimum polygons should be created for touching corners.
Return:	Polygon[]	The produced polygon set.

13.66 Class Shapes (version 0.21) A collection of shapes.

A shapes collection is a collection of geometrical objects, such as polygons, boxes, paths, edges or text objects.

Method Overview

insert	Insert a shape from a shape reference into the shapes list.
transform	Transform the shape given by the reference with the given transformation.
transform	Transform the shape given by the reference with the given complex transformation.
replace	Replace the given shape with a box.
replace	Replace the given shape with a path.
replace	Replace the given shape with an edge object.
replace	Replace the given shape with a text object.
replace	Replace the given shape with a simple polygon.
replace	Replace the given shape with a polygon.
insert	Insert a box into the shapes list.
insert	Insert a path into the shapes list.
insert	Insert a edge into the shapes list.
insert	Insert a text into the shapes list.
insert	Insert a simple polygon into the shapes list.
insert	Insert a polygon into the shapes list.
insert	Insert a box with properties into the shapes list.
insert	Insert a path with properties into the shapes list.
insert	Insert a edge with properties into the shapes list.
insert	Insert a text with properties into the shapes list.
insert	Insert a simple polygon with properties into the shapes list.
insert	Insert a polygon with properties into the shapes list.
each	Get all shapes.
each	Get all shapes.
each_touching	Get all shapes that touch the search box (region).
each_touching	Get all shapes that touch the search box (region).
each_overlapping	Get all shapes that overlap the search box (region).
each_overlapping	Get all shapes that overlap the search box (region).
erase	Erase the shape pointed to by the given Shape object.
is_valid?	Test if the given Shape object is still pointing to a valid object.
is_empty?	Returns a value indicating whether the shapes container is empty.
clear	Clear the shape container.
size	Report the number of shapes in this container.
replace_prop_id	Replace (or install) the properties of a shape.
s_all	“s_all” constant.
s_all_with_properties	“s_all_with_properties” constant.
s_polygons	“s_polygons” constant.
s_boxes	“s_boxes” constant.
s_edges	“s_edges” constant.
s_paths	“s_paths” constant.
s_texts	“s_texts” constant.
s_user_objects	“s_user_objects” constant.
s_properties	“s_properties” constant.
assign	Assign the contents of another object to self.
dup	Creates a copy of self.

destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.66.1 assign(Shapes other)
Assign the contents of another object to self.

This method assigns the contents of another object to self. This is a deep copy that does not only copy the reference but the actual content.

13.66.2 clear
Clear the shape container.

This method can only be used in editable mode.

This method has been introduced in version 0.16.

13.66.3 destroy
Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.66.4 [const] boolean destroyed
Tell, if the object was destroyed.

Return:	true	The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
	false	The object is still alive.

13.66.5 [const] Shapes dup
Creates a copy of self.

Return:	Shapes	The copy of self.

13.66.6 [const] yield Shape each(unsigned)
Get all shapes.

Input:	unsigned	An "or"-ed combination of the s_... constants.
Return:	Shape	An array of shapes.

13.66.7 [const] yield Shape each
Get all shapes.

This call is equivalent to each(s_all).

This convenience method has been introduced in version 0.16.

Return:	Shape	An array of shapes.

13.66.8 [const] yield Shape each_overlapping(Box region)
Get all shapes that overlap the search box (region).

This call is equivalent to each_overlapping(s_all,region).

This convenience method has been introduced in version 0.16.

Input: region — The rectangular search region.
Return: Shape — An array of shapes.

13.66.9 [const] yield Shape each_overlapping(unsigned, Box region)
Get all shapes that overlap the search box (region).

This convenience method has been introduced in version 0.16.

Input: unsigned — An "or"-ed combination of the s_... constants.
region — The rectangular search region.
Return: Shape — An array of shapes.

13.66.10 [const] yield Shape each_touching(Box region)
Get all shapes that overlap the search box (region).

This call is equivalent to each_overlapping(s_all,region).

This convenience method has been introduced in version 0.16.

Input: region — The rectangular search region.
Return: Shape — An array of shapes.

13.66.11 [const] yield Shape each_touching(unsigned, Box region)
Get all shapes that overlap the search box (region).

This convenience method has been introduced in version 0.16.

Input: unsigned — An "or"-ed combination of the s_... constants.
region — The rectangular search region.
Return: Shape — An array of shapes.

13.66.12 erase(Shape shape)
Erase the shape pointed to by the given Shape object.

This method can only be used in editable mode.

This method has been introduced in version 0.16.

Input: shape — The shape which to destroy. Erasing a shape will invalidate the shape reference. Access to this reference may then render invalid results.

13.66.13 namerefsec:Shape insert(namerefsec:Shape shape)
Insert a shape from a shape reference into the shapes list.

This method has been introduced in version 0.16.

Input: shape — The shape object.
Return: Shape — A reference to the new shape (a Shape object).

13.66.14 Shape insert(Box box)
Insert a box into the shapes list.

A synonym for: Shape insert_box(Box box).

