GDS-Render

Generated on Sat Apr 9 2022 20:42:55 for GDS-Render by Doxygen 1.9.3

Sat Apr 9 2022 20:42:55

Chapter 1

GDS-Render

This programm converts GDS layout files to

- PDF Files using the Cairo Renderer
- Latex code (TikZ) using the LaTeX / TikZ Renderer

See the Usage page for details and Compilation for building instructions and Version Number for the versioning scheme of this program.

Chapter 2

Compilation

2.1 Preface

GDS-Render is designed for UNIX-like, especially GNU/Linux based systems. It was developed under a Linux system. Therefore, best performance is expected using a Linux operating system.

2.2 Dependencies

The dependencies of GDS-Render are:

2.2.1 Program Dependencies

- GLib2
- GTK3
- Cairographics

2.2.2 Compilation Dependencies

These dependencies are not needed for running the program; just for compilation.

- Build System (GCC + binutils, make, etc...). Most distributions supply a "development" meta-package containing this stuff.
- cmake >= 2.8
- More or less optional: git. Used for extraction of the precise version number. It is strongly recommended to provide git!
- Optional: doxygen for this nice documentation.

The dependency list of GTK3 already includes Cairographics and GLib2. You should be on the safe side with a recent GTK3 version.

Development is done with the following library versions:

Cairographics	GLib2	GTK3
1.17.2	2.↩ 64.2	3.24.18

2.3 Compilation Instructions

2.3.1 General Linux Build Instruction

Go to the build directory you want to compile in. This may be the gds-render project root. Execute
`cmake -DCMAKE_BUILD_TYPE=Release <Path to gds-render root>`

for a build in release configuartion. Use `-DCMAKE_BUILD_TYPE=Debug` for debugging. Cmake will check the dependencies.

Once cmake has finished, type
`make`

to build the program and
`make documentation`

to build the doxygen documentation.

2.3.2 Archlinux Package

The subfolder 'AUR' contains a PKGBUILD file to build an Archlinux/Pacman package.

2.3.3 Compiler Warnings

The compiler will throw the following warnings. Compiled with GCC 9.3.0.

Warning	Assessment
warning: ‘calculate_path_miter_points’ defined but not used [-Wunused-function]	Ignore. Function will be used in later versions.

2.3.4 Compilation for Windows

Warning

> Windows is not a target system for this application, considering that this program converts GDS files which are most likely generated under a Linux system. The tips shown in this section are a guidance for anyone trying to build this application for Windows.
>
> Note that the Windows compatibility may decrease in future releases and a simple compilation like with this version might not be possible anymore.

The current release of 'gds-render' does not compile under a windows system, due to incompatibilities in the external library renderer. It is possible to comment out the code that causes the incompatibility. The external renderer will not be usable after this.

Steps:

- Go to file external-renderer.c
- Remove `#include` <dlfcn.h>
- comment out all code in external_renderer_render_cell

The program should now compile.

Warning

This guide is out of date. The Cairo renderer doesn't compile under windows anymore due to the usage of the fork() system call. It is possible to patch this out in order to restore Windows compatibility.

Chapter 3

Layer Mapping File Specification

File Format

The layer mapping file contains information on how to render the layers. The information is stored in CSV format – *True CSV*; not that rubbish with semicolons that Excel calls CSV.

Each line representing a layer consists of following fields:

```
layer,r,g,b,a,export,name
```

- **layer**: Layer number identifiying this layer.
- **r**,**b**,**g**,**a**: RGBA color value uning double precision float values in the range from 0 to 1.
- **export**: Either '1' or '0'. Defining whether to render this layer into the output file.
- **name**: The name of the layer.

the order of the layers inside the layer mapping file defines the layer stack in the rendered output. The first layer is at the bottom, the last at the top.

Handling Inside the GUI

The layer mapping file can be imported and exported inside the GUI.

Export

During export, all layer configurations are written to the mapping file

Import

During import, all layer configurations are loaded from the mapping file. This overwrites any configuration done to that layer. Layers that are not present in the layer mapping file are appended at the end of the list. This means, they are rendered on top of the other layers. Because the layer mapping file does not contain any information on these layers, their configuration is not reset during import.

Chapter 4

Usage

4.1 Command Line Interface

To use the application on the command line check 'gds-render --help'.

Usage: gds-render [OPTION...] FILE - Convert GDS file <FILE> to graphic

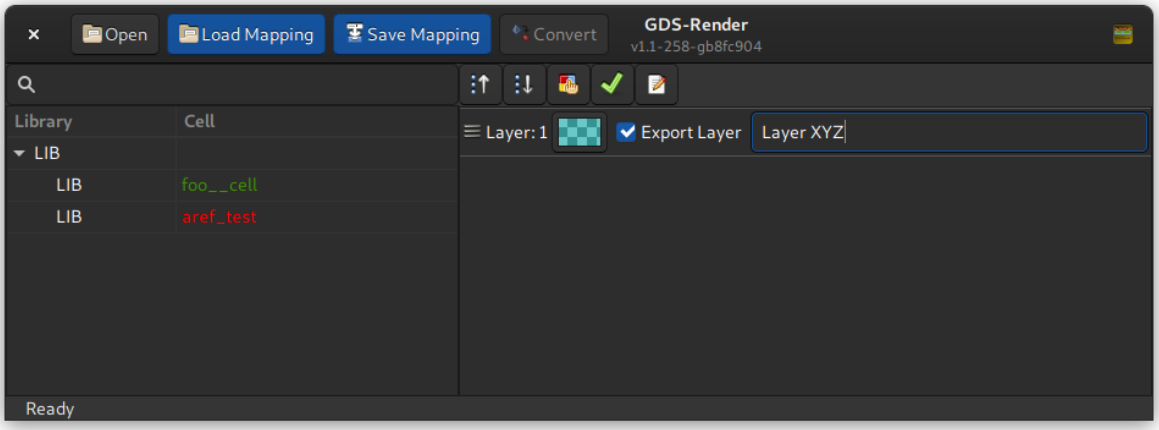
Help Options:
-h, --help Show help options
--help-all Show all help options
--help-gtk Show GTK+ Options

Application Options:
-v, --version Print version
-r, --renderer=pdf|svg|tikz|ext Renderer to use
-s, --scale=<SCALE> Divide output coordinates by <SCALE>
-o, --output-file=PATH Output file path
-m, --mapping=PATH Path for Layer Mapping File
-c, --cell=NAME Cell to render
-a, --tex-standalone Create standalone PDF
-l, --tex-layers Create PDF Layers (OCG)
-P, --custom-render-lib=PATH Path to a custom shared object, that implements the render_cell_to_file function
--display=DISPLAY X display to use

4.2 Graphical User Interface

The graphical user interface (GUI) can be used to open GDS Files, configure the layer rendering (colors, order, transparency etc.), and convert cells.

It is possible to export the layer configurations so they can be used later on. Even in the Command Line Interface



The cell selector on the left shows the GDS Libraries and Cells. The cells are marked green if all references inside the cell could be found. If not all references could be found, the cell is marked orange. This doens't show if child cells have missing childs. Only one level of the hierarchy is checked in order to make it easier to spot an errorneous cell. Cells with missing child cells are still renderable but -- obviously -- faulty. If a cell or any sub-cell contains a reference loop, the cell is marked red. In this case it can't be selected for rendering.

In the above image one cell is green; so everything is okay. And the other one is red, which indicates a reference loop. This cell cannot be selected for rendering!

Chapter 5

Version Number

5.1 Main Versioning Scheme

The version number of this application consists of a given version in the format of 'v1.0'. Where the first number indicates a major release and the second number indicates minor changes.

Versions, including release candidates and patch-levels, are tagged in git.

5.1.1 Release Candidates

Release candidates are software versions that seem stable and functional to become a new version but testing is not fully finished. These versions are marked with an '-rcX', where X is the number of the release candidate. The 3rd release candidate of version 4.2 would be '*v4.2-rc3*'. Release candidates are in a frozen state. Only bugfixes that are necessary for functionality are applied to these versions before releasing the final version.

5.1.2 Patch Levels

If an already released version contains bugs that need to be fixed, the version number is not incremented. Insted a new version number with a patch-level is created. The patch-level is appended with a dash directly after the version number. The fist patch-level of version 3.5 would be: 'v3.5-1'.

5.2 Git Based Version Number

The application and this documentation contain a git-based version number. With this version number not only released versions but all development points of the software can be uniquely identified.

An example for such a version number is: *v1.0-rc4-41-gaa41373-dirty*

It consists of the last Main Versioning Scheme (in this case version 1.0 – Release candidate 4) and some other information from the source code management system. The number after the version tag is the commit count after the given version. In this case the specified version is 41 commits after the last tagged version 'v1.0-rc4'. The next section always starts with a 'g' (for git) and after that contains the first letters of the commit ID. In this case an additional '-dirty' is appended, showing that the software version contains unstaged changes.

In tabular form: *v1.0-rc4-41-gaa41373-dirty*

Last tagged version	Commits since that version	Start of commit ID	Unstaged changes?
1.0-rc4	41	aa41373	yes

This git-based version number is automatically put into the application and this documentation during the application's compilation / the documentation's generation. For this *git* is needed. Therefore, it is highly recommended to have 'git' installed for compilation although it is no build dependency. In case of a missing git installation, the string "! version not set !" is compiled into the application.

Chapter 6

GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Lesser General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
one line to give the program's name and an idea of what it does.
Copyright (C) yyyy  name of author

This program is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License
as published by the Free Software Foundation; either version 2
of the License, or (at your option) any later version.
```

```
This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301, USA.
```

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
Gnomovision version 69, Copyright (C) year name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details
type 'show w'.  This is free software, and you are welcome
to redistribute it under certain conditions; type 'show c'
for details.
```

The hypothetical commands `show w' and `show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w' and `show c'; they could even be mouse-clicks or menu items–whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright
interest in the program 'Gnomovision'
(which makes passes at compilers) written
by James Hacker.

signature of Ty Coon, 1 April 1989
Ty Coon, President of Vice
```

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License.

Chapter 7

GDS-Render Readme

This software is a rendering programm for GDS2 layout files. The GDS2 format is mainly used in integrated circuit development. This program allows the conversion of a GDS file to a vector graphics file.

Output Formats

- Export GDS Layout to LaTeX (using TikZ).
- Export to PDF (Cairographics).

Features

Note: Due to various size limitations of both TikZ and the PDF export, the layout might not render correctly. In this case adjust the scale value. A higher scale value scales down your design.

- Configurable layer stack-up.
- Layer colors configurable as ARGB color values.
- Command line interface.
- ~~Awesome~~ Somehow usable GUI.

License and Other Stuff

- Free software (GPLv2 *only*)
- Coded in plain C using GTK+3.0, Glib2, and Cairographics

Chapter 8

Module Index

8.1 Modules

Here is a list of all modules:

Chapter 9

Data Structure Index

9.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 10

File Index

10.1 File List

Here is a list of all files with brief descriptions:

Chapter 11

Module Documentation

11.1 Activity Bar

Collaboration diagram for Activity Bar:

Data Structures

- struct _ActivityBar

  Opaque ActivityBar object. Not viewable outside this source file.

Macros

- #define TYPE_ACTIVITY_BAR (activity_bar_get_type())

Functions

- ActivityBar * activity_bar_new ()

  Create new Object ActivityBar.
- void activity_bar_set_ready (ActivityBar *bar)

  Deletes all applied tasks and sets bar to "Ready".
- void activity_bar_set_busy (ActivityBar *bar, const char *text)

  Enable spinner and set `text`. If text is NULL, 'Working...' is displayed.
- static void activity_bar_dispose (GObject *obj)
- static void activity_bar_class_init (ActivityBarClass *klass)
- static void activity_bar_init (ActivityBar *self)

11.1.1 Detailed Description

Activity Status Bar

11.1.2 Macro Definition Documentation

11.1.2.1 TYPE_ACTIVITY_BAR

```
#define TYPE_ACTIVITY_BAR (activity_bar_get_type())
```

Definition at line 42 of file activity-bar.h.

11.1.3 Function Documentation

11.1.3.1 activity_bar_class_init()

```
static void activity_bar_class_init (
            ActivityBarClass * klass )  [static]
```

Definition at line 66 of file activity-bar.c.

Here is the call graph for this function:

11.1.3.2 activity_bar_dispose()

```
static void activity_bar_dispose (
            GObject * obj )  [static]
```

Definition at line 52 of file activity-bar.c.

Here is the caller graph for this function:

11.1.3.3 activity_bar_init()

```
static void activity_bar_init (
            ActivityBar * self )  [static]
```

Definition at line 73 of file activity-bar.c.

11.1.3.4 activity_bar_new()

```
ActivityBar * activity_bar_new ( )
```

Create new Object ActivityBar.

Returns

New object. In case of error: NULL.

Definition at line 91 of file activity-bar.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.1.3.5 activity_bar_set_busy()

```
void activity_bar_set_busy (
            ActivityBar * bar,
            const char * text )
```

Enable spinner and set `text`. If text is NULL, 'Working...' is displayed.

Parameters

bar	Activity bar object
text	Text to display, may be NULL

Definition at line 108 of file activity-bar.c.

Here is the caller graph for this function:

11.1.3.6 activity_bar_set_ready()

```
void activity_bar_set_ready (
            ActivityBar * bar )
```

Deletes all applied tasks and sets bar to "Ready".

Parameters

in	*bar*	AcitivityBar object.

Definition at line 102 of file activity-bar.c.

Here is the caller graph for this function:

11.2 Cairo Renderer

Collaboration diagram for Cairo Renderer:

Data Structures

- struct _CairoRenderer
- struct cairo_layer

  The cairo_layer struct Each rendered layer is represented by this struct.

Macros

- #define GDS_RENDER_TYPE_CAIRO_RENDERER (cairo_renderer_get_type())
- #define MAX_LAYERS (5000)

  Maximum layer count the output renderer can process. Typically GDS only specifies up to 255 layers.

Functions

- CairoRenderer * cairo_renderer_new_svg ()

  Create new CairoRenderer for SVG output.
- CairoRenderer * cairo_renderer_new_pdf ()

  Create new CairoRenderer for PDF output.
- static void revert_inherited_transform (struct cairo_layer *layers)

  Revert the last transformation on all layers.
- static void apply_inherited_transform_to_all_layers (struct cairo_layer *layers, const struct gds_point *origin, double magnification, gboolean flipping, double rotation, double scale)

  Applies transformation to all layers.
- static void render_cell (struct gds_cell *cell, struct cairo_layer *layers, double scale)

  render_cell Render a cell with its sub-cells
- static int read_line_from_fd (int fd, char *buff, size_t buff_size)

  Read a line from a file descriptor.
- static int cairo_renderer_render_cell_to_vector_file (GdsOutputRenderer *renderer, struct gds_cell *cell, GList *layer_infos, const char *pdf_file, const char *svg_file, double scale)

  Render `cell` to a PDF file specified by `pdf_file`.
- static void cairo_renderer_init (CairoRenderer *self)
- static int cairo_renderer_render_output (GdsOutputRenderer *renderer, struct gds_cell *cell, double scale)
- static void cairo_renderer_class_init (CairoRendererClass *klass)

11.2.1 Detailed Description

11.2.2 Macro Definition Documentation

11.2.2.1 GDS_RENDER_TYPE_CAIRO_RENDERER