Starting with version 0.16, this method returns a reference to the newly created shape.

Input:	box	The box object.
Return:	Shape	A reference to the new shape (a Shape object).

13.66.15 Shape insert(Box box, unsigned)
Insert a box with properties into the shapes list.

A synonym for: Shape insert_box_with_properties(Box box, unsigned).

Starting with version 0.16, this method returns a reference to the newly created shape.

Input:	box	The box object with properties.
	unsigned	The property ID which must be obtained from the Layout object's property_id method. This associates a property set with a property Id.
Return:	Shape	A reference to the new shape (a Shape object).

13.66.16 Shape insert(Edge edge)
Insert an edges into the shapes list.

A synonym for: Shape insert_edge(Edge edge, unsigned).

Starting with version 0.16, this method returns a reference to the newly created shape.

Input:	edge	The edge object with properties.
Return:	Shape	A reference to the new shape (a Shape object).

13.66.17 Shape insert(Edge edge, unsigned)
Insert an edge with properties into the shapes list.

A synonym for: Shape insert_edge_with_properties(Edge edge, unsigned).

Starting with version 0.16, this method returns a reference to the newly created shape.

Input:	edge	The edge object with properties.
	unsigned	The property ID which must be obtained from the Layout object's property_id method. This associates a property set with a property Id.
Return:	Shape	A reference to the new shape (a Shape object).

13.66.18 Shape insert(Path path)
Insert a path into the shapes list.

A synonym for: Shape insert_path(Path path).

Starting with version 0.16, this method returns a reference to the newly created shape.

Input:	path	The path object with properties.
Return:	Shape	A reference to the new shape (a Shape object).

13.66.19 Shape insert(Path path, unsigned)
Insert a path with properties into the shapes list.

A synonym for: Shape insert_path_with_properties(Path path, unsigned).

Starting with version 0.16, this method returns a reference to the newly created shape.

Input:	path	The path object with properties.
	unsigned	The property ID which must be obtained from the Layout object's property_id method. This associates a property set with a property Id.
Return:	Shape	A reference to the new shape (a Shape object).

13.66.20 Shape insert(Polygon polygon)
Insert a polygon into the shapes list.

A synonym for: Shape insert_polygon(Polygon polygon).

Starting with version 0.16, this method returns a reference to the newly created shape.

Input:	polygon	The polygon object.
Return:	Shape	A reference to the new shape (a Shape object).

13.66.21 Shape insert(Polygon polygon, unsigned)
Insert a polygon with properties into the shapes list.

A synonym for: Shape insert_polygon_with_properties(Polygon polygon, unsigned).

Starting with version 0.16, this method returns a reference to the newly created shape.

Input:	polygon	The polygon object with properties.
	unsigned	The property ID which must be obtained from the Layout object's property_id method. This associates a property set with a property Id.
Return:	Shape	A reference to the new shape (a Shape object).

13.66.22 Shape insert(SimplePolygon simple_polygon)
Insert a simple polygon into the shapes list.

A synonym for: Shape insert_simple_polygon(SimplePolygon simple_polygon).

Starting with version 0.16, this method returns a reference to the newly created shape.

Input:	simple_polygon	The simple polygon object with properties.
Return:	Shape	A reference to the new shape (a Shape object).

13.66.23 Shape insert(SimplePolygon simple_polygon, unsigned)
Insert a simple polygon with properties into the shapes list.

A synonym for: Shape insert_simple_polygon_with_properties(SimplePolygon simple_polygon, unsigned).

Starting with version 0.16, this method returns a reference to the newly created shape.

Input:	simple_polygon	The simple polygon object with properties.
	unsigned	The property ID which must be obtained from the Layout object's property_id method. This associates a property set with a property Id.
Return:	Shape	A reference to the new shape (a Shape object).

13.66.24 Shape insert(Text text)
Insert a text into the shapes list.

A synonym for: Shape insert_text (Text text).

Starting with version 0.16, this method returns a reference to the newly created shape.

Input:	text	The text object.
Return:	Shape	A reference to the new shape (a Shape object).

13.66.25 Shape insert(Text text, unsigned)
Insert a text with properties into the shapes list.

A synonym for: Shape insert_text_with_properties (Text text, unsigned).

Starting with version 0.16, this method returns a reference to the newly created shape.

Input:	text	The text object.
	unsigned	The property ID which must be obtained from the Layout object's property_id method. This associates a property set with a property Id.
Return:	Shape	A reference to the new shape (a Shape object).

13.66.26 [const] boolean is_empty?
Returns a value indicating whether the shapes container is empty.

This method has been introduced in version 0.20.

Return:	true	An empty object.
	false	A none empty object.

13.66.27 [const] boolean is_valid?(Shape shape)
Test if the given Shape object is still pointing to a valid object.

This method has been introduced in version 0.16.

Return:	true	A valid object. The shape represented by the given reference has been deleted, but another shape has been inserted already that occupies the original shape's position.
	false	The shape represented by the given reference has been deleted.

13.66.28 Shape replace(Shape shape, Box box)
Replace the given shape with a box.

This method replaces the given shape with the object specified. It does not change the property Id. To change the property Id, use the replace_prop_id method. To replace a shape and discard the property Id, erase the shape and insert a new shape.

This method is permitted in editable mode only.

This method has been introduced with version 0.16.

Input:	shape	The given shape to replace.
	box	The specified object.
Return:	Shape	A reference to the new shape (a Shape object).

13.66.29 Shape replace(Shape shape, Edge edge)
Replace the given shape with an edge object.

This method replaces the given shape with the object specified. It does not change the property Id. To change the property Id, use the replace_prop_id method. To replace a shape and discard the property Id, erase the shape and insert a new shape.

This method is permitted in editable mode only.

This method has been introduced with version 0.16.

Input:	shape	The given shape to replace.
	edge	The specified object.
Return:	Shape	A reference to the new shape (a Shape object).

13.66.30 Shape replace(Shape shape, Path path)
Replace the given shape with a path.

This method replaces the given shape with the object specified. It does not change the property Id. To change the property Id, use the replace_prop_id method. To replace a shape and discard the property Id, erase the shape and insert a new shape.

This method is permitted in editable mode only.

This method has been introduced with version 0.16.

Input:	shape	The given shape to replace.
	path	The specified object.
Return:	Shape	A reference to the new shape (a Shape object).

13.66.31 Shape replace(Shape shape, Polygon polygon)
Replace the given shape with a polygon.

This method replaces the given shape with the object specified. It does not change the property Id. To change the property Id, use the replace_prop_id method. To replace a shape and discard the property Id, erase the shape and insert a new shape.

This method is permitted in editable mode only.

This method has been introduced with version 0.16.

Input:	shape	The given shape to replace.
	polygon	The specified object.
Return:	Shape	A reference to the new shape (a Shape object).

13.66.32 Shape replace(Shape shape, SimplePolygon simple_polygon)
.

This method replaces the given shape with the object specified. It does not change the property Id. To change the property Id, use the replace_prop_id method. To replace a shape and discard the property Id, erase the shape and insert a new shape.

This method is permitted in editable mode only.

This method has been introduced with version 0.16.

Input:	shape	The given shape to replace.
	simple_polygon	The specified object.
Return:	Shape	A reference to the new shape (a Shape object).

13.66.33 Shape replace(Shape shape, Text text)
Replace the given shape with a text object.

This method replaces the given shape with the object specified. It does not change the property Id. To change the property Id, use the replace_prop_id method. To replace a shape and discard the property Id, erase the shape and insert a new shape.

This method is permitted in editable mode only.

This method has been introduced with version 0.16.

Input:	shape	The given shape to replace.
	text	The specified object.
Return:	Shape	A reference to the new shape (a Shape object).

13.66.34 Shape replace_prop_id(Shape shape, unsigned)
Replace (or install) the properties of a shape.

This method changes the properties Id of the given shape or install a properties Id on that shape if it does not have one yet. The property Id must be obtained from the Layout object's properties_id method which associates a property set with a property Id. This method will potentially invalidate the shape reference passed to it. Use the reference returned for future references. This method is permitted in editable mode only.

This method has been introduced with version 0.16.

Input:	shape	The given shape to replace.
	unsigned	The properties Id to change or install.
Return:	Shape	A reference to the new shape (a Shape object).

13.66.35 [static] unsigned s_all
"s_all" constant.

Return: unsigned .

13.66.36 [static] unsigned s_all_with_properties
"s_all_with_properties" constant.

Return: unsigned .

13.66.37 [static] unsigned s_boxes
"s_boxes" constant.

Return: unsigned .

13.66.38 [static] unsigned s_edges
"s_edges" constant.

Return: unsigned .

13.66.39 [static] unsigned s_paths
"s_paths" constant.

Return: unsigned .

13.66.40 [static] unsigned s_polygons
"s_polygons" constant.

Return: unsigned .

13.66.41 [static] unsigned s_properties
"s_properties" constant.

Return: unsigned .

13.66.42 [static] unsigned s_texts
"s_texts" constant.

Return: unsigned .

13.66.43 [static] unsigned s_user_objects
"s_user_objects" constant.

Return: unsigned .

13.66.44 [const] unsigned size
Report the number of shapes in this container.

This method was introduced in version 0.16

Return: unsigned The number of shapes in this container.

13.66.45 Shape transform(Shape shape, Trans t)
Transform the shape given by the reference with the given transformation.

The original shape may be deleted and re-inserted by this method. Therefore, a new reference is returned.

This method is permitted in editable mode only.

This method has been introduced in version 0.16.

Input: shape The given shape to replace.
t The given transformation to perform.
Return: Shape A reference to the new shape (a Shape object).

13.66.46 Shape transform(Shape shape, CplxTrans t)
Transform the shape given by the reference with the given complex transformation.

The original shape may be deleted and re-inserted by this method. Therefore, a new reference is returned.

This method is permitted in editable mode only.

This method has been introduced in version 0.16.