```
#define GDS_RENDER_TYPE_CAIRO_RENDERER (cairo_renderer_get_type())
```

Definition at line 39 of file cairo-renderer.h.

11.2.2.2 MAX_LAYERS

```
#define MAX_LAYERS (5000)
```

Maximum layer count the output renderer can process. Typically GDS only specifies up to 255 layers.

Definition at line 41 of file cairo-renderer.h.

11.2.3 Function Documentation

11.2.3.1 apply_inherited_transform_to_all_layers()

```
static void apply_inherited_transform_to_all_layers (
            struct cairo_layer * layers,
            const struct gds_point * origin,
            double magnification,
            gboolean flipping,
            double rotation,
            double scale )  [static]
```

Applies transformation to all layers.

Parameters

layers	Array of layers
origin	Origin translation
magnification	Scaling
flipping	Mirror image on x-axis before rotating
rotation	Rotation in degrees
scale	Scale the image down by. Only used for sclaing origin coordinates. Not applied to layer.

Definition at line 81 of file cairo-renderer.c.

Here is the caller graph for this function:

11.2.3.2 cairo_renderer_class_init()

```
static void cairo_renderer_class_init (
            CairoRendererClass * klass )  [static]
```

Definition at line 484 of file cairo-renderer.c.

Here is the call graph for this function:

11.2.3.3 cairo_renderer_init()

```
static void cairo_renderer_init (
            CairoRenderer * self )  [static]
```

Definition at line 442 of file cairo-renderer.c.

11.2.3.4 cairo_renderer_new_pdf()

```
CairoRenderer * cairo_renderer_new_pdf ( )
```

Create new CairoRenderer for PDF output.

Returns

New object

Definition at line 491 of file cairo-renderer.c.

Here is the caller graph for this function:

11.2.3.5 cairo_renderer_new_svg()

```
CairoRenderer * cairo_renderer_new_svg ( )
```

Create new CairoRenderer for SVG output.

Returns

New object

Definition at line 501 of file cairo-renderer.c.

Here is the caller graph for this function:

11.2.3.6 cairo_renderer_render_cell_to_vector_file()

```
static int cairo_renderer_render_cell_to_vector_file (
            GdsOutputRenderer * renderer,
            struct gds_cell * cell,
            GList * layer_infos,
            const char * pdf_file,
            const char * svg_file,
            double scale )  [static]
```

Render `cell` to a PDF file specified by `pdf_file`.

Parameters

renderer	The current renderer this function is running from
cell	Toplevel cell to Cairo Renderer
layer_infos	List of layer information. Specifies color and layer stacking
pdf_file	PDF output file. Set to NULL if no PDF file has to be generated
svg_file	SVG output file. Set to NULL if no SVG file has to be generated
scale	Scale the output image down by `scale`

Returns

Error

Definition at line 233 of file cairo-renderer.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.2.3.7 cairo_renderer_render_output()

```
static int cairo_renderer_render_output (
            GdsOutputRenderer * renderer,
            struct gds_cell * cell,
            double scale )  [static]
```

Definition at line 448 of file cairo-renderer.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.2.3.8 read_line_from_fd()

```
static int read_line_from_fd (
             int fd,
             char * buff,
             size_t buff_size )  [static]
```

Read a line from a file descriptor.

In case of a broken pipe / closed writing end, it will terminate

Parameters

fd	File descriptor to read from
buff	Buffer to write data in
buff_size	Buffer size

Returns

length of read data

Definition at line 203 of file cairo-renderer.c.

Here is the caller graph for this function:

11.2.3.9 render_cell()

```
static void render_cell (
             struct gds_cell * cell,
             struct cairo_layer * layers,
             double scale )  [static]
```

render_cell Render a cell with its sub-cells

Parameters

cell	Cell to render
layers	Cell will be rendered into these layers
scale	sclae image down by this factor

Definition at line 111 of file cairo-renderer.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.2.3.10 revert_inherited_transform()

```
static void revert_inherited_transform (
             struct cairo_layer * layers )  [static]
```

Revert the last transformation on all layers.

Parameters

layers	Pointer to cairo_layer structures

Definition at line 61 of file cairo-renderer.c.

Here is the caller graph for this function:

11.3 Command Line Interface

Data Structures

- struct external_renderer_params

  External renderer paramameters to command line renderer.

Functions

- static int string_array_count (char **string_array)
- static int create_renderers (char **renderers, char **output_file_names, gboolean tex_layers, gboolean tex_standalone, const struct external_renderer_params *ext_params, GList **renderer_list, LayerSettings *layer_settings)
- static struct gds_cell * find_gds_cell_in_lib (struct gds_library *lib, const char *cell_name)
- int command_line_convert_gds (const char *gds_name, const char *cell_name, char **renderers, char **output_file_names, const char *layer_file, struct external_renderer_params *ext_param, gboolean tex↩_standalone, gboolean tex_layers, double scale)

  Convert GDS according to command line parameters.

11.3.1 Detailed Description

11.3.2 Function Documentation

11.3.2.1 command_line_convert_gds()

```
int command_line_convert_gds (
            const char * gds_name,
            const char * cell_name,
            char ** renderers,
            char ** output_file_names,
            const char * layer_file,
            struct external_renderer_params * ext_param,
            gboolean tex_standalone,
            gboolean tex_layers,
            double scale )
```

Convert GDS according to command line parameters.

Parameters

gds_name	Path to GDS File
cell_name	Cell name
renderers	Renderer ids
output_file_names	Output file names
layer_file	Layer mapping file
ext_param	Settings for external library renderer
tex_standalone	Standalone TeX
tex_layers	TeX OCR layers
scale	Scale value

Returns

Error code, 0 if successful

Definition at line 138 of file command-line.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.3.2.2 create_renderers()

```
static int create_renderers (
             char ** renderers,
             char ** output_file_names,
             gboolean tex_layers,
             gboolean tex_standalone,
             const struct external_renderer_params * ext_params,
             GList ** renderer_list,
             LayerSettings * layer_settings )  [static]
```

Definition at line 55 of file command-line.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.3.2.3 find_gds_cell_in_lib()

```
static struct gds_cell * find_gds_cell_in_lib (
             struct gds_library * lib,
             const char * cell_name )  [static]
```

Definition at line 122 of file command-line.c.

Here is the caller graph for this function:

11.3.2.4 string_array_count()

```
static int string_array_count (
             char ** string_array )  [static]
```

Definition at line 42 of file command-line.c.

Here is the caller graph for this function:

11.4 External Shared Object Renderer

Collaboration diagram for External Shared Object Renderer:

Data Structures

- struct _ExternalRenderer

Macros

- #define EXPORT_FUNC __attribute__((visibility("default")))

  This define is used to export a function from a shared object.
- #define EXTERNAL_LIBRARY_RENDER_FUNCTION exported_render_cell_to_file

  Function name expected to be found in external library for rendering.
- #define EXTERNAL_LIBRARY_INIT_FUNCTION exported_init

  Function name expected to be found in external library for initialization.
- #define EXTERNAL_LIBRARY_FORK_REQUEST exported_fork_request

  Global integer specified by an external renderer to signal, that the init and render functions shall be executed in a subprocess.
- #define EXPORTED_FUNC_DECL(FUNC) EXPORT_FUNC FUNC

  Define for declaring the exported functions.
- #define GDS_RENDER_TYPE_EXTERNAL_RENDERER (external_renderer_get_type())
- #define FORCE_FORK 0U

  if != 0, then forking is forced regardless of the shared object's settings

Enumerations

- enum { PROP_SO_PATH = 1 , PROP_PARAM_STRING , N_PROPERTIES }

Functions

- ExternalRenderer * external_renderer_new ()

  Create new ExternalRenderer object.
- ExternalRenderer * external_renderer_new_with_so_and_param (const char *so_path, const char *param↩_string)

  Create new ExternalRenderer object with specified shared object path.
- static int external_renderer_render_cell (struct gds_cell *toplevel_cell, GList *layer_info_list, const char *output_file, double scale, const char *so_path, const char *params)

  Execute render function in shared object to render the supplied cell.
- static int external_renderer_render_output (GdsOutputRenderer *renderer, struct gds_cell *cell, double scale)
- static void external_renderer_get_property (GObject *obj, guint property_id, GValue *value, GParamSpec *pspec)
- static void external_renderer_set_property (GObject *obj, guint property_id, const GValue *value, GParam↩Spec *pspec)
- static void external_renderer_dispose (GObject *self_obj)
- static void external_renderer_class_init (ExternalRendererClass *klass)
- static void external_renderer_init (ExternalRenderer *self)

Variables

- static GParamSpec * external_renderer_properties [N_PROPERTIES] = {NULL}

11.4.1 Detailed Description

11.4.2 Properties

This class inherits all properties from its parent GDS Output Renderer base class. In addition to that, it implements the following properties:

Property Name	Description
shared-object-path	Path to the shared object used for rendering
param-string	Command line parameters passed to external renderer's init function

All these properties have to be set for rendering.

11.4.3 Necessary Functions

The following functions and variables are necessary for an external renderer to implement:

Code Define	Prototype	Description
EXTERNAL_LIBRARY_RENDER_FUNCTION	int EXTERNAL_LIBRARY_RENDER_FUNCTION(struct RenderCell *toplevel, GList *layer_info_	Render cell to output file
EXTERNAL_LIBRARY_INIT_FUNCTION	int EXTERNAL_LIBRARY_INIT_FUNCTION(const char *params_string, const char *version_s	Init function. Executed before rendering. This is given the command line parameters specified for the external renderer and the version string of the currently running gds-render program.
EXTERNAL_LIBRARY_FORK_REQUEST	int EXTERNAL_LIBRARY_FORK_REQUEST;	The pure presence of this integer results in the execution inside a subprocess of hte whole shared object's code

11.4.4 Macro Definition Documentation

11.4.4.1 EXPORT_FUNC

```
#define EXPORT_FUNC __attribute__((visibility("default")))
```

This define is used to export a function from a shared object.

Definition at line 38 of file external-renderer-interfaces.h.

11.4.4.2 EXPORTED_FUNC_DECL

```
#define EXPORTED_FUNC_DECL(
              FUNC ) EXPORT_FUNC FUNC
```

Define for declaring the exported functions.

This not only helps with the declaration but also makes the symbols visible, so they can be called form outside the library

Definition at line 72 of file external-renderer-interfaces.h.

11.4.4.3 EXTERNAL_LIBRARY_FORK_REQUEST

```
#define EXTERNAL_LIBRARY_FORK_REQUEST exported_fork_request
```

Global integer specified by an external renderer to signal, that the init and render functions shall be executed in a subprocess.

The pure presence of this symbol name causes forking. The content of this variable is don't care.

Note

Use this if you mess with the internal structures of gds-render

Definition at line 65 of file external-renderer-interfaces.h.

11.4.4.4 EXTERNAL_LIBRARY_INIT_FUNCTION

```
#define EXTERNAL_LIBRARY_INIT_FUNCTION exported_init
```

Function name expected to be found in external library for initialization.

```
int EXTERNAL_LIBRARY_INIT_FUNCTION(const char *option_string, const char *version_string);
```

Definition at line 57 of file external-renderer-interfaces.h.

11.4.4.5 EXTERNAL_LIBRARY_RENDER_FUNCTION

```
#define EXTERNAL_LIBRARY_RENDER_FUNCTION exported_render_cell_to_file
```

Function name expected to be found in external library for rendering.

The function has to be defined as follows:

```
int EXTERNAL_LIBRARY_RENDER_FUNCTION(struct gds_cell *toplevel, GList *layer_info_list, const char
      *output_file_name, double scale);
```

Definition at line 48 of file external-renderer-interfaces.h.

11.4.4.6 FORCE_FORK

```
#define FORCE_FORK 0U
```

if != 0, then forking is forced regardless of the shared object's settings

Definition at line 39 of file external-renderer.c.

11.4.4.7 GDS_RENDER_TYPE_EXTERNAL_RENDERER

```
#define GDS_RENDER_TYPE_EXTERNAL_RENDERER (external_renderer_get_type())
```

Definition at line 40 of file external-renderer.h.

11.4.5 Enumeration Type Documentation

11.4.5.1 anonymous enum

```
anonymous enum
```

Enumerator

PROP_SO_PATH	Shared object path property.
PROP_PARAM_STRING	
N_PROPERTIES	Shared object renderer parameter string from CLI. Used to get property count

Definition at line 47 of file external-renderer.c.

11.4.6 Function Documentation

11.4.6.1 external_renderer_class_init()

```
static void external_renderer_class_init (
            ExternalRendererClass * klass )  [static]
```

Definition at line 232 of file external-renderer.c.

Here is the call graph for this function:

11.4.6.2 external_renderer_dispose()

```
static void external_renderer_dispose (
            GObject * self_obj )  [static]
```

Definition at line 216 of file external-renderer.c.

Here is the caller graph for this function:

11.4.6.3 external_renderer_get_property()

```
static void external_renderer_get_property (
            GObject * obj,
            guint property_id,
            GValue * value,
            GParamSpec * pspec )  [static]
```

Definition at line 174 of file external-renderer.c.

Here is the caller graph for this function:

11.4.6.4 external_renderer_init()

```
static void external_renderer_init (
            ExternalRenderer * self )  [static]
```

Definition at line 264 of file external-renderer.c.

11.4.6.5 external_renderer_new()

```
ExternalRenderer * external_renderer_new ( )
```

Create new ExternalRenderer object.

Returns

New object

Definition at line 270 of file external-renderer.c.

11.4.6.6 external_renderer_new_with_so_and_param()

```
ExternalRenderer * external_renderer_new_with_so_and_param (
            const char * so_path,
            const char * param_string )
```

Create new ExternalRenderer object with specified shared object path.

Parameters

so_path	Path to shared object, the rendering function is searched in
param_string	Command line parameter string passed to external renderer

Returns

New object.

Definition at line 275 of file external-renderer.c.

Here is the caller graph for this function:

11.4.6.7 external_renderer_render_cell()

```
static int external_renderer_render_cell (
            struct gds_cell * toplevel_cell,
            GList * layer_info_list,
            const char * output_file,
            double scale,
            const char * so_path,
            const char * params )  [static]
```

Execute render function in shared object to render the supplied cell.

Parameters

toplevel_cell	Cell to render
layer_info_list	Layer information (Color etc.)
output_file	Destination file
scale	the scaling value to scale the output cell down by.
so_path	Path to shared object
params	Parameters passed to EXTERNAL_LIBRARY_INIT_FUNCTION

Returns

0 if successful

Definition at line 65 of file external-renderer.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.4.6.8 external_renderer_render_output()

```
static int external_renderer_render_output (
            GdsOutputRenderer * renderer,
            struct gds_cell * cell,
            double scale )  [static]
```

Definition at line 149 of file external-renderer.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.4.6.9 external_renderer_set_property()

```
static void external_renderer_set_property (
            GObject * obj,
            guint property_id,
            const GValue * value,
            GParamSpec * pspec )  [static]
```

Definition at line 193 of file external-renderer.c.

Here is the caller graph for this function:

11.4.7 Variable Documentation

11.4.7.1 external_renderer_properties

```
GParamSpec* external_renderer_properties[N_PROPERTIES] = {NULL}  [static]
```

Definition at line 230 of file external-renderer.c.

11.5 GDS Output Renderer base class

Collaboration diagram for GDS Output Renderer base class:

Modules

- Cairo Renderer
- External Shared Object Renderer
- LaTeX / TikZ Renderer

Data Structures

- struct _GdsOutputRendererClass

  Base output renderer class structure.
- struct renderer_params
- struct idle_function_params
- struct GdsOutputRendererPrivate

Macros

- #define GDS_RENDER_TYPE_OUTPUT_RENDERER (gds_output_renderer_get_type())

Enumerations

- enum { GDS_OUTPUT_RENDERER_GEN_ERR = -100 , GDS_OUTPUT_RENDERER_PARAM_ERR = -200 }
- enum { PROP_OUTPUT_FILE = 1 , PROP_LAYER_SETTINGS , N_PROPERTIES }
- enum gds_output_renderer_signal_ids { ASYNC_FINISHED = 0 , ASYNC_PROGRESS_CHANGED , GDS_OUTPUT_RENDERER_SIGNAL_COUNT }

Functions

- G_DECLARE_DERIVABLE_TYPE (GdsOutputRenderer, gds_output_renderer, GDS_RENDER, OUTPUT↩ _RENDERER, GObject)
- GdsOutputRenderer ∗ gds_output_renderer_new ()

  Create a new GdsOutputRenderer GObject.
- GdsOutputRenderer ∗ gds_output_renderer_new_with_props (const char ∗output_file, LayerSettings ∗layer_settings)

  Create a new GdsOutputRenderer GObject with its properties.
- int gds_output_renderer_render_output (GdsOutputRenderer ∗renderer, struct gds_cell ∗cell, double scale)

  gds_output_renderer_render_output
- void gds_output_renderer_set_output_file (GdsOutputRenderer ∗renderer, const gchar ∗file_name)

  Convenience function for setting the "output-file" property.
- const char ∗ gds_output_renderer_get_output_file (GdsOutputRenderer ∗renderer)

  Convenience function for getting the "output-file" property.
- LayerSettings ∗ gds_output_renderer_get_and_ref_layer_settings (GdsOutputRenderer ∗renderer)

  Get layer settings.
- void gds_output_renderer_set_layer_settings (GdsOutputRenderer ∗renderer, LayerSettings ∗settings)

  Set layer settings.
- int gds_output_renderer_render_output_async (GdsOutputRenderer ∗renderer, struct gds_cell ∗cell, double scale)

  Render output asynchronously.
- void gds_output_renderer_update_async_progress (GdsOutputRenderer ∗renderer, const char ∗status)

  This function emits the 'progress-changed' in the thread/context that triggered an asynchronous rendering.
- static int gds_output_renderer_render_dummy (GdsOutputRenderer ∗renderer, struct gds_cell ∗cell, double scale)
- static void gds_output_renderer_dispose (GObject ∗self_obj)
- static void gds_output_renderer_get_property (GObject ∗obj, guint property_id, GValue ∗value, GParamSpec ∗pspec)
- static void gds_output_renderer_set_property (GObject ∗obj, guint property_id, const GValue ∗value, GParamSpec ∗pspec)
- static void gds_output_renderer_class_init (GdsOutputRendererClass ∗klass)
- void gds_output_renderer_init (GdsOutputRenderer ∗self)
- static void gds_output_renderer_async_wrapper (GTask ∗task, gpointer source_object, gpointer task_data, GCancellable ∗cancellable)
- static void gds_output_renderer_async_finished (GObject ∗src_obj, GAsyncResult ∗res, gpointer user_data)
- static gboolean idle_event_processor_callback (gpointer user_data)

Variables

- static guint gds_output_renderer_signals [GDS_OUTPUT_RENDERER_SIGNAL_COUNT]
- static GParamSpec ∗ gds_output_renderer_properties [N_PROPERTIES] = {NULL}

11.5.1 Detailed Description

The renderers are used to convert the cell structures read from the GDS layout file into different output formats.

The GdsOutputRenderer base class is used to derive all renderers from.

Warning

Although the GdsOutputRenderer class provides compatibility for asynchronous rendering, the class is not thread safe / re-entrant. Only use it from a signle context. Not even the rendering function called is allowed to modifiy this object.

A allowed function to be called from the async rendering thread is gds_output_renderer_update_async_progress and the get functions for the properties.

Note

The context that owned the renderer has to ensure that only one rendering is active at a time for a single instance of a renderer.

By default this class implements the following features:

11.5.2 Properties

Property Name	Description
layer-settings	LayerSettings object containing the layer rendering information
output-file	Output file name for rendering

All these properties have to be set for rendering.

11.5.3 Signals / Events

Signal Name	Description	Callback prototype
async-finished	The asynchronous rendering is finished	void callback(GdsOutputRenderer *src, gpointer user_data)
progress-changed	The asynchronous rendering progress changed	void callback(GdsOutputRenderer *src, const char *progress, gpointer user_data)

Note

The `char *progress` supplied to the callback function must not be modified or freed.

11.5.4 Macro Definition Documentation

11.5.4.1 GDS_RENDER_TYPE_OUTPUT_RENDERER

```
#define GDS_RENDER_TYPE_OUTPUT_RENDERER (gds_output_renderer_get_type())
```

Definition at line 41 of file gds-output-renderer.h.

11.5.5 Enumeration Type Documentation

11.5.5.1 anonymous enum

```
anonymous enum
```

Enumerator

GDS_OUTPUT_RENDERER_GEN_ERR	Error set by the _GdsOutputRendererClass::render_output virtual function, if renderer is invalid.
GDS_OUTPUT_RENDERER_PARAM_ERR	Error set by the _GdsOutputRendererClass::render_output virtual function, if parameters are faulty.

Definition at line 61 of file gds-output-renderer.h.

11.5.5.2 anonymous enum

```
anonymous enum
```

Enumerator

PROP_OUTPUT_FILE	
PROP_LAYER_SETTINGS	
N_PROPERTIES	

Definition at line 55 of file gds-output-renderer.c.

11.5.5.3 gds_output_renderer_signal_ids

```
enum gds_output_renderer_signal_ids
```

Enumerator

ASYNC_FINISHED	
ASYNC_PROGRESS_CHANGED	
GDS_OUTPUT_RENDERER_SIGNAL_COUNT	

Definition at line 63 of file gds-output-renderer.c.

11.5.6 Function Documentation

11.5.6.1 G_DECLARE_DERIVABLE_TYPE()

```
G_DECLARE_DERIVABLE_TYPE (
            GdsOutputRenderer ,
            gds_output_renderer ,
            GDS_RENDER ,
            OUTPUT_RENDERER ,
            GObject  )
```

11.5.6.2 gds_output_renderer_async_finished()

```
static void gds_output_renderer_async_finished (
            GObject * src_obj,
            GAsyncResult * res,
            gpointer user_data )  [static]
```

Definition at line 346 of file gds-output-renderer.c.

Here is the caller graph for this function:

11.5.6.3 gds_output_renderer_async_wrapper()

```
static void gds_output_renderer_async_wrapper (
            GTask * task,
            gpointer source_object,
            gpointer task_data,
            GCancellable * cancellable )  [static]
```

Definition at line 318 of file gds-output-renderer.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.5.6.4 gds_output_renderer_class_init()

```
static void gds_output_renderer_class_init (
            GdsOutputRendererClass * klass )  [static]
```

Definition at line 164 of file gds-output-renderer.c.

Here is the call graph for this function:

11.5.6.5 gds_output_renderer_dispose()

```
static void gds_output_renderer_dispose (
            GObject * self_obj )  [static]
```

Definition at line 78 of file gds-output-renderer.c.

Here is the caller graph for this function:

11.5.6.6 gds_output_renderer_get_and_ref_layer_settings()

```
LayerSettings * gds_output_renderer_get_and_ref_layer_settings (
            GdsOutputRenderer * renderer )
```

Get layer settings.

This is a convenience function for getting the "layer-settings" property. This also references it. This is to prevent race conditions with another thread that might alter the layer settings before they are read out.

Parameters

renderer	Renderer

Returns

Layer settings object

Definition at line 255 of file gds-output-renderer.c.

Here is the caller graph for this function:

11.5.6.7 gds_output_renderer_get_output_file()

```
const char * gds_output_renderer_get_output_file (
            GdsOutputRenderer * renderer )
```

Convenience function for getting the "output-file" property.

Parameters

renderer	

Returns

Output file path. This must not be freed

Definition at line 247 of file gds-output-renderer.c.

Here is the caller graph for this function:

11.5.6.8 gds_output_renderer_get_property()

```
static void gds_output_renderer_get_property (
            GObject * obj,
            guint property_id,
            GValue * value,
            GParamSpec * pspec )  [static]
```

Definition at line 114 of file gds-output-renderer.c.

Here is the caller graph for this function:

11.5.6.9 gds_output_renderer_init()

```
void gds_output_renderer_init (
            GdsOutputRenderer * self )
```

Definition at line 208 of file gds-output-renderer.c.

11.5.6.10 gds_output_renderer_new()

```
GdsOutputRenderer * gds_output_renderer_new ( )
```

Create a new GdsOutputRenderer GObject.

Returns

New object

Definition at line 224 of file gds-output-renderer.c.

11.5.6.11 gds_output_renderer_new_with_props()

```
GdsOutputRenderer * gds_output_renderer_new_with_props (
            const char * output_file,
            LayerSettings * layer_settings )
```

Create a new GdsOutputRenderer GObject with its properties.

Parameters

output_file	Output file of the renderer
layer_settings	Layer settings object

Returns

New object

Definition at line 229 of file gds-output-renderer.c.

11.5.6.12 gds_output_renderer_render_dummy()

```
static int gds_output_renderer_render_dummy (
            GdsOutputRenderer * renderer,
            struct gds_cell * cell,
            double scale )  [static]
```

Definition at line 66 of file gds-output-renderer.c.

Here is the caller graph for this function:

11.5.6.13 gds_output_renderer_render_output()

```
int gds_output_renderer_render_output (
            GdsOutputRenderer * renderer,
            struct gds_cell * cell,
            double scale )
```

gds_output_renderer_render_output

Parameters

renderer	Renderer object
cell	Cell to render
scale	scale value. The output is scaled *down* by this value

Returns

0 if successful

Definition at line 281 of file gds-output-renderer.c.

Here is the caller graph for this function:

11.5.6.14 gds_output_renderer_render_output_async()

```
int gds_output_renderer_render_output_async (
            GdsOutputRenderer * renderer,
            struct gds_cell * cell,
            double scale )
```

Render output asynchronously.

This function will render in a separate thread. To wait for the completion of the rendering process.

Note

A second async thread cannot be spawned.

Parameters

renderer	Output renderer
cell	Cell to render
scale	Scale

Returns

0 if successful. In case no thread can be spawned < 0

Definition at line 363 of file gds-output-renderer.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.5.6.15 gds_output_renderer_set_layer_settings()

```
void gds_output_renderer_set_layer_settings (
            GdsOutputRenderer * renderer,
            LayerSettings * settings )
```

Set layer settings.

This is a convenience function for setting the "layer-settings" property.

If another Layer settings has previously been supplied, it is unref'd.

Parameters

renderer	Renderer
settings	LayerSettings object

Definition at line 274 of file gds-output-renderer.c.

Here is the caller graph for this function:

11.5.6.16 gds_output_renderer_set_output_file()

```
void gds_output_renderer_set_output_file (
            GdsOutputRenderer * renderer,
            const gchar * file_name )
```

Convenience function for setting the "output-file" property.

Parameters

renderer	Renderer object
file_name	Output file path

Definition at line 237 of file gds-output-renderer.c.

Here is the caller graph for this function:

11.5.6.17 gds_output_renderer_set_property()

```
static void gds_output_renderer_set_property (
            GObject * obj,
            guint property_id,
            const GValue * value,
            GParamSpec * pspec )  [static]
```

Definition at line 134 of file gds-output-renderer.c.

Here is the caller graph for this function:

11.5.6.18 gds_output_renderer_update_async_progress()

```
void gds_output_renderer_update_async_progress (
            GdsOutputRenderer * renderer,
            const char * status )
```

This function emits the 'progress-changed' in the thread/context that triggered an asynchronous rendering.

If the rendering is not asynchronous, this function has no effect.

Parameters

renderer	GdsOutputrenderer object
status	Status to supply to signal emission

Definition at line 422 of file gds-output-renderer.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.5.6.19 idle_event_processor_callback()

```
static gboolean idle_event_processor_callback (
            gpointer user_data )  [static]
```

Definition at line 394 of file gds-output-renderer.c.

Here is the caller graph for this function:

11.5.7 Variable Documentation

11.5.7.1 gds_output_renderer_properties

```
GParamSpec* gds_output_renderer_properties[N_PROPERTIES] = {NULL}  [static]
```

Definition at line 162 of file gds-output-renderer.c.

11.5.7.2 gds_output_renderer_signals

```
guint gds_output_renderer_signals[GDS_OUTPUT_RENDERER_SIGNAL_COUNT]  [static]
```

Definition at line 64 of file gds-output-renderer.c.

11.6 Geometric Helper Functions

Data Structures

- union bounding_box
  Union describing a bounding box.
- struct vector_2d

Macros

- #define MIN(a, b) (((a) < (b)) ? (a) : (b))
  Return smaller number.
- #define MAX(a, b) (((a) > (b)) ? (a) : (b))
  Return bigger number.
- #define ABS_DBL(a) ((a) < 0 ? -(a) : (a))
- #define ABS_DBL(a) ((a) < 0.0 ? -(a) : (a))
- #define DEG2RAD(a) ((a)*M_PI/180.0)

Typedefs

- typedef void(* conv_generic_to_vector_2d_t) (void *, struct vector_2d *)

Functions

- void bounding_box_calculate_from_polygon (GList *vertices, conv_generic_to_vector_2d_t conv_func, union bounding_box *box)

  Calculate bounding box of polygon.
- void bounding_box_update_with_box (union bounding_box *destination, union bounding_box *update)

  Update an exisitng bounding box with another one.
- void bounding_box_prepare_empty (union bounding_box *box)

  Prepare an empty bounding box.
- static void calculate_path_miter_points (struct vector_2d *a, struct vector_2d *b, struct vector_2d *c, struct vector_2d *m1, struct vector_2d *m2, double width)

  Calculate path miter points for a pathwith a `width` and the anchors `a b c`.
- void bounding_box_update_with_path (GList *vertices, double thickness, conv_generic_to_vector_2d_t conv_func, union bounding_box *box)

  Calculate the bounding box of a path and update the given bounding box.
- void bounding_box_update_with_point (union bounding_box *destination, conv_generic_to_vector_2d_t conv_func, void *pt)

  Update bounding box with a point.
- void bounding_box_get_all_points (struct vector_2d *points, union bounding_box *box)

  Return all four corner points of a bounding box.
- void bounding_box_apply_transform (double scale, double rotation_deg, bool flip_at_x, union bounding_box *box)

  Apply transformations onto bounding box.
- static void convert_gds_point_to_2d_vector (struct gds_point *pt, struct vector_2d *vector)
- static void update_box_with_gfx (union bounding_box *box, struct gds_graphics *gfx)

  Update the given bounding box with the bounding box of a graphics element.
- void calculate_cell_bounding_box (union bounding_box *box, struct gds_cell *cell)

  Calculate bounding box of a gds cell.
- double vector_2d_scalar_multipy (struct vector_2d *a, struct vector_2d *b)
- void vector_2d_normalize (struct vector_2d *vec)
- void vector_2d_rotate (struct vector_2d *vec, double angle)
- struct vector_2d * vector_2d_copy (struct vector_2d *opt_res, struct vector_2d *vec)
- struct vector_2d * vector_2d_alloc (void)
- void vector_2d_free (struct vector_2d *vec)
- void vector_2d_scale (struct vector_2d *vec, double scale)
- double vector_2d_abs (struct vector_2d *vec)
- double vector_2d_calculate_angle_between (struct vector_2d *a, struct vector_2d *b)
- void vector_2d_subtract (struct vector_2d *res, struct vector_2d *a, struct vector_2d *b)
- void vector_2d_add (struct vector_2d *res, struct vector_2d *a, struct vector_2d *b)

11.6.1 Detailed Description

The geometric helper function are used to calculate bounding boxes

Warning

Code is incomplete. Please double check for functionality!

11.6.2 Macro Definition Documentation

11.6.2.1 ABS_DBL [1/2]