Input: shape The given shape to replace.
t The given complex transformation to perform.
Return: Shape A reference to the new shape (a Shape object).

13.67 Class SimplePolygon (version 0.21) A polygon class with integer coordinates.

A simple polygon consists of an outer hull. The hull contour consists of several points. The point list is normalized such that the leftmost, lowest point is the first one. The orientation is normalized such that the orientation of the hull contour is clockwise.

It is in no way checked that the contours are not over-lapping. This must be ensured by the user of the object when filling the contours.

Method Overview

new	Default constructor: creates an empty (invalid) polygon.
new	Constructor given the points of the simple polygon.
new	Constructor converting a box to a simple polygon.
==	Equality test.
!=	Inequality test.
points=	Set the points of the simple polygon.
point	Get a specific point of the contour@args p.
points	Get the number of points.
each_point	Iterate over the points that make up the simple polygon.
each_edge	Iterate over the edges that make up the simple polygon.
inside	Test, if the given point is inside the simple polygon.
compress	Compress the simple polygon.
move	Moves the simple polygon.
moved	Returns the moved simple polygon.
transformed	Transform the simple polygon.
transformed_cplx	Transform the simple polygon with a complex transformation.
transformed_cplx	Transform the simple polygon with a complex transformation.
to_s	Convert to a string.
area	The area of the polygon.
bbox	Return the bounding box of the polygon.
from_dpoly	Construct an integer coordinate polygon from a floating-point coordinate one.
assign	Assign the contents of another object to self.
dup	Creates a copy of self.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.67.1 [const] boolean !=(SimplePolygon p) Inequality test.

Input:	p	The object to compare against.
Return:	true	Inequality.
	false	???.

13.67.2 [const] boolean ==(SimplePolygon p) Equality test.

Input:	p	The object to compare against.
Return:	true	The polygons are equal.
	false	???.

13.67.3 [const] long area
The area of the simple polygon.

The area is correct only if the polygon is not self-overlapping and oriented clockwise.

Return: long The area of the polygon.

13.67.4 assign(SimplePolygon other)
Assign the contents of another object to self.

This method assigns the contents of another object to self. This is a deep copy that does not only copy the reference but the actual content.

13.67.5 [const] const refBox bbox
Return the bounding box of the simple polygon.

Return: ??? The bounding box of the simple polygon.

13.67.6 compress(boolean)
Compress the simple polygon.

Removes redundant points from the polygon, such as points being on a line formed by two other points.

Input: true Additionally removes points if the two adjacent edges form a spike.
false Basic behavior.

13.67.7 destroy
Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.67.8 [const] boolean destroyed
Tell, if the object was destroyed.

Return: true The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
false The object is still alive.

13.67.9 [const] SimplePolygon dup
Creates a copy of self.

Return: SimplePolygon The copy of self.

13.67.10 yield Edge each_edge
Iterate over the edges that make up the simple polygon.

Return: yield An array of the edges that make up the simple polygon.

13.67.11 [const] yield Point each_point(unsigned)
Iterate over the points that make up the simple polygon.

Return: yield — An array of the points that make up the simple polygon.

13.67.12 [static] SimplePolygon from_dpoly(DSimplePolygon double_poly)
Construct a floating-point coordinate polygon from an integer coordinate one.

This method has been added in version 0.15.

Input: double_poly — The given integer coordinate simple polygon.
Return: SimplePolygon — The resulting floating-point coordinate simple polygon.

13.67.13 [const] boolean inside(Point p)
Test, if the given point is inside the simple polygon.

This tests works well only if the polygon is not self-overlapping and oriented clockwise.

Input: true — The given point is inside the polygon.
false — The given point is outside the polygon.

13.67.14 ref SimplePolygon move(Point p)
Moves the polygon.

Moves the polygon by the given offset and returns the reference of the moved polygon. The polygon is overwritten.

Input: p — The distance to move the polygon.
Return: ref — The reference of the moved polygon.

13.67.15 [const] SimplePolygon moved(Point p)
Returns the moved polygon.

Moves the polygon by the given offset and returns the moved simple polygon. The polygon is not modified.

Input: p — The distance to move the polygon.
Return: SimplePolygon — The moved polygon.

13.67.16 [static] SimplePolygon new
Default constructor: creates an empty (invalid) polygon.

13.67.17 [static] SimplePolygon new(Point p[])
Constructor given the points of the simple polygon.

A synonym for: [static] SimplePolygon new_p(Point p[]).

Input: p[] — An array of points to insert as a new simple polygon.

13.67.18 [static] SimplePolygon new(Box box)
Constructor converting a box to a polygon.

A synonym for: [static] SimplePolygon new_b(Box box).

Input: box — The box to convert to a polygon.

13.67.19 Point point(unsigned p)
Get a specific point of a contour@args p.

This method was introduced in version 0.18.

Input: unsigned p — The index of the point to get. If the index of the point is not valid, a default value is returned.

Return: Point — The specific hole point.

13.67.20 unsigned points
Get the number of points.

This method was introduced in version 0.18.