```
#define ABS_DBL(
              a ) ((a) < 0 ?  -(a) :  (a))
```

Definition at line 38 of file bounding-box.c.

11.6.2.2 ABS_DBL [2/2]

```
#define ABS_DBL(
              a ) ((a) < 0.0 ?  -(a) :  (a))
```

Definition at line 36 of file vector-operations.c.

11.6.2.3 DEG2RAD

```
#define DEG2RAD(
              a ) ((a)*M_PI/180.0)
```

Definition at line 42 of file vector-operations.h.

11.6.2.4 MAX

```
#define MAX(
              a,
              b ) (((a) > (b)) ?  (a) :  (b))
```

Return bigger number.

Definition at line 37 of file bounding-box.c.

11.6.2.5 MIN

```
#define MIN(
              a,
              b ) (((a) < (b)) ?  (a) :  (b))
```

Return smaller number.

Definition at line 36 of file bounding-box.c.

11.6.3 Typedef Documentation

11.6.3.1 conv_generic_to_vector_2d_t

```
typedef void(* conv_generic_to_vector_2d_t) (void *, struct vector_2d *)
```

Definition at line 76 of file bounding-box.h.

11.6.4 Function Documentation

11.6.4.1 bounding_box_apply_transform()

```
void bounding_box_apply_transform (
            double scale,
            double rotation_deg,
            bool flip_at_x,
            union bounding_box * box )
```

Apply transformations onto bounding box.

All corner points $\vec{P_i}$ of the bounding box are transformed to output points $\vec{P_o}$ by:

$$\vec{P_o} = s \cdot \begin{pmatrix} \cos(\phi) & -\sin(\phi) \\ \sin(\phi) & \cos(\phi) \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ 0 & -1^m \end{pmatrix} \cdot \vec{P_i}, \text{ with:}$$

- s: Scale
- m: 1, if flipped_at_x is True, else 0
- ϕ: Rotation angle in radians. The conversion degrees => radians is done internally

The result is the bounding box generated around all output points

Parameters

scale	Scaling factor
rotation_deg	Rotation of bounding box around the origin in degrees (counterclockwise)
flip_at_x	Flip the boundig box on the x axis before rotating.
box	Bounding box the operations should be applied to.

Note

Keep in mind, that this bounding box is actually the bounding box of the rotated boundig box and not the object itself. It might be too big.

Definition at line 207 of file bounding-box.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.6.4.2 bounding_box_calculate_from_polygon()

```
void bounding_box_calculate_from_polygon (
            GList * vertices,
            conv_generic_to_vector_2d_t conv_func,
            union bounding_box * box )
```

Calculate bounding box of polygon.

Parameters

vertices	List of vertices that describe the polygon
conv_func	Conversion function to convert vertices to vector_2d structs.
box	Box to write to. This box is not updated! All previous data is discarded

Definition at line 40 of file bounding-box.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.6.4.3 bounding_box_get_all_points()

```
void bounding_box_get_all_points (
            struct vector_2d * points,
            union bounding_box * box )
```

Return all four corner points of a bounding box.

Parameters

out	*points*	Array of 4 vector_2d structs that has to be allocated by the caller
	box	Bounding box

Definition at line 192 of file bounding-box.c.

Here is the caller graph for this function:

11.6.4.4 bounding_box_prepare_empty()

```
void bounding_box_prepare_empty (
            union bounding_box * box )
```

Prepare an empty bounding box.

Updating this specially prepared box, results in a bounding box that is the same size as the update

Parameters

box	Box to preapre

Definition at line 86 of file bounding-box.c.

Here is the caller graph for this function:

11.6.4.5 bounding_box_update_with_box()

```
void bounding_box_update_with_box (
            union bounding_box * destination,
            union bounding_box * update )
```

Update an exisitng bounding box with another one.

Parameters

destination	Target box to update
update	Box to update the target with

Definition at line 71 of file bounding-box.c.

Here is the caller graph for this function:

11.6.4.6 bounding_box_update_with_path()

```
void bounding_box_update_with_path (
            GList * vertices,
            double thickness,
            conv_generic_to_vector_2d_t conv_func,
            union bounding_box * box )
```

Calculate the bounding box of a path and update the given bounding box.

Parameters

vertices	Vertices the path is made up of
thickness	Thisckness of the path
conv_func	Conversion function for vertices to vector_2d structs
box	Bounding box to write results in.

Warning

This function is not yet implemented correctly. Miter points of paths are not taken into account. If a path is the outmost object of your cell *and* it is not parallel to one of the coordinate axes, the calculated bounding box size might be off. In other cases it should be reasonable close to the real bounding box.

Definition at line 148 of file bounding-box.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.6.4.7 bounding_box_update_with_point()

```
void bounding_box_update_with_point (
            union bounding_box * destination,
            conv_generic_to_vector_2d_t conv_func,
            void * pt )
```

Update bounding box with a point.

Parameters

destination	Bounding box to update
conv_func	Conversion function to convert `pt` to a vector_2d. May be NULL
pt	Point to update bounding box with

Definition at line 174 of file bounding-box.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.6.4.8 calculate_cell_bounding_box()

```
void calculate_cell_bounding_box (
            union bounding_box * box,
            struct gds_cell * cell )
```

Calculate bounding box of a gds cell.

This function updates a given bounding box with the dimensions of a gds_cell. Please note that the handling of path miter points is not complete yet. If a path object is the outmost object of your cell at any edge, the resulting bounding box might be the wrong size. The devistion from the real size is guaranteed to be within the width of the path object.

Parameters

box	Resulting boundig box. Will be updated and not overwritten
cell	Toplevel cell

Warning

Handling of Path graphic objects not yet implemented correctly.

Definition at line 80 of file cell-geometrics.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.6.4.9 calculate_path_miter_points()

```
static void calculate_path_miter_points (
            struct vector_2d * a,
```

```
            struct vector_2d * b,
            struct vector_2d * c,
            struct vector_2d * m1,
            struct vector_2d * m2,
            double width )  [static]
```

Calculate path miter points for a pathwith a `width` and the anchors `a b c`.

Parameters

in	*a*	
in	*b*	
in	*c*	
out	*m1*	
out	*m2*	
in	*width*	

Returns

Miter points in `m1` and `m2`

Note

This function is currently unused (and untested). Ignore any compiler warning regarding this function.

Definition at line 105 of file bounding-box.c.

Here is the call graph for this function:

11.6.4.10 convert_gds_point_to_2d_vector()

```
static void convert_gds_point_to_2d_vector (
            struct gds_point * pt,
            struct vector_2d * vector )  [static]
```

Definition at line 35 of file cell-geometrics.c.

Here is the caller graph for this function:

11.6.4.11 update_box_with_gfx()

```
static void update_box_with_gfx (
            union bounding_box * box,
            struct gds_graphics * gfx )  [static]
```

Update the given bounding box with the bounding box of a graphics element.

Parameters

box	box to update
gfx	Graphics element

Definition at line 46 of file cell-geometrics.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.6.4.12 vector_2d_abs()

```
double vector_2d_abs (
            struct vector_2d * vec )
```

Definition at line 114 of file vector-operations.c.

Here is the caller graph for this function:

11.6.4.13 vector_2d_add()

```
void vector_2d_add (
            struct vector_2d * res,
            struct vector_2d * a,
            struct vector_2d * b )
```

Definition at line 142 of file vector-operations.c.

Here is the caller graph for this function:

11.6.4.14 vector_2d_alloc()

```
struct vector_2d * vector_2d_alloc (
            void  )
```

Definition at line 94 of file vector-operations.c.

Here is the caller graph for this function:

11.6.4.15 vector_2d_calculate_angle_between()

```
double vector_2d_calculate_angle_between (
            struct vector_2d * a,
            struct vector_2d * b )
```

Definition at line 123 of file vector-operations.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.6.4.16 vector_2d_copy()

```
struct vector_2d * vector_2d_copy (
            struct vector_2d * opt_res,
            struct vector_2d * vec )
```

Definition at line 75 of file vector-operations.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.6.4.17 vector_2d_free()

```
void vector_2d_free (
            struct vector_2d * vec )
```

Definition at line 99 of file vector-operations.c.

11.6.4.18 vector_2d_normalize()

```
void vector_2d_normalize (
            struct vector_2d * vec )
```

Definition at line 46 of file vector-operations.c.

Here is the caller graph for this function:

11.6.4.19 vector_2d_rotate()

```
void vector_2d_rotate (
            struct vector_2d * vec,
            double angle )
```

Definition at line 57 of file vector-operations.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.6.4.20 vector_2d_scalar_multipy()

```
double vector_2d_scalar_multipy (
            struct vector_2d * a,
            struct vector_2d * b )
```

Definition at line 38 of file vector-operations.c.

Here is the caller graph for this function:

11.6.4.21 vector_2d_scale()

```
void vector_2d_scale (
            struct vector_2d * vec,
            double scale )
```

Definition at line 105 of file vector-operations.c.

Here is the caller graph for this function:

11.6.4.22 vector_2d_subtract()

```
void vector_2d_subtract (
            struct vector_2d * res,
            struct vector_2d * a,
            struct vector_2d * b )
```

Definition at line 134 of file vector-operations.c.

Here is the caller graph for this function:

11.7 Graphical User Interface

Collaboration diagram for Graphical User Interface:

Modules

- LayerSelector Object
- LibCellRenderer GObject
- Custom GTK Widgets

Data Structures

- struct gui_button_states
- struct _GdsRenderGui

Macros

- #define RENDERER_TYPE_GUI (gds_render_gui_get_type())

Enumerations

- enum cell_store_columns { CELL_SEL_LIBRARY = 0 , CELL_SEL_CELL , CELL_SEL_CELL_ERROR_STATE , CELL_SEL_COLUMN_COUNT }

  Columns of selection tree view.
- enum gds_render_gui_signal_sig_ids { SIGNAL_WINDOW_CLOSED = 0 , SIGNAL_COUNT }

Functions

- static gboolean on_window_close (gpointer window, GdkEvent *event, gpointer user)

  Main window close event.
- static gboolean tree_sel_func (GtkTreeSelection *selection, GtkTreeModel *model, GtkTreePath *path, gboolean path_currently_selected, gpointer data)

  This function only allows valid cells to be selected.
- static void cell_tree_view_change_filter (GtkWidget *entry, gpointer data)

  Trigger refiltering of cell filter.
- static gboolean cell_store_filter_visible_func (GtkTreeModel *model, GtkTreeIter *iter, gpointer data)

  cell_store_filter_visible_func Decides whether an element of the tree model `model` is visible.
- int gds_render_gui_setup_cell_selector (GdsRenderGui *self)

  Setup a GtkTreeView with the necessary columns.
- static void on_load_gds (gpointer button, gpointer user)

  Callback function of Load GDS button.
- static void process_button_state_changes (GdsRenderGui *self)
- static void on_auto_color_clicked (gpointer button, gpointer user)

  Callback for auto coloring button.
- static void async_rendering_finished_callback (GdsOutputRenderer *renderer, gpointer gui)
- static void async_rendering_status_update_callback (GdsOutputRenderer *renderer, const char *status_↩ message, gpointer data)
- static void on_convert_clicked (gpointer button, gpointer user)

  Convert button callback.
- static void cell_tree_view_activated (gpointer tree_view, GtkTreePath *path, GtkTreeViewColumn *column, gpointer user)

  cell_tree_view_activated Callback for 'double click' on cell selector element
- static void cell_selection_changed (GtkTreeSelection *sel, GdsRenderGui *self)

  Callback for cell-selection change event.
- static void sort_up_callback (GtkWidget *widget, gpointer user)
- static void sort_down_callback (GtkWidget *widget, gpointer user)
- static void gds_render_gui_dispose (GObject *gobject)
- static void gds_render_gui_class_init (GdsRenderGuiClass *klass)
- static void on_select_all_layers_clicked (GtkWidget *button, gpointer user_data)

  Callback for the 'select all layers'-button.
- static gboolean auto_naming_ask_for_override (GdsRenderGui *gui)
- static void auto_naming_clicked (GtkWidget *button, gpointer user_data)
- GtkWindow * gds_render_gui_get_main_window (GdsRenderGui *gui)

  Get main window.
- static void gds_render_gui_init (GdsRenderGui *self)
- GdsRenderGui * gds_render_gui_new ()

  Create new GdsRenderGui Object.
- G_BEGIN_DECLS G_DECLARE_FINAL_TYPE (GdsRenderGui, gds_render_gui, RENDERER, GUI, GObject)

Variables

- static guint gds_render_gui_signals [SIGNAL_COUNT]

11.7.1 Detailed Description

11.7.2 Macro Definition Documentation

11.7.2.1 RENDERER_TYPE_GUI

```
#define RENDERER_TYPE_GUI (gds_render_gui_get_type())
```

Definition at line 40 of file gds-render-gui.h.

11.7.3 Enumeration Type Documentation

11.7.3.1 cell_store_columns

```
enum cell_store_columns
```

Columns of selection tree view.

Enumerator

CELL_SEL_LIBRARY	
CELL_SEL_CELL	
CELL_SEL_CELL_ERROR_STATE	Used for cell color and selectability
CELL_SEL_COLUMN_COUNT	Not a column. Used to determine count of columns.

Definition at line 47 of file gds-render-gui.c.

11.7.3.2 gds_render_gui_signal_sig_ids

```
enum gds_render_gui_signal_sig_ids
```

Enumerator

SIGNAL_WINDOW_CLOSED	
SIGNAL_COUNT	

Definition at line 54 of file gds-render-gui.c.

11.7.4 Function Documentation

11.7.4.1 async_rendering_finished_callback()

```
static void async_rendering_finished_callback (
            GdsOutputRenderer * renderer,
            gpointer gui )  [static]
```

Definition at line 395 of file gds-render-gui.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.7.4.2 async_rendering_status_update_callback()

```
static void async_rendering_status_update_callback (
            GdsOutputRenderer * renderer,
            const char * status_message,
            gpointer data )  [static]
```

Definition at line 408 of file gds-render-gui.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.7.4.3 auto_naming_ask_for_override()

```
static gboolean auto_naming_ask_for_override (
            GdsRenderGui * gui )  [static]
```

Definition at line 694 of file gds-render-gui.c.

Here is the caller graph for this function:

11.7.4.4 auto_naming_clicked()

```
static void auto_naming_clicked (
            GtkWidget * button,
            gpointer user_data )  [static]
```

Definition at line 720 of file gds-render-gui.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.7.4.5 cell_selection_changed()

```
static void cell_selection_changed (
            GtkTreeSelection * sel,
            GdsRenderGui * self )  [static]
```

Callback for cell-selection change event.

This function activates/deactivates the convert button depending on whether a cell is selected for conversion or not

Parameters

sel	
self	

Definition at line 594 of file gds-render-gui.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.7.4.6 cell_store_filter_visible_func()

```
static gboolean cell_store_filter_visible_func (
             GtkTreeModel * model,
             GtkTreeIter * iter,
             gpointer data )  [static]
```

cell_store_filter_visible_func Decides whether an element of the tree model `model` is visible.

Parameters

model	Tree model
iter	Current element / iter in Model to check
data	Data. Set to static stores variable

Returns

TRUE if visible, else FALSE

Note

TODO: Maybe implement Damerau-Levenshtein distance matching

Definition at line 174 of file gds-render-gui.c.

Here is the caller graph for this function:

11.7.4.7 cell_tree_view_activated()

```
static void cell_tree_view_activated (
             gpointer tree_view,
             GtkTreePath * path,
             GtkTreeViewColumn * column,
             gpointer user )  [static]
```

cell_tree_view_activated Callback for 'double click' on cell selector element

Parameters

tree_view	The tree view the event occured in
path	path to the selected row
column	The clicked column
user	pointer to GdsRenderGui object

Definition at line 576 of file gds-render-gui.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.7.4.8 cell_tree_view_change_filter()

```
static void cell_tree_view_change_filter (
            GtkWidget * entry,
            gpointer data )  [static]
```

Trigger refiltering of cell filter.

Parameters

entry	Unused widget, that emitted the signal
data	GdsrenderGui self instance

Definition at line 158 of file gds-render-gui.c.

Here is the caller graph for this function:

11.7.4.9 G_DECLARE_FINAL_TYPE()

```
G_BEGIN_DECLS G_DECLARE_FINAL_TYPE (
            GdsRenderGui ,
            gds_render_gui ,
            RENDERER ,
            GUI ,
            GObject  )
```

11.7.4.10 gds_render_gui_class_init()

```
static void gds_render_gui_class_init (
            GdsRenderGuiClass * klass )  [static]
```

Definition at line 662 of file gds-render-gui.c.

Here is the call graph for this function:

11.7.4.11 gds_render_gui_dispose()

```
static void gds_render_gui_dispose (
            GObject * gobject )  [static]
```

Definition at line 631 of file gds-render-gui.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.7.4.12 gds_render_gui_get_main_window()

```
GtkWindow * gds_render_gui_get_main_window (
            GdsRenderGui * gui )
```

Get main window.

This function returns the main window of the GUI, which can later be displayed. All handling of hte GUI is taken care of inside the GdsRenderGui Object

Returns

The generated main window

Definition at line 739 of file gds-render-gui.c.

Here is the caller graph for this function:

11.7.4.13 gds_render_gui_init()

```
static void gds_render_gui_init (
            GdsRenderGui * self )  [static]
```

Definition at line 744 of file gds-render-gui.c.

Here is the call graph for this function:

11.7.4.14 gds_render_gui_new()

```
GdsRenderGui * gds_render_gui_new ( )
```

Create new GdsRenderGui Object.

Returns

New object

Definition at line 847 of file gds-render-gui.c.

Here is the caller graph for this function:

11.7.4.15 gds_render_gui_setup_cell_selector()

```
int gds_render_gui_setup_cell_selector (
            GdsRenderGui * self )
```

Setup a GtkTreeView with the necessary columns.

Parameters

self	Current GUI object

Definition at line 218 of file gds-render-gui.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.7.4.16 on_auto_color_clicked()

```
static void on_auto_color_clicked (
            gpointer button,
            gpointer user )  [static]
```

Callback for auto coloring button.

Parameters

button	
user	

Definition at line 386 of file gds-render-gui.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.7.4.17 on_convert_clicked()

```
static void on_convert_clicked (
            gpointer button,
            gpointer user )  [static]
```

Convert button callback.

Parameters

button	
user	

Definition at line 425 of file gds-render-gui.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.7.4.18 on_load_gds()

```
static void on_load_gds (
            gpointer button,
            gpointer user )  [static]
```

Callback function of Load GDS button.

Parameters

button	
user	GdsRenderGui instance

Definition at line 263 of file gds-render-gui.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.7.4.19 on_select_all_layers_clicked()

```
static void on_select_all_layers_clicked (
            GtkWidget * button,
            gpointer user_data )  [static]
```

Callback for the 'select all layers'-button.

Parameters

button	Button that triggered the event
user_data	the GdsrenderGui object containing the main-window the button is placed in

Definition at line 685 of file gds-render-gui.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.7.4.20 on_window_close()

```
static gboolean on_window_close (
            gpointer window,
            GdkEvent * event,
            gpointer user )  [static]
```

Main window close event.

Parameters

window	GtkWindow which is closed
event	unused event
user	GdsRenderGui instance

Returns

Status of the event handling. Always true.

Definition at line 95 of file gds-render-gui.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.7.4.21 process_button_state_changes()

```
static void process_button_state_changes (
            GdsRenderGui * self )  [static]
```

Definition at line 364 of file gds-render-gui.c.

Here is the caller graph for this function:

11.7.4.22 sort_down_callback()

```
static void sort_down_callback (
            GtkWidget * widget,
            gpointer user )  [static]
```

Definition at line 620 of file gds-render-gui.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.7.4.23 sort_up_callback()

```
static void sort_up_callback (
            GtkWidget * widget,
            gpointer user )  [static]
```

Definition at line 609 of file gds-render-gui.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.7.4.24 tree_sel_func()

```
static gboolean tree_sel_func (
            GtkTreeSelection * selection,
            GtkTreeModel * model,
            GtkTreePath * path,
            gboolean path_currently_selected,
            gpointer data )  [static]
```

This function only allows valid cells to be selected.

Parameters

selection	
model	
path	
path_currently_selected	
data	

Returns

TRUE if element is selectable, FALSE if not

Definition at line 126 of file gds-render-gui.c.

Here is the caller graph for this function:

11.7.5 Variable Documentation

11.7.5.1 gds_render_gui_signals

```
guint gds_render_gui_signals[SIGNAL_COUNT]  [static]
```

Definition at line 56 of file gds-render-gui.c.

11.8 LaTeX / TikZ Renderer

Collaboration diagram for LaTeX / TikZ Renderer:

Data Structures

- struct _LatexRenderer

  Struct representing the LaTeX-Renderer object.

Macros

- #define GDS_RENDER_TYPE_LATEX_RENDERER (latex_renderer_get_type())
- #define LATEX_LINE_BUFFER_KB (10)

  Buffer for LaTeX Code line in KiB.
- #define WRITEOUT_BUFFER(buff) fwrite((buff)->str, sizeof(char), (buff)->len, tex_file)

  Writes a GString `buffer` to the fixed file tex_file.

Enumerations

- enum { PROP_STANDALONE = 1 , PROP_PDF_LAYERS , N_PROPERTIES }

Functions

- LatexRenderer * latex_renderer_new ()

  Create new LatexRenderer object.
- LatexRenderer * latex_renderer_new_with_options (gboolean pdf_layers, gboolean standalone)

  Create new LatexRenderer object.
- static void write_layer_definitions (FILE *tex_file, GList *layer_infos, GString *buffer)

  Write the layer declarration to TeX file.
- static gboolean write_layer_env (FILE *tex_file, GdkRGBA *color, int layer, GList *linfo, GString *buffer)

  Write layer Envirmonment.
- static void generate_graphics (FILE *tex_file, GList *graphics, GList *linfo, GString *buffer, double scale)

  Writes a graphics object to the specified tex_file.
- static void render_cell (struct gds_cell *cell, GList *layer_infos, FILE *tex_file, GString *buffer, double scale, GdsOutputRenderer *renderer)

  Render cell to file.
- static int latex_render_cell_to_code (struct gds_cell *cell, GList *layer_infos, FILE *tex_file, double scale, gboolean create_pdf_layers, gboolean standalone_document, GdsOutputRenderer *renderer)
- static int latex_renderer_render_output (GdsOutputRenderer *renderer, struct gds_cell *cell, double scale)
- static void latex_renderer_init (LatexRenderer *self)
- static void latex_renderer_get_property (GObject *obj, guint property_id, GValue *value, GParamSpec *pspec)
- static void latex_renderer_set_property (GObject *obj, guint property_id, const GValue *value, GParamSpec *pspec)
- static void latex_renderer_class_init (LatexRendererClass *klass)

Variables

- static GParamSpec * latex_renderer_properties [N_PROPERTIES] = {NULL}

11.8.1 Detailed Description

This is the class implementing the LaTeX / TikZ output rendering

11.8.2 Properties

This class inherits all properties from its parent GDS Output Renderer base class. In addition to that, it implements the following properties:

Property Name	Description
standalone	Configure output LaTeX document to be standalone compilable (requires standalone documentclass)
pdf-layers	Create OCG layers in LaTeX output

11.8.3 Macro Definition Documentation

11.8.3.1 GDS_RENDER_TYPE_LATEX_RENDERER

```
#define GDS_RENDER_TYPE_LATEX_RENDERER (latex_renderer_get_type())
```

Definition at line 41 of file latex-renderer.h.

11.8.3.2 LATEX_LINE_BUFFER_KB

```
#define LATEX_LINE_BUFFER_KB (10)
```

Buffer for LaTeX Code line in KiB.

Definition at line 46 of file latex-renderer.h.

11.8.3.3 WRITEOUT_BUFFER

```
#define WRITEOUT_BUFFER(
             buff ) fwrite((buff)->str, sizeof(char), (buff)->len, tex_file)
```

Writes a GString `buffer` to the fixed file tex_file.

Note

This is a convinience macro. Do not use this anywhere else. It might change behavior in futurtre releases

Definition at line 60 of file latex-renderer.c.

11.8.4 Enumeration Type Documentation

11.8.4.1 anonymous enum

```
anonymous enum
```

Enumerator

PROP_STANDALONE	
PROP_PDF_LAYERS	
N_PROPERTIES	

Definition at line 50 of file latex-renderer.c.

11.8.5 Function Documentation

11.8.5.1 generate_graphics()

```
static void generate_graphics (
            FILE * tex_file,
            GList * graphics,
            GList * linfo,
            GString * buffer,
            double scale )  [static]
```

Writes a graphics object to the specified tex_file.

This function opens the layer, writes a graphics object and closes the layer

Parameters

tex_file	File to write to
graphics	Object to render
linfo	Layer information
buffer	Working buffer
scale	Scale abject down by this value

Definition at line 166 of file latex-renderer.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.8.5.2 latex_render_cell_to_code()

```
static int latex_render_cell_to_code (
            struct gds_cell * cell,
            GList * layer_infos,
            FILE * tex_file,
            double scale,
            gboolean create_pdf_layers,
            gboolean standalone_document,
            GdsOutputRenderer * renderer )  [static]
```

Definition at line 295 of file latex-renderer.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.8.5.3 latex_renderer_class_init()

```
static void latex_renderer_class_init (
            LatexRendererClass * klass )  [static]
```

Definition at line 424 of file latex-renderer.c.

Here is the call graph for this function:

11.8.5.4 latex_renderer_get_property()

```
static void latex_renderer_get_property (
             GObject * obj,
             guint property_id,
             GValue * value,
             GParamSpec * pspec )  [static]
```

Definition at line 388 of file latex-renderer.c.

Here is the caller graph for this function:

11.8.5.5 latex_renderer_init()

```
static void latex_renderer_init (
             LatexRenderer * self )  [static]
```

Definition at line 382 of file latex-renderer.c.

11.8.5.6 latex_renderer_new()

```
LatexRenderer * latex_renderer_new ( )
```

Create new LatexRenderer object.

Returns

New object

Definition at line 452 of file latex-renderer.c.

11.8.5.7 latex_renderer_new_with_options()

```
LatexRenderer * latex_renderer_new_with_options (
             gboolean pdf_layers,
             gboolean standalone )
```

Create new LatexRenderer object.

This function sets the 'pdf-layers' and 'standalone' properties for the newly created object.

They can later be changes by modifying the properties again. On top of that, The options can be changed in the resulting LaTeX output file if needed.

Parameters

pdf_layers	If PDF OCR layers should be enabled
standalone	If output TeX file should be standalone compilable

Returns

New object

Definition at line 457 of file latex-renderer.c.

Here is the caller graph for this function:

11.8.5.8 latex_renderer_render_output()

```
static int latex_renderer_render_output (
            GdsOutputRenderer * renderer,
            struct gds_cell * cell,
            double scale )  [static]
```

Definition at line 349 of file latex-renderer.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.8.5.9 latex_renderer_set_property()

```
static void latex_renderer_set_property (
            GObject * obj,
            guint property_id,
            const GValue * value,
            GParamSpec * pspec )  [static]
```

Definition at line 405 of file latex-renderer.c.

Here is the caller graph for this function:

11.8.5.10 render_cell()

```
static void render_cell (
            struct gds_cell * cell,
            GList * layer_infos,
            FILE * tex_file,
            GString * buffer,
            double scale,
            GdsOutputRenderer * renderer )  [static]
```

Render cell to file.

Parameters

cell	Cell to render
layer_infos	Layer information
tex_file	File to write to
buffer	Working buffer
scale	Scale output down by this value
renderer	The current renderer as GdsOutputRenderer. This is used to emit the status updates to the GUI

Definition at line 245 of file latex-renderer.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.8.5.11 write_layer_definitions()

```
static void write_layer_definitions (
            FILE * tex_file,
            GList * layer_infos,
            GString * buffer )  [static]
```

Write the layer declarration to TeX file.

This writes the declaration of the layers and the mapping in which order the layers shall be rendered by TikZ. Layers are written in the order they are positioned inside the `layer_infos` list.

Parameters

tex_file	TeX-File to write to
layer_infos	List containing layer_info structs.
buffer	

Note

The field layer_info::stacked_position is ignored. Stack depends on list order.

Definition at line 74 of file latex-renderer.c.

Here is the caller graph for this function:

11.8.5.12 write_layer_env()

```
static gboolean write_layer_env (
            FILE * tex_file,
            GdkRGBA * color,
            int layer,
            GList * linfo,
            GString * buffer )  [static]
```

Write layer Envirmonment.

If the requested layer shall be rendered, this code writes the necessary code to open the layer. It also returns the color the layer shall be rendered in.

The followingenvironments are generated:

```
\begin{pgfonlayer}{<layer>}
% If pdf layers shall be used also this is enabled:
\begin{scope}[ocg={ref=<layer>, status=visible,name={<Layer Name>}}]
```

If the layer shall not be rendered, FALSE is returned and the color is not filled in and the cod eis not written to the file.

Parameters

tex_file	TeX file to write to
color	Return of the layer's color
layer	Requested layer number
linfo	Layer information list containing layer_info structs
buffer	Some working buffer

Returns

TRUE, if the layer shall be rendered.

Note

The opened environments have to be closed afterwards

Definition at line 133 of file latex-renderer.c.

Here is the caller graph for this function:

11.8.6 Variable Documentation

11.8.6.1 latex_renderer_properties

```
GParamSpec* latex_renderer_properties[N_PROPERTIES] = {NULL}  [static]
```

Definition at line 422 of file latex-renderer.c.

11.9 LayerSelector Object

Collaboration diagram for LayerSelector Object:

Data Structures

- struct _LayerSelector

Macros

- #define TYPE_LAYER_SELECTOR (layer_selector_get_type())

Enumerations

- enum layer_selector_sort_algo { LAYER_SELECTOR_SORT_DOWN = 0 , LAYER_SELECTOR_SORT_UP }

  Defines how to sort the layer selector list box.

Functions

- G_BEGIN_DECLS G_DECLARE_FINAL_TYPE (LayerSelector, layer_selector, LAYER, SELECTOR, GObject)
- LayerSelector * layer_selector_new (GtkListBox *list_box)

  layer_selector_new
- void layer_selector_generate_layer_widgets (LayerSelector *selector, GList *libs)

  Generate layer widgets in in the LayerSelector instance.
- void layer_selector_set_load_mapping_button (LayerSelector *selector, GtkWidget *button, GtkWindow *main_window)

  Supply button for loading the layer mapping.
- void layer_selector_set_save_mapping_button (LayerSelector *selector, GtkWidget *button, GtkWindow *main_window)

  Supply button for saving the layer mapping.
- LayerSettings * layer_selector_export_rendered_layer_info (LayerSelector *selector)

  Get a list of all layers that shall be exported when rendering the cells.
- void layer_selector_force_sort (LayerSelector *selector, enum layer_selector_sort_algo sort_function)

  Force the layer selector list to be sorted according to `sort_function`.
- void layer_selector_select_all_layers (LayerSelector *layer_selector, gboolean select)

  Set 'export' value of all layers in the LayerSelector to the supplied select value.
- void layer_selector_auto_color_layers (LayerSelector *layer_selector, ColorPalette *palette, double global_alpha)

  Apply colors from palette to all layers. Aditionally set alpha.
- void layer_selector_auto_name_layers (LayerSelector *layer_selector, gboolean overwrite)

  Auto name all layers in the layer selector.
- gboolean layer_selector_contains_elements (LayerSelector *layer_selector)

  Check if the given layer selector contains layer elements.
- size_t layer_selector_num_of_named_elements (LayerSelector *layer_selector)

  Get number of layer elements that are named.
- static void sel_layer_element_drag_begin (GtkWidget *widget, GdkDragContext *context, gpointer data)
- static void sel_layer_element_drag_end (GtkWidget *widget, GdkDragContext *context, gpointer data)
- static void sel_layer_element_drag_data_get (GtkWidget *widget, GdkDragContext *context, GtkSelectionData *selection_data, guint info, guint time, gpointer data)
- static GtkListBoxRow * layer_selector_get_last_row (GtkListBox *list)
- static GtkListBoxRow * layer_selector_get_row_before (GtkListBox *list, GtkListBoxRow *row)
- static GtkListBoxRow * layer_selector_get_row_after (GtkListBox *list, GtkListBoxRow *row)
- static void layer_selector_drag_data_received (GtkWidget *widget, GdkDragContext *context, gint x, gint y, GtkSelectionData *selection_data, guint info, guint32 time, gpointer data)
- static gboolean layer_selector_drag_motion (GtkWidget *widget, GdkDragContext *context, int x, int y, guint time)
- static void layer_selector_drag_leave (GtkWidget *widget, GdkDragContext *context, guint time)
- static void layer_selector_dispose (GObject *self)
- static void layer_selector_class_init (LayerSelectorClass *klass)
- static void layer_selector_setup_dnd (LayerSelector *self)
- static void layer_selector_init (LayerSelector *self)
- static void layer_selector_clear_widgets (LayerSelector *self)

- static gboolean layer_selector_check_if_layer_widget_exists (LayerSelector *self, int layer)
  Check if a specific layer element with the given layer number is present in the layer selector.
- static void sel_layer_element_setup_dnd_callbacks (LayerSelector *self, LayerElement *element)
  Setup the necessary drag and drop callbacks of layer elements.
- static void layer_selector_analyze_cell_layers (LayerSelector *self, struct gds_cell *cell)
  Analyze `cell` layers and append detected layers to layer selector `self`.
- static gint layer_selector_sort_func (GtkListBoxRow *row1, GtkListBoxRow *row2, gpointer unused)
  sort_func Sort callback for list box
- static LayerElement * layer_selector_find_layer_element_in_list (GList *el_list, int layer)
  Find LayerElement in list with specified layer number.
- static void layer_selector_load_layer_mapping_from_file (LayerSelector *self, const gchar *file_name)
  Load the layer mapping from a CSV formatted file.
- static void layer_selector_load_mapping_clicked (GtkWidget *button, gpointer user_data)
  Callback for Load Mapping Button.
- static void layer_selector_save_layer_mapping_data (LayerSelector *self, const gchar *file_name)
  Save layer mapping of selector `self` to a file.
- static void layer_selector_save_mapping_clicked (GtkWidget *button, gpointer user_data)
  Callback for Save Layer Mapping Button.

Variables

- static const char * dnd_additional_css

11.9.1 Detailed Description

This objects implements the layer selector and displays the layers in a list box. It uses LayerElement objects to display the individual layers inside the list box.

11.9.2 Macro Definition Documentation

11.9.2.1 TYPE_LAYER_SELECTOR

```
#define TYPE_LAYER_SELECTOR (layer_selector_get_type())
```

Definition at line 43 of file layer-selector.h.

11.9.3 Enumeration Type Documentation

11.9.3.1 layer_selector_sort_algo

```
enum layer_selector_sort_algo
```

Defines how to sort the layer selector list box.

Enumerator

LAYER_SELECTOR_SORT_DOWN	
LAYER_SELECTOR_SORT_UP	

Definition at line 48 of file layer-selector.h.

11.9.4 Function Documentation

11.9.4.1 G_DECLARE_FINAL_TYPE()

```
G_BEGIN_DECLS G_DECLARE_FINAL_TYPE (
             LayerSelector ,
             layer_selector ,
             LAYER ,
             SELECTOR ,
             GObject  )
```

11.9.4.2 layer_selector_analyze_cell_layers()

```
static void layer_selector_analyze_cell_layers (
             LayerSelector * self,
             struct gds_cell * cell )  [static]
```

Analyze `cell` layers and append detected layers to layer selector `self`.

Parameters

self	LayerSelector instance
cell	Cell to analyze

Definition at line 497 of file layer-selector.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.9.4.3 layer_selector_auto_color_layers()

```
void layer_selector_auto_color_layers (
             LayerSelector * layer_selector,
             ColorPalette * palette,
             double global_alpha )
```

Apply colors from palette to all layers. Aditionally set alpha.

Parameters

layer_selector	LayerSelector object
palette	Color palette to use
global_alpha	Additional alpha value that is applied to all layers. Must be > 0

Definition at line 816 of file layer-selector.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.9.4.4 layer_selector_auto_name_layers()

```
void layer_selector_auto_name_layers (
            LayerSelector * layer_selector,
            gboolean overwrite )
```

Auto name all layers in the layer selector.

This functions sets the name of the layer equal to its number. The `overwrite` parameter specifies if already set layer names are overwritten.

Parameters

layer_selector	LayerSelector
overwrite	Overwrite existing layer names

Definition at line 854 of file layer-selector.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.9.4.5 layer_selector_check_if_layer_widget_exists()

```
static gboolean layer_selector_check_if_layer_widget_exists (
            LayerSelector * self,
            int layer )  [static]
```

Check if a specific layer element with the given layer number is present in the layer selector.

Parameters

self	LayerSelector instance
layer	Layer number to check for

Returns

TRUE if layer is present, else FALSE

Definition at line 449 of file layer-selector.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.9.4.6 layer_selector_class_init()

```
static void layer_selector_class_init (
            LayerSelectorClass * klass )  [static]
```

Definition at line 334 of file layer-selector.c.

Here is the call graph for this function:

11.9.4.7 layer_selector_clear_widgets()

```
static void layer_selector_clear_widgets (
            LayerSelector * self )  [static]
```

Definition at line 423 of file layer-selector.c.

Here is the caller graph for this function:

11.9.4.8 layer_selector_contains_elements()

```
gboolean layer_selector_contains_elements (
            LayerSelector * layer_selector )
```

Check if the given layer selector contains layer elements.

This function checks whether there are elements present. If an invalid object pointer `layer_selector` is passed, the function returns FALSE

Parameters

in	*layer_selector*	Selector to check

Returns

True, if there is at least one layer present inside the selector

Definition at line 884 of file layer-selector.c.

Here is the caller graph for this function:

11.9.4.9 layer_selector_dispose()

```
static void layer_selector_dispose (
            GObject * self )  [static]
```

Definition at line 315 of file layer-selector.c.

Here is the caller graph for this function:

11.9.4.10 layer_selector_drag_data_received()

```
static void layer_selector_drag_data_received (
            GtkWidget * widget,
            GdkDragContext * context,
            gint x,
            gint y,
            GtkSelectionData * selection_data,
            guint info,
            guint32 time,
            gpointer data )  [static]
```

Definition at line 152 of file layer-selector.c.

Here is the caller graph for this function:

11.9.4.11 layer_selector_drag_leave()

```
static void layer_selector_drag_leave (
            GtkWidget * widget,
            GdkDragContext * context,
            guint time )  [static]
```

Definition at line 259 of file layer-selector.c.

Here is the caller graph for this function:

11.9.4.12 layer_selector_drag_motion()

```
static gboolean layer_selector_drag_motion (
            GtkWidget * widget,
            GdkDragContext * context,
            int x,
            int y,
            guint time )  [static]
```

Definition at line 198 of file layer-selector.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.9.4.13 layer_selector_export_rendered_layer_info()

```
LayerSettings * layer_selector_export_rendered_layer_info (
            LayerSelector * selector )
```

Get a list of all layers that shall be exported when rendering the cells.

Parameters

selector	Layer selector instance

Returns

LayerSettings containing the layer information

Definition at line 388 of file layer-selector.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.9.4.14 layer_selector_find_layer_element_in_list()

```
static LayerElement * layer_selector_find_layer_element_in_list (
            GList * el_list,
            int layer )  [static]
```

Find LayerElement in list with specified layer number.

Parameters

el_list	List with elements of type LayerElement
layer	Layer number

Returns

Found LayerElement. If nothing is found, NULL.

Definition at line 578 of file layer-selector.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.9.4.15 layer_selector_force_sort()

```
void layer_selector_force_sort (
            LayerSelector * selector,
            enum layer_selector_sort_algo sort_function )
```

Force the layer selector list to be sorted according to `sort_function`.

Parameters

selector	LayerSelector instance
sort_function	The sorting method (up or down sorting)

Definition at line 779 of file layer-selector.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.9.4.16 layer_selector_generate_layer_widgets()

```
void layer_selector_generate_layer_widgets (
            LayerSelector * selector,
            GList * libs )
```

Generate layer widgets in in the LayerSelector instance.

Note

This clears all previously inserted elements

Parameters

selector	LayerSelector instance
libs	The libraries to add

Definition at line 549 of file layer-selector.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.9.4.17 layer_selector_get_last_row()

```
static GtkListBoxRow * layer_selector_get_last_row (
            GtkListBox * list )  [static]
```

Definition at line 119 of file layer-selector.c.

Here is the caller graph for this function:

11.9.4.18 layer_selector_get_row_after()

```
static GtkListBoxRow * layer_selector_get_row_after (
            GtkListBox * list,
            GtkListBoxRow * row )  [static]
```

Definition at line 144 of file layer-selector.c.

Here is the caller graph for this function:

11.9.4.19 layer_selector_get_row_before()

```
static GtkListBoxRow * layer_selector_get_row_before (
            GtkListBox * list,
            GtkListBoxRow * row )  [static]
```

Definition at line 136 of file layer-selector.c.

Here is the caller graph for this function:

11.9.4.20 layer_selector_init()

```
static void layer_selector_init (
            LayerSelector * self )  [static]
```

Definition at line 361 of file layer-selector.c.

11.9.4.21 layer_selector_load_layer_mapping_from_file()

```
static void layer_selector_load_layer_mapping_from_file (
            LayerSelector * self,
            const gchar * file_name )  [static]
```

Load the layer mapping from a CSV formatted file.

This function imports the layer specification from a file (see Layer Mapping File Specification). The layer ordering defined in the file is kept. All layers present in the current loaded library, which are not present in the layer mapping file are appended at the end of the layer selector list.

Parameters

self	LayerSelector instance
file_name	File name to load from

Definition at line 602 of file layer-selector.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.9.4.22 layer_selector_load_mapping_clicked()

```
static void layer_selector_load_mapping_clicked (
            GtkWidget * button,
            gpointer user_data )  [static]
```

Callback for Load Mapping Button.

Parameters

button	
user_data	

Definition at line 685 of file layer-selector.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.9.4.23 layer_selector_new()

```
LayerSelector * layer_selector_new (
            GtkListBox * list_box )
```

layer_selector_new

Parameters

list_box	The associated list box, the content is displayed in

Returns

Newly created layer selector

Definition at line 373 of file layer-selector.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.9.4.24 layer_selector_num_of_named_elements()

```
size_t layer_selector_num_of_named_elements (
            LayerSelector * layer_selector )
```

Get number of layer elements that are named.

Parameters

in	*layer_selector*	Layer selector

Returns

Number of layers with a name != NULL or != ""

Definition at line 899 of file layer-selector.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.9.4.25 layer_selector_save_layer_mapping_data()

```
static void layer_selector_save_layer_mapping_data (
            LayerSelector * self,
            const gchar * file_name )  [static]
```

Save layer mapping of selector `self` to a file.

Parameters

self	LayerSelector instance
file_name	File name to save to

Definition at line 714 of file layer-selector.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.9.4.26 layer_selector_save_mapping_clicked()

```
static void layer_selector_save_mapping_clicked (
            GtkWidget * button,
            gpointer user_data )  [static]
```

Callback for Save Layer Mapping Button.

Parameters

button	
user_data	

Definition at line 731 of file layer-selector.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.9.4.27 layer_selector_select_all_layers()

```
void layer_selector_select_all_layers (
            LayerSelector * layer_selector,
            gboolean select )
```

Set 'export' value of all layers in the LayerSelector to the supplied select value.

Parameters

layer_selector	LayerSelector object
select	

Definition at line 796 of file layer-selector.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.9.4.28 layer_selector_set_load_mapping_button()