Return: unsigned — The number of points.

13.67.21 points=(Point p[])
Set the points of the simple polygon.

Input: p[] — An array of points to assign to the simple polygon.

13.67.22 string to_s
Convert to a string.

Return: string — The string representing the simple polygon.

13.67.23 [const] SimplePolygon transformed(Trans t)
Transform the simple polygon.

Transforms the simple polygon with the given transformation. Does not modify the polygon but returns the transformed polygon.

Input: t — The transformation to apply.

Return: SimplePolygon — The transformed simple polygon.

13.67.24 [const] SimplePolygon transformed_cplx(CplxTrans t)
Transform the simple polygon.

Transforms the simple polygon with the given transformation. Does not modify the polygon but returns the transformed polygon.

Input: t — The transformation to apply.

Return: SimplePolygon — The transformed simple polygon.

13.67.25 [const] SimplePolygon transformed_cplx(ICplxTrans t) Transform the simple polygon.

Transforms the simple polygon with the given transformation. Does not modify the polygon but returns the transformed polygon.

This method was introduced in version 0.18.

Input:	t	The transformation to apply.
Return:	SimplePolygon	The transformed simple polygon (in this case an integer coordinate object).

13.68 Class StringListValue (version 0.21) Encapsulate a string list.

This class is provided as a return value of FileDialog. By using an object rather than a pure string list, an object with has_value? = false can be returned indicating that the "Cancel" button was pressed.

Method Overview

has_value?	True, if a value is present.
value	Get the actual value (a list of strings)
assign	Assign the contents of another object to self.
dup	Creates a copy of self.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.68.1 assign(StringListValue other) Assign the contents of another object to self.

This method assigns the contents of another object to self. This is a deep copy that does not only copy the reference but the actual content.

13.68.2 destroy Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.68.3 [const] boolean destroyed Tell, if the object was destroyed.

Return: true — The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
false — The object is still alive.

13.68.4 [const] StringListValue dup Creates a copy of self.

Return: StringListValue — The copy of self.

13.68.5 [const] boolean has_value? True, if a value is present.

Return: true — A value is present.
false — No value is present.

13.68.6 [const] string[] value Get the actual value (a list of strings).

Return: string[] — The actual value(s) as a list of strings.

13.69 Class StringValue (version 0.21) Encapsulate a string value.

This class is provided as a return value of InputDialog::get_string, InputDialog::get_item and FileDialog. By using an object rather than a pure value, an object with has_value? = false can be returned indicating that the "Cancel" button was pressed.

Method Overview

has_value?	True, if a value is present.
to_s	Get the actual value (a synonym for value).
value	Get the actual value.
assign	Assign the contents of another object to self.
dup	Creates a copy of self.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.69.1 assign(StringValue other) Assign the contents of another object to self.

This method assigns the contents of another object to self. This is a deep copy that does not only copy the reference but the actual content.

13.69.2 destroy Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.69.3 [const] boolean destroyed Tell, if the object was destroyed.

Return:	true	The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
	false	The object is still alive.

13.69.4 [const] StringValue dup Creates a copy of self.

Return:	StringValue	The copy of self.

13.69.5 [const] boolean has_value? True, if a value is present.

Return:	true	A value is present.
	false	No value is present.

13.69.6 [const] string to_s

Get the actual value (a synonym for value).

Return: string The actual value(s) as a list of strings.

13.69.7 [const] string value

Get the actual value.

Return: string The actual value(s) as a list of strings.

13.70 Class Text (version 0.21) A text object.

A text object has a point (location), a text, a text transformation, a text size and a font id. Text size and font id are provided to be be able to render the text correctly.

Method Overview

from_dtext	Construct an integer coordinate text object from a floating-point coordinate text.
transformed_cplx	Transform the text with the given complex transformation.
new	Default constructor.
new	Constructor with string and transformation.
new	Constructor with string, transformation, text height and font.
string=	Assign a text string to this object.
string	Get the text string.
trans=	Assign a transformation (text position and orientation) to this object.
trans	Get the transformation.
size=	Set the text height of this object.
size	Get the text height.
font=	Set the font number.
font	Get the font number.
move	Moves the text by a certain distance.
moved	Returns the text moved by a certain distance.
transformed	Transform the text with the given simple transformation.
transformed_cplx	Transform the text with the given complex transformation.
transformed_cplx	Transform the text with the given complex transformation.
<	Less operator.
==	Equality test.
!=	Inequality test.
to_s	Convert to a string.
assign	Assign the contents of another object to self.
dup	Creates a copy of self.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.70.1 [const] boolean !=(Text text) Inequality test.

Input:	text	The object to compare against.
Return:	true	Inequality.
	false	???.

13.70.2 [const] boolean <(Text text) Less operator.

This operator is provided to establish some, not necessarily a certain sorting order.

Input:	text	The object to compare against.
Return:	true	This polygon is less than the given one.
	false	???.

13.70.3 [const] boolean ==(Text text)
Equality test.

Input:	text	The object to compare against.
Return:	true	The polygons are equal.
	false	???.