```
void layer_selector_set_load_mapping_button (
            LayerSelector * selector,
            GtkWidget * button,
            GtkWindow * main_window )
```

Supply button for loading the layer mapping.

Parameters

selector	LayerSelector instance
button	Load button. Will be referenced
main_window	Parent window for dialogs. Will be referenced

Definition at line 755 of file layer-selector.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.9.4.29 layer_selector_set_save_mapping_button()

```
void layer_selector_set_save_mapping_button (
            LayerSelector * selector,
```

```
            GtkWidget * button,
            GtkWindow * main_window )
```

Supply button for saving the layer mapping.

Parameters

selector	LayerSelector instance
button	Save button. Will be refeneced
main_window	Parent window for dialogs. Will be referenced

Definition at line 767 of file layer-selector.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.9.4.30 layer_selector_setup_dnd()

```
static void layer_selector_setup_dnd (
            LayerSelector * self )  [static]
```

Definition at line 350 of file layer-selector.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.9.4.31 layer_selector_sort_func()

```
static gint layer_selector_sort_func (
            GtkListBoxRow * row1,
            GtkListBoxRow * row2,
            gpointer unused )  [static]
```

sort_func Sort callback for list box

Parameters

row1	
row2	
unused	

Note

Do not use this function. This is an internal callback

Returns

See sort function documentation of GTK+

Definition at line 525 of file layer-selector.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.9.4.32 sel_layer_element_drag_begin()

```
static void sel_layer_element_drag_begin (
            GtkWidget * widget,
            GdkDragContext * context,
            gpointer data )  [static]
```

Definition at line 62 of file layer-selector.c.

Here is the caller graph for this function:

11.9.4.33 sel_layer_element_drag_data_get()

```
static void sel_layer_element_drag_data_get (
            GtkWidget * widget,
            GdkDragContext * context,
            GtkSelectionData * selection_data,
            guint info,
            guint time,
            gpointer data )  [static]
```

Definition at line 103 of file layer-selector.c.

Here is the caller graph for this function:

11.9.4.34 sel_layer_element_drag_end()

```
static void sel_layer_element_drag_end (
            GtkWidget * widget,
            GdkDragContext * context,
            gpointer data )  [static]
```

Definition at line 91 of file layer-selector.c.

Here is the caller graph for this function:

11.9.4.35 sel_layer_element_setup_dnd_callbacks()

```
static void sel_layer_element_setup_dnd_callbacks (
            LayerSelector * self,
            LayerElement * element )  [static]
```

Setup the necessary drag and drop callbacks of layer elements.

Parameters

self	LayerSelector instance. Used to get the DnD target entry.
element	LayerElement instance to set the callbacks

Definition at line 476 of file layer-selector.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.9.5 Variable Documentation

11.9.5.1 dnd_additional_css

```
const char* dnd_additional_css  [static]
```

Definition at line 279 of file layer-selector.c.

11.10 LibCellRenderer GObject

Collaboration diagram for LibCellRenderer GObject:

Data Structures

- struct _LibCellRenderer

Macros

- #define TYPE_LIB_CELL_RENDERER (lib_cell_renderer_get_type())
- #define LIB_CELL_RENDERER_ERROR_WARN (1U<<0)
- #define LIB_CELL_RENDERER_ERROR_ERR (1U<<1)

Typedefs

- typedef struct _LibCellRenderer LibCellRenderer

Enumerations

- enum { PROP_LIB = 1 , PROP_CELL , PROP_ERROR_LEVEL , PROP_COUNT }

Functions

- void lib_cell_renderer_init (LibCellRenderer *self)
- static void lib_cell_renderer_constructed (GObject *obj)
- static void convert_error_level_to_color (GdkRGBA *color, unsigned int error_level)
- static void lib_cell_renderer_set_property (GObject *object, guint param_id, const GValue *value, GParam↩Spec *pspec)
- static void lib_cell_renderer_get_property (GObject *object, guint param_id, GValue *value, GParamSpec *pspec)
- void lib_cell_renderer_class_init (LibCellRendererClass *klass)
- GtkCellRenderer * lib_cell_renderer_new (void)

  Create a new renderer for renderering gds_cell and gds_library elements.
- GType lib_cell_renderer_get_type (void)

  lib_cell_renderer_get_type

Variables

- static GParamSpec * properties [PROP_COUNT]

11.10.1 Detailed Description

The LibCellRenderer Object is used to render gds_cell and gds_library elements to a GtkTreeView.

The LibCellRenderer class is derived from a GtkCellRendererText and works the same way. The additinal features are three new properties:

- *gds-lib*: This property can be used to set a gds_library structure. The renderer will render the name of the library.
- *gds-cell*: This property can be used to set a gds_cell structure. The renderer will render the name of the cell.
- *error-level*: Set the error level of the cell/library. This affects the foreground color of hte rendered output.

Internally the class operates by setting the 'text' property, which is inherited form the base class to the library/cell name (gds_library::name and gds_cell::name fields). The error level (LIB_CELL_RENDERER_ERROR_WARN and LIB_CELL_RENDERER_ERROR_ERR) is translated to the inherited 'foreground-rgba' property.

11.10.2 Macro Definition Documentation

11.10.2.1 LIB_CELL_RENDERER_ERROR_ERR

```
#define LIB_CELL_RENDERER_ERROR_ERR (1U<<1)
```

Definition at line 45 of file lib-cell-renderer.h.

11.10.2.2 LIB_CELL_RENDERER_ERROR_WARN

```
#define LIB_CELL_RENDERER_ERROR_WARN (1U<<0)
```

Error levels

Definition at line 44 of file lib-cell-renderer.h.

11.10.2.3 TYPE_LIB_CELL_RENDERER

```
#define TYPE_LIB_CELL_RENDERER (lib_cell_renderer_get_type())
```

Definition at line 39 of file lib-cell-renderer.h.

11.10.3 Typedef Documentation

11.10.3.1 LibCellRenderer

```
typedef struct _LibCellRenderer LibCellRenderer
```

11.10.4 Enumeration Type Documentation

11.10.4.1 anonymous enum

```
anonymous enum
```

Enumerator

PROP_LIB	Library to display the name of.
PROP_CELL	Cell to display the name of.
PROP_ERROR_LEVEL	Error level of cell/library for coloring.
PROP_COUNT	Sentinel.

Definition at line 36 of file lib-cell-renderer.c.

11.10.5 Function Documentation

11.10.5.1 convert_error_level_to_color()

```
static void convert_error_level_to_color (
            GdkRGBA * color,
            unsigned int error_level )  [static]
```

Definition at line 54 of file lib-cell-renderer.c.

Here is the caller graph for this function:

11.10.5.2 lib_cell_renderer_class_init()

```
void lib_cell_renderer_class_init (
            LibCellRendererClass * klass )
```

Definition at line 129 of file lib-cell-renderer.c.

Here is the call graph for this function:

11.10.5.3 lib_cell_renderer_constructed()

```
static void lib_cell_renderer_constructed (
            GObject * obj )  [static]
```

Definition at line 49 of file lib-cell-renderer.c.

Here is the caller graph for this function:

11.10.5.4 lib_cell_renderer_get_property()

```
static void lib_cell_renderer_get_property (
            GObject * object,
            guint param_id,
            GValue * value,
            GParamSpec * pspec )  [static]
```

Definition at line 113 of file lib-cell-renderer.c.

Here is the caller graph for this function:

11.10.5.5 lib_cell_renderer_get_type()

```
GType lib_cell_renderer_get_type (
            void  )
```

lib_cell_renderer_get_type

Returns

GObject Type

11.10.5.6 lib_cell_renderer_init()

```
void lib_cell_renderer_init (
            LibCellRenderer * self )
```

Definition at line 43 of file lib-cell-renderer.c.

11.10.5.7 lib_cell_renderer_new()

```
GtkCellRenderer * lib_cell_renderer_new (
            void  )
```

Create a new renderer for renderering gds_cell and gds_library elements.

Returns

New renderer object

Definition at line 149 of file lib-cell-renderer.c.

Here is the caller graph for this function:

11.10.5.8 lib_cell_renderer_set_property()

```
static void lib_cell_renderer_set_property (
            GObject * object,
            guint param_id,
            const GValue * value,
            GParamSpec * pspec )  [static]
```

Definition at line 78 of file lib-cell-renderer.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.10.6 Variable Documentation

11.10.6.1 properties

```
GParamSpec* properties[PROP_COUNT]  [static]
```

Definition at line 127 of file lib-cell-renderer.c.

11.11 External Renderer Plugins

Collaboration diagram for External Renderer Plugins:

Modules

- Example Plugin for External Renderer

11.11.1 Detailed Description

These plugins can be loaded with the External Shared Object Renderer

11.12 Custom GTK Widgets

Collaboration diagram for Custom GTK Widgets:

Modules

- Activity Bar
- RendererSettingsDialog
- LayerElement

11.12.1 Detailed Description

11.13 GDS-Utilities

Data Structures

- struct gds_cell_array_instance

  Struct representing an array instantiation.
- struct gds_point

  A point in the 2D plane. Sometimes referred to as vertex.
- struct gds_cell_checks

  Stores the result of the cell checks.
- struct gds_time_field

  Date information for cells and libraries.
- struct gds_graphics

  A GDS graphics object.
- struct gds_cell_instance

  This represents an instanc of a cell inside another cell.
- struct gds_cell

  A Cell inside a gds_library.
- struct gds_library

  GDS Toplevel library.

Macros

- #define GDS_DEFAULT_UNITS (10E-9)

  Default units assumed for library.
- #define GDS_ERROR(fmt, ...) printf("[PARSE_ERROR] " fmt "\n", ##__VA_ARGS__)

  Print GDS error.
- #define GDS_WARN(fmt, ...) printf("[PARSE_WARNING] " fmt "\n", ##__VA_ARGS__)

  Print GDS warning.
- #define GDS_INF(fmt, ...)
- #define GDS_PRINT_DEBUG_INFOS (0)

  1: Print infos, 0: Don't print
- #define CELL_NAME_MAX (100)

  Maximum length of a gds_cell::name or a gds_library::name.
- #define MIN(a, b) (((a) < (b)) ? (a) : (b))

  Return smaller number.
- #define MAX(a, b) (((a) > (b)) ? (a) : (b))

  Return bigger number.

Enumerations

- enum gds_record {
  INVALID = 0x0000 , HEADER = 0x0002 , BGNLIB = 0x0102 , LIBNAME = 0x0206 ,
  UNITS = 0x0305 , ENDLIB = 0x0400 , BGNSTR = 0x0502 , STRNAME = 0x0606 ,
  ENDSTR = 0x0700 , BOUNDARY = 0x0800 , PATH = 0x0900 , SREF = 0x0A00 ,
  ENDEL = 0x1100 , XY = 0x1003 , MAG = 0x1B05 , ANGLE = 0x1C05 ,
  SNAME = 0x1206 , STRANS = 0x1A01 , BOX = 0x2D00 , LAYER = 0x0D02 ,
  DATATYPE = 0x0E02 , WIDTH = 0x0F03 , PATHTYPE = 0x2102 , COLROW = 0x1302 ,
  AREF = 0x0B00 }
- enum { GDS_CELL_CHECK_NOT_RUN = -1 }

  Defintion of check counter default value that indicates that the corresponding check has not yet been executed.
- enum graphics_type { GRAPHIC_PATH = 0 , GRAPHIC_POLYGON = 1 , GRAPHIC_BOX = 2 }

  Types of graphic objects.
- enum path_type { PATH_FLUSH = 0 , PATH_ROUNDED = 1 , PATH_SQUARED = 2 }

  Defines the line caps of a path.

Functions

- static int name_cell_ref (struct gds_cell_instance ∗cell_inst, unsigned int bytes, char ∗data)

  Name cell reference.
- static int name_array_cell_ref (struct gds_cell_array_instance ∗cell_inst, unsigned int bytes, char ∗data)

  Name cell reference.
- static double gds_convert_double (const char ∗data)

  Convert GDS 8-byte real to double.
- static signed int gds_convert_signed_int (const char ∗data)

  Convert GDS INT32 to int.
- static int16_t gds_convert_signed_int16 (const char ∗data)

  Convert GDS INT16 to int16.
- static uint16_t gds_convert_unsigned_int16 (const char ∗data)

  Convert GDS UINT16 String to uint16.
- static GList ∗ append_library (GList ∗curr_list, struct gds_library ∗∗library_ptr)

  Append library to list.
- static GList ∗ prepend_graphics (GList ∗curr_list, enum graphics_type type, struct gds_graphics ∗∗graphics↩
  _ptr)

  Prepend graphics to list.
- static GList ∗ append_vertex (GList ∗curr_list, int x, int y)

  Appends vertext List.
- static GList ∗ append_cell (GList ∗curr_list, struct gds_cell ∗∗cell_ptr)

  append_cell Append a gds_cell to a list
- static GList ∗ append_cell_ref (GList ∗curr_list, struct gds_cell_instance ∗∗instance_ptr)

  Append a cell reference to the reference GList.
- static int name_library (struct gds_library ∗current_library, unsigned int bytes, char ∗data)

  Name a gds_library.
- static int name_cell (struct gds_cell ∗cell, unsigned int bytes, char ∗data, struct gds_library ∗lib)

  Names a gds_cell.
- static void parse_reference_list (gpointer gcell_ref, gpointer glibrary)

  Search for cell reference `gcell_ref` in `glibrary`.
- static void scan_cell_reference_dependencies (gpointer gcell, gpointer library)

  Scans cell references inside cell This function searches all the references in `gcell` and updates the gds_cell_instance::cell_ref field in each instance.
- static void scan_library_references (gpointer library_list_item, gpointer user)

Scans library's cell references.

- static void gds_parse_date (const char *buffer, int length, struct gds_time_field *mod_date, struct gds_time_field *access_date)

  gds_parse_date
- static void convert_aref_to_sref (struct gds_cell_array_instance *aref, struct gds_cell *container_cell)

  Convert AREF to a bunch of SREFs and append them to `container_cell`.
- int parse_gds_from_file (const char *filename, GList **library_array)

  Parse a GDS file.
- static void delete_cell_inst_element (struct gds_cell_instance *cell_inst)

  delete_cell_inst_element
- static void delete_vertex (struct gds_point *vertex)

  delete_vertex
- static void delete_graphics_obj (struct gds_graphics *gfx)

  delete_graphics_obj
- static void delete_cell_element (struct gds_cell *cell)

  delete_cell_element
- static void delete_library_element (struct gds_library *lib)

  delete_library_element
- int clear_lib_list (GList **library_list)

  Deletes all libraries including cells, references etc.
- int gds_tree_check_cell_references (struct gds_library *lib)

  gds_tree_check_cell_references checks if all child cell references can be resolved in the given library
- static int gds_tree_check_list_contains_cell (GList *list, struct gds_cell *cell)

  Check if list contains a cell.
- static int gds_tree_check_iterate_ref_and_check (struct gds_cell *cell_to_check, GList **visited_cells)

  This function follows down the reference list of a cell and marks each visited subcell and detects loops.
- int gds_tree_check_reference_loops (struct gds_library *lib)

  gds_tree_check_reference_loops checks if the given library contains reference loops

11.13.1 Detailed Description

11.13.2 Macro Definition Documentation

11.13.2.1 CELL_NAME_MAX