13.70.4 assign(Text other)
Assign the contents of another object to self.

This method assigns the contents of another object to self. This is a deep copy that does not only copy the reference but the actual content.

13.70.5 destroy
Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.70.6 [const] boolean destroyed
Tell, if the object was destroyed.

Return:	true	The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
	false	The object is still alive.

13.70.7 [const] Text dup
Creates a copy of self.

Return:	Text	The copy of self.

13.70.8 integer font
Get the font number.

Return:	integer	The integer representing a font.

13.70.9 font=(integer)
Set the font number.

Input:	integer	The integer representing a font.

13.70.10 [static] Text from_dtext(DText double_text)
Construct an integer coordinate text object from a floating-point coordinate text.

Input:	double_text	The floating-point coordinate text object.
Return:	Text	The integer coordinate text object.

13.70.11 ref Text move(Point p)
Moves the text by a certain distance.

Moves the text by a given offset and returns the moved text. Does not check for coordinate overflows.

Input:	p	The offset to move the text.
Return:	ref	A reference to this text object.

13.70.12 Text moved(Point p)
Returns the text moved by a certain distance.

Moves the text by a given offset and returns the moved text. Does not modify *this. Does not check for coordinate overflows.

Input:	p	The offset to move the text.
Return:	Text	The moved text.

13.70.13 [static] Text new
Default constructor.

Creates a text with unit transformation and empty text.

Return:	Text	The empty text object.

13.70.14 [static] Text new(string, Trans t)
Constructor with string and transformation.

A synonym for: [static] Text new_st(string, Trans t).

A string and a transformation is provided to this constructor. The transformation specifies the location and orientation of the text object.

Input:	string	The given text string.
	t	The specified transformation.
Return:	Text	The text object.

13.70.15 [static] Text new(string, Trans t, height, width)
Constructor with string, transformation, text height and font.

A synonym for: [static] Text new_sthr(string, Trans t, height, width).

A string and a transformation is provided to this constructor. The transformation specifies the location and orientation of the text object. In addition, the text height and font can be specified.

Input:	string	The given text string.
	t	The specified transformation.
	height	The text height.
	width	The text width.
Return:	Text	The text object.

13.70.16 [const] integer size
Get the text height.

Return:	integer	The text height.

13.70.17 size=(integer)
Set the text height of this object.

Input: integer The text height.

13.70.18 [const] string string
Get the text string.

Return: string The text string.

13.70.19 string=(string)
Assign a text string to this object.

Input: string The text string.

13.70.20 [const] string to_s
Convert to a string.

Return: string The text string.

13.70.21 [const] const ref Trans trans
Get the transformation.

13.70.22 trans=(Trans)
Assign a transformation (text position and orientation) to this object.

13.70.23 [const] Text transformed(Trans t)
Transform the text with the given simple transformation.

Input: t The transformation to apply.
Return: Text The transformed text.

13.70.24 [const] Text transformed_cplx(ICplxTrans t)
Transform the text with the given complex transformation.

Input: t The transformation to apply.
Return: Text The transformed text (in this case an integer coordinate object now).

13.70.25 [const] DText transformed_cplx(CplxTrans t)
Transform the text with the given complex transformation.

Input: t The transformation to apply.
Return: DText The transformed text (a DText now).

13.71 Class Trans (version 0.21) A simple transformation.

The simple transformation applies a displacement vector and a simple fix point transformation. This version acts on integer coordinates.

Method Overview

from_dtrans	Conversion constructor from a floating-point transformation.
new	Creates a unit transformation.
new	Conversion constructor from a fix-point transformation.
new	The standard constructor using angle and mirror flag.
new	The standard constructor using angle and mirror flag and two coordinate values for displacement.
new	The standard constructor using a code rather than angle and mirror.
new	The standard constructor using a code rather than angle and mirror and two coordinate values for displacement.
new	The standard constructor using a displacement only.
new	The standard constructor using a displacement given as two coordinates.
inverted	Inversion.
invert	In-place inversion.
ctrans	The transformation of a distance.
trans	The transformation of a point.
*	Multiplication (concatenation) of transformations.
<	A sorting criterion.
==	Equality test.
!=	Inequality test.
to_s	String conversion.
disp	Accessor to the point.
rot	Returns the respective rotation code if possible.
angle	Gets the angle.
is_mirror?	Gets the mirror flag.
angle=	Sets the angle.
disp=	Sets the displacement.
mirror=	Sets the mirror flag.
rot=	Sets the angle/mirror code
m_*/r_*	Various angle/mirror codes for the named transformation.
r0	"unrotated" transformation.
r90	"rotated by 90 degree counterclockwise" transformation.
r180	"rotated by 180 degree counterclockwise" transformation.
r270	"rotated by 270 degree counterclockwise" transformation.
m0	"mirrored at the x-axis" transformation.
m45	"mirrored at the 45 degree axis" transformation.
m90	"mirrored at the y (90 degree) axis" transformation.
m135	"mirrored at the 135 degree axis" transformation.
assign	Assign the contents of another object to self.
dup	Creates a copy of self.
destroy	Explicitly destroy the object.
destroyed	Tell, if the object was destroyed.

13.71.1 [const] boolean !=(Trans)
Inequality test.

Input:	Trans text	The object to compare against.
Return:	true	This object and the given one are not equal.
	false	???.

13.71.2 [const] Trans *(Trans t)
Multiplication (concatenation) of transformations.

The * operator returns self*t ("t is applied before this transformation").

Input:	t	The transformation to apply before.
Return:	Trans	The modified transformation.

13.71.3 [const] boolean <(Trans)
A sorting criterion.

Input:	e	The object to compare against.
Return:	true	The object is 'less' than the other.
	false	??.

13.71.4 [const] boolean ==(Trans)
Equality test.

Input:	e	The object to compare against.
Return:	true	Equality.
	false	??.

13.71.5 [const] integer angle
Gets the angle in units of 90 degree.

This value delivers the rotation component. In addition, a mirroring at the x axis may be applied before if the is_mirror? property is true.

Return:	integer	The rotation angle in units of 90 degree.

13.71.6 angle=(integer a)
Sets the angle in units of 90 degree.

This method was introduced in version 0.20.

Input:	a	The new angle.

13.71.7 assign(Trans other)
Assign the contents of another object to self.

This method assigns the contents of another object to self. This is a deep copy that does not only copy the reference but the actual content.

13.71.8 [const] integer ctrans(d)
The transformation of a distance.

The ctransmethod transforms the given distance: $e = t(d)$. For the simple transformations, there is no magnification and no modification of the distance therefore.

Input: d — The distance to transform.
Return: integer — The transformed distance.

13.71.9 destroy
Explicitly destroy the object.

Explicitly destroy the object on C++ side if it was owned by the Ruby interpreter. Subsequent access to this object will throw an exception. If the object is not owned by Ruby, this method will do nothing.

13.71.10 [const]boolean destroyed
Tell, if the object was destroyed.

Return: true — The object was destroyed, either explicitly or by the C++ side. The latter may happen, if the object is owned by a C++ object which got destroyed itself.
false — The object is still alive.

13.71.11 [const] const ref Point disp
Accessor to the point.

Return: ref — The accessor to the point.

13.71.12 disp=(Point u)
Sets the displacement.

This method was introduced in version 0.20.

Input: u — The new displacement.

13.71.13 [const] Trans dup
Creates a copy of self.

Return: Trans — The copy of self.

13.71.14 [static] Trans from_dtrans(DTrans double_trans)
Conversion constructor from an floating-point coordinate transformation.

Input: double_trans — The floating-point coordinate transformation.
Return: Trans — The integer coordinate transformation.

13.71.15 Trans invert
In-place inversion.

Inverts the transformation and replaces this transformation by the inverted one.

Return: Trans — The inverted and replaced transformation.

13.71.16 [const] Trans inverted
Inversion.

Return: Trans — The inverted transformation.

13.71.17 [const] boolean is_mirror?
Gets the mirror flag.

Return: true — The transformation is composed of a mirroring at the x-axis followed by a rotation by the angle given by the angle property.
false — ???.

13.71.18 [static] integer m_*/r_*
Various angle/mirror codes for the named transformation.

13.71.18.1 [static] integer m0 – "mirrored at the x-axis".

Return: integer — The angle/mirror code for this transformation.

13.71.18.2 [static] integer m135 – "mirrored at the 135 degree axis".

Return: integer — The angle/mirror code for this transformation.

13.71.18.3 [static] integer m45 – "mirrored at the 45 degree axis".

Return: integer — The angle/mirror code for this transformation.

13.71.18.4 [static] integer m90 – "mirrored at the 90 degree axis".

Return: integer — The angle/mirror code for this transformation.

13.71.18.5 [static] integer r0 – "unrotated".

Return: integer — The angle/mirror code for this transformation.

13.71.18.6 [static] integer r180 – "rotated by 180 degree counterclockwise".

Return: integer — The angle/mirror code for this transformation.

13.71.18.7 [static] integer r270 – "rotated by 270 degree counterclockwise".

Return: integer — The angle/mirror code for this transformation.

13.71.18.8 [static] integer r90 – "rotated by 90 degree counterclockwise".

Return: integer — The angle/mirror code for this transformation.

13.71.19 [const] double mag
Gets the magnification.

Return: integer The angle/mirror code for this transformation.

13.71.20 mirror=(boolean)
Sets the mirror flag.

"mirroring" describes a reflection at the x-axis which is included in the transformation prior to rotation.

This method was introduced in version 0.20.

Input: boolean The new mirror flag.

13.71.21 [static] Trans new
Creates a unit transformation.

13.71.22 [static] Trans new(f)
Conversion constructor from a fix point transformation.

A synonym of: [static] Trans new_f(f).

This constructor will create a transformation with a fix point transformation but no displacement.

Input: f The rotation/mirror code (r0 .. m135 constants).

13.71.23 [static] Trans new(rot, boolean, ref Point u)
The standard constructor using angle and mirror flag.

A synonym of: [static] Trans new_rmu(rot, boolean, ref Point u).

The sequence of operations is: mirroring at x axis, rotation, application of displacement.

Input: rot The rotation in units of 90 degree.
boolean True, if mirrored at x axis.
u The displacement.

13.71.24 [static] Trans new(rot, boolean, x, y)
The standard constructor using angle and mirror flag and two coordinate values for displacement.

A synonym of: [static] Trans new_rmxy(rot, boolean, x,y).

The sequence of operations is: mirroring at x axis, rotation, application of displacement.

Input: rot The rotation in units of 90 degree.
boolean True, if mirrored at x axis.
x The horizontal displacement.
y The vertical displacement.

13.71.25 [static] Trans new(f, Point u)
The standard constructor using a code rather than angle and mirror.

A synonym of: [static] Trans new_fu(f, Point u).

Input: f The rotation/mirror code (r0 .. m135 constants).
u The displacement.

13.71.26 [static] Trans new(f, x, y)
The standard constructor using a code rather than angle and mirror and two coordinate values for displacement.

A synonym of: [static] Trans new_fxy(f, x, y).

The sequence of operations is: mirroring at x axis, rotation, application of displacement.

Input: f The rotation/mirror code (r0 .. m135 constants).
x The horizontal displacement.
y The vertical displacement.

13.71.27 [static] Transnew(Point u)
The standard constructor using a displacement only.

A synonym of: [static] Trans new_u(Point u).

Input: u The displacement.

13.71.28 new(x, y)
The standard constructor using a displacement given as two coordinates.

Input: x The horizontal displacement.
y The vertical displacement.

13.71.29 [const] integer rot
Gets the angle/mirror code.

The angle/mirror code is one of the constants r0, r90, r180, r270, m0, m45, m90 and m135. rx is the rotation by an angle of x counter clockwise. mx is the mirroring at the axis given by the angle x (to the x-axis).

Return: integer The angle/mirror code for this transformation.

13.71.30 rot=(r)
Sets the angle/mirror code.

This method was introduced in version 0.20.

Input: r The new angle/rotation code (see rot property).

13.71.31 [const] string to_s
String conversion.

Return: string The string representing the object.

13.71.32 [const] Point trans(Point p)
The transformation of a point.

The trans method transforms the given point. q = t(p).

Input:	p	The point to transform.
Return:	Point	The transformed point.

The End

Comment: ToDo

GDS = Graphic Database System

GDSII stream format, common acronym GDSII, is a database file format originally developed by Calma in the 1970s and now owned by Cadence Design Systems. The GDSII format is the de facto industry standard for data exchange of integrated circuit or IC layout artwork. It is a binary file format representing planar geometric shapes, text labels, and other information about the layout in hierarchical form. The data can be used to reconstruct all or part of the artwork to be used in sharing layouts, transferring artwork between different tools, or creating photo masks.

DXF = Drawing Interchange Format, or Drawing Exchange Format developed by Autodesk, Inc.

OASIS = Open Artwork System Interchange Standard

The trade name OASIS is a registered trademark in the USA of Thomas J. Grebinski, Alamo, California and licensed for use exclusively by SEMI

(OASIS™) is a specification for hierarchical integrated circuit mask layout data format for interchange between EDA software, IC mask writing tools and mask inspection tools. The name is the trademark of SEMI. It is developed by SEMI for microelectronics and fabrication industry as a replacement for GDSII format, the IC industry de facto standard for IC layout data exchange for more than two decades. Like GDSII, OASIS™is a hardware- and software-independent binary data format. It delivers the improvements of a smaller file size over GDSII file format. The smaller file size may result in a faster loading of files, but due to its internal structure a higher computation power is needed which may lead to longer loading and saving times. The OASIS file format is not as common as the GDSII file format.

CIF = Caltech Intermediate Format

CIF is a recent form for the description of integrated circuits. Created by the university community, CIF has provided a common database structure for the integration of many research tools. CIF provides a limited set of graphics primitives that are useful for describing the two-dimensional shapes on the different layers of a chip. The format allows hierarchical description, which makes the representation concise. In addition, it is a terse but human-readable text format. CIF is therefore a concise and powerful descriptive form for VLSI geometry.

GerberPCB = The Gerber format is a file format used by printed circuit board (PCB) industry software to describe the images of a printed circuit board (copper layers, solder mask, legend, drill holes, etc.). The Gerber format is the de-facto industry standard for printed circuit board image transfer.

The specification can be freely downloaded.

There are two versions. RS-274X (”extended Gerber”) is the most commonly used today. The previous version was a subset of EIA RS-274-D (”standard Gerber”); it is deprecated and is largely superseded by RS-274X.

The Gerber format was developed by Gerber Systems Corp., a company founded by Heinz Joseph “Joe” Gerber. The format is now controlled and owned by Ucamco through its acquisition of Barco ETS, a company which previously acquired Gerber Systems Corp.