```
#define CELL_NAME_MAX (100)
```

Maximum length of a gds_cell::name or a gds_library::name.

Definition at line 37 of file gds-types.h.

11.13.2.2 GDS_DEFAULT_UNITS

```
#define GDS_DEFAULT_UNITS (10E-9)
```

Default units assumed for library.

Note

This value is usually overwritten with the value defined in the library.

Definition at line 51 of file gds-parser.c.

11.13.2.3 GDS_ERROR

```
#define GDS_ERROR(
              fmt,
              ...  ) printf("[PARSE_ERROR] " fmt "\n", ##__VA_ARGS__)
```

Print GDS error.

Definition at line 53 of file gds-parser.c.

11.13.2.4 GDS_INF

```
#define GDS_INF(
              fmt,
              ...  )
```

Definition at line 60 of file gds-parser.c.

11.13.2.5 GDS_PRINT_DEBUG_INFOS

```
#define GDS_PRINT_DEBUG_INFOS (0)
```

1: Print infos, 0: Don't print

Definition at line 38 of file gds-parser.h.

11.13.2.6 GDS_WARN

```
#define GDS_WARN(
              fmt,
              ...  ) printf("[PARSE_WARNING] " fmt "\n", ##__VA_ARGS__)
```

Print GDS warning.

Definition at line 54 of file gds-parser.c.

11.13.2.7 MAX

```
#define MAX(
              a,
              b ) (((a) > (b)) ?  (a) :  (b))
```

Return bigger number.

Definition at line 41 of file gds-types.h.

11.13.2.8 MIN

```
#define MIN(
              a,
              b ) (((a) < (b)) ?  (a) :  (b))
```

Return smaller number.

Definition at line 40 of file gds-types.h.

11.13.3 Enumeration Type Documentation

11.13.3.1 anonymous enum

```
anonymous enum
```

Defintion of check counter default value that indicates that the corresponding check has not yet been executed.

Enumerator

GDS_CELL_CHECK_NOT_RUN	

Definition at line 45 of file gds-types.h.

11.13.3.2 gds_record

`enum gds_record`

Enumerator

INVALID	
HEADER	
BGNLIB	
LIBNAME	
UNITS	
ENDLIB	
BGNSTR	
STRNAME	
ENDSTR	
BOUNDARY	
PATH	
SREF	
ENDEL	
XY	
MAG	
ANGLE	
SNAME	
STRANS	
BOX	
LAYER	
DATATYPE	
WIDTH	
PATHTYPE	
COLROW	
AREF	

Definition at line 62 of file gds-parser.c.

11.13.3.3 graphics_type

`enum graphics_type`

Types of graphic objects.

Enumerator

GRAPHIC_PATH	Path. Esentially a line.
GRAPHIC_POLYGON	An arbitrary polygon.

Enumerator

GRAPHIC_BOX	A rectangle. **Warning** Implementation in renderers might be buggy!

Definition at line 48 of file gds-types.h.

11.13.3.4 path_type

```
enum path_type
```

Defines the line caps of a path.

Enumerator

PATH_FLUSH	
PATH_ROUNDED	
PATH_SQUARED	

Definition at line 58 of file gds-types.h.

11.13.4 Function Documentation

11.13.4.1 append_cell()

```
static GList * append_cell (
            GList * curr_list,
            struct gds_cell ** cell_ptr )  [static]
```

append_cell Append a gds_cell to a list

Usage similar to append_cell_ref().

Parameters

curr_list	List containing gds_cell elements. May be NULL
cell_ptr	newly created cell

Returns

new pointer to list

Definition at line 344 of file gds-parser.c.

Here is the caller graph for this function:

11.13.4.2 append_cell_ref()

```
static GList * append_cell_ref (
            GList * curr_list,
            struct gds_cell_instance ** instance_ptr )  [static]
```

Append a cell reference to the reference GList.

Appends a new gds_cell_instance to `curr_list` and returns the new element via `instance_ptr`

Parameters

curr_list	List of gds_cell_instance elements. May be NULL
instance_ptr	newly created element

Returns

new GList pointer

Definition at line 373 of file gds-parser.c.

Here is the caller graph for this function:

11.13.4.3 append_library()

```
static GList * append_library (
            GList * curr_list,
            struct gds_library ** library_ptr )  [static]
```

Append library to list.

Parameters

curr_list	List containing gds_library elements. May be NULL.
library_ptr	Return of newly created library.

Returns

Newly created list pointer

Definition at line 269 of file gds-parser.c.

Here is the caller graph for this function:

11.13.4.4 append_vertex()

```
static GList * append_vertex (
            GList * curr_list,
            int x,
            int y )  [static]
```

Appends vertext List.

Parameters

curr_list	List containing gds_point elements. May be NULL.
x	x-coordinate of new point
y	y-coordinate of new point

Returns

new Pointer to List.

Definition at line 323 of file gds-parser.c.

Here is the caller graph for this function:

11.13.4.5 clear_lib_list()

```
int clear_lib_list (
            GList ** library_list )
```

Deletes all libraries including cells, references etc.

Parameters

library_list	Pointer to a list of gds_library. Is set to NULL after completion.

Returns

0

Definition at line 1146 of file gds-parser.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.13.4.6 convert_aref_to_sref()

```
static void convert_aref_to_sref (
            struct gds_cell_array_instance * aref,
            struct gds_cell * container_cell )  [static]
```

Convert AREF to a bunch of SREFs and append them to `container_cell`.

This function converts a single array reference (`aref`) to gds_cell_array_instance::rows * gds_cell_array_instance::columns single references (SREFs). See gds_cell_instance.

Both gds_cell_array_instance::rows and gds_cell_array_instance::columns must be larger than zero.

Parameters

in	*aref*	Array reference to parse
in	*container_cell*	cell to add the converted single references to.

Definition at line 575 of file gds-parser.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.13.4.7 delete_cell_element()

```
static void delete_cell_element (
            struct gds_cell * cell )  [static]
```

delete_cell_element

Parameters

cell	

Definition at line 1122 of file gds-parser.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.13.4.8 delete_cell_inst_element()

```
static void delete_cell_inst_element (
            struct gds_cell_instance * cell_inst )  [static]
```

delete_cell_inst_element

Parameters

cell_inst	

Definition at line 1089 of file gds-parser.c.

Here is the caller graph for this function:

11.13.4.9 delete_graphics_obj()

```
static void delete_graphics_obj (
            struct gds_graphics * gfx )  [static]
```

delete_graphics_obj

Parameters

gfx	

Definition at line 1109 of file gds-parser.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.13.4.10 delete_library_element()

```
static void delete_library_element (
            struct gds_library * lib )  [static]
```

delete_library_element

Parameters

lib	

Definition at line 1136 of file gds-parser.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.13.4.11 delete_vertex()

```
static void delete_vertex (
            struct gds_point * vertex )  [static]
```

delete_vertex

Parameters

vertex	

Definition at line 1099 of file gds-parser.c.

Here is the caller graph for this function:

11.13.4.12 gds_convert_double()

```
static double gds_convert_double (
            const char * data )  [static]
```

Convert GDS 8-byte real to double.

Parameters

data	8 Byte GDS real

Returns

result

Definition at line 172 of file gds-parser.c.

Here is the caller graph for this function:

11.13.4.13 gds_convert_signed_int()

```
static signed int gds_convert_signed_int (
            const char * data )  [static]
```

Convert GDS INT32 to int.

Parameters

data	Buffer containing the int

Returns

result

Definition at line 217 of file gds-parser.c.

Here is the caller graph for this function:

11.13.4.14 gds_convert_signed_int16()

```
static int16_t gds_convert_signed_int16 (
            const char * data )  [static]
```

Convert GDS INT16 to int16.

Parameters

data	Buffer containing the INT16

Returns

result

Definition at line 238 of file gds-parser.c.

Here is the caller graph for this function:

11.13.4.15 gds_convert_unsigned_int16()

```
static uint16_t gds_convert_unsigned_int16 (
            const char * data )  [static]
```

Convert GDS UINT16 String to uint16.

Parameters

data	Buffer containing the uint16

Returns

result

Definition at line 253 of file gds-parser.c.

Here is the caller graph for this function:

11.13.4.16 gds_parse_date()

```
static void gds_parse_date (
            const char * buffer,
            int length,
            struct gds_time_field * mod_date,
            struct gds_time_field * access_date )  [static]
```

gds_parse_date

Parameters

buffer	Buffer that contains the GDS Date field
length	Length of `buffer`
mod_date	Modification Date
access_date	Last Access Date

Definition at line 530 of file gds-parser.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.13.4.17 gds_tree_check_cell_references()

```
int gds_tree_check_cell_references (
            struct gds_library * lib )
```

gds_tree_check_cell_references checks if all child cell references can be resolved in the given library

This function will only mark cells that directly contain unresolved references.

If a cell contains a reference to a cell with unresolved references, it is not flagged.

Parameters

lib	The GDS library to check

Returns

less than 0 if an error occured during processing; 0 if all child cells could be resolved; greater than zero if the processing was successful but not all cell references could be resolved. In this case the number of unresolved references is returned

Definition at line 40 of file gds-tree-checker.c.

Here is the caller graph for this function:

11.13.4.18 gds_tree_check_iterate_ref_and_check()

```
static int gds_tree_check_iterate_ref_and_check (
             struct gds_cell * cell_to_check,
             GList ** visited_cells )  [static]
```

This function follows down the reference list of a cell and marks each visited subcell and detects loops.

Parameters

cell_to_check	The cell to check for reference loops
visited_cells	Pointer to list head. May be zero.

Returns

0 if no loops exist; error in processing: <0; loop found: >0

Definition at line 111 of file gds-tree-checker.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.13.4.19 gds_tree_check_list_contains_cell()

```
static int gds_tree_check_list_contains_cell (
             GList * list,
             struct gds_cell * cell )  [static]
```

Check if list contains a cell.

Parameters

list	GList to check. May be a null pointer
cell	Cell to check for

Returns

0 if cell is not in list. 1 if cell is in list

Definition at line 93 of file gds-tree-checker.c.

Here is the caller graph for this function:

11.13.4.20 gds_tree_check_reference_loops()

```
int gds_tree_check_reference_loops (
            struct gds_library * lib )
```

gds_tree_check_reference_loops checks if the given library contains reference loops

Parameters

lib	GDS library

Returns

negative if an error occured, zero if there are no reference loops, else a positive number representing the number of affected cells

Definition at line 158 of file gds-tree-checker.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.13.4.21 name_array_cell_ref()

```
static int name_array_cell_ref (
            struct gds_cell_array_instance * cell_inst,
            unsigned int bytes,
            char * data )  [static]
```

Name cell reference.

Parameters

cell_inst	Cell reference
bytes	Length of name
data	Name

Returns

0 if successful

Definition at line 144 of file gds-parser.c.

Here is the caller graph for this function:

11.13.4.22 name_cell()

```
static int name_cell (
            struct gds_cell * cell,
            unsigned int bytes,
            char * data,
            struct gds_library * lib )  [static]
```

Names a gds_cell.

Parameters

cell	Cell to name
bytes	Length of name
data	Name
lib	Library in which `cell` is located

Returns

0 id successful

Definition at line 432 of file gds-parser.c.

Here is the caller graph for this function:

11.13.4.23 name_cell_ref()

```
static int name_cell_ref (
            struct gds_cell_instance * cell_inst,
            unsigned int bytes,
            char * data )  [static]
```

Name cell reference.

Parameters

cell_inst	Cell reference
bytes	Length of name
data	Name

Returns

0 if successful

Definition at line 114 of file gds-parser.c.

Here is the caller graph for this function:

11.13.4.24 name_library()

```
static int name_library (
            struct gds_library * current_library,
            unsigned int bytes,
            char * data )  [static]
```

Name a gds_library.

Parameters

current_library	Library to name
bytes	Lenght of name
data	Name

Returns

0 if successful

Definition at line 401 of file gds-parser.c.

Here is the caller graph for this function:

11.13.4.25 parse_gds_from_file()

```
int parse_gds_from_file (
            const char * filename,
            GList ** library_array )
```

Parse a GDS file.

This function parses a GDS File and creates a list of libraries, which then contain the different cells.

The function appends The detected libraries to the `library_array` list. The library array may be empty, meaning ∗library_list may be NULL.

Parameters

`in`	*filename*	Path to the GDS file
`in,out`	*library_array*	GList Pointer.

Returns

0 if successful

Definition at line 620 of file gds-parser.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.13.4.26 parse_reference_list()

```
static void parse_reference_list (
            gpointer gcell_ref,
            gpointer glibrary )  [static]
```

Search for cell reference `gcell_ref` in `glibrary`.

Search cell referenced by `gcell_ref` inside `glibrary` and update gds_cell_instance::cell_ref with found gds_cell

Parameters

gcell_ref	gpointer cast of struct gds_cell_instance ∗
glibrary	gpointer cast of struct gds_library ∗

Definition at line 464 of file gds-parser.c.

Here is the caller graph for this function:

11.13.4.27 prepend_graphics()

```
static GList * prepend_graphics (
            GList * curr_list,
            enum graphics_type type,
            struct gds_graphics ** graphics_ptr )  [static]
```

Prepend graphics to list.

Parameters

curr_list	List containing gds_graphics elements. May be NULL
type	Type of graphics
graphics_ptr	newly created graphic is written here

Returns

new list pointer

Definition at line 294 of file gds-parser.c.

Here is the caller graph for this function:

11.13.4.28 scan_cell_reference_dependencies()

```
static void scan_cell_reference_dependencies (
            gpointer gcell,
            gpointer library )  [static]
```

Scans cell references inside cell This function searches all the references in `gcell` and updates the gds_cell_instance::cell_ref field in each instance.

Parameters

gcell	pointer cast of gds_cell *
library	Library where the cell references are searched in

Definition at line 496 of file gds-parser.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.13.4.29 scan_library_references()

```
static void scan_library_references (
            gpointer library_list_item,
            gpointer user )  [static]
```

Scans library's cell references.

This function searches all the references between cells and updates the gds_cell_instance::cell_ref field in each instance

Parameters

library_list_item	List containing gds_library elements
user	not used

Definition at line 514 of file gds-parser.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.14 Version Number

Variables

- const char * _app_version_string

  This string holds the Git Based Version Number of the app.
- const char * _app_git_commit

  This string holds the git commit hash of the current HEAD revision.
- const char * _app_version_string = "! version not set !"

  This string holds the Git Based Version Number of the app.
- const char * _app_git_commit = "! Commit hash not available !"

  This string holds the git commit hash of the current HEAD revision.

11.14.1 Detailed Description

See Git Based Version Number

11.14.2 Variable Documentation

11.14.2.1 _app_git_commit [1/2]

```
const char* _app_git_commit  [extern]
```

This string holds the git commit hash of the current HEAD revision.

Definition at line 40 of file version.c.

11.14.2.2 _app_git_commit [2/2]

```
const char* _app_git_commit = "!  Commit hash not available !"
```

This string holds the git commit hash of the current HEAD revision.

Definition at line 40 of file version.c.

11.14.2.3 _app_version_string [1/2]

```
const char* _app_version_string  [extern]
```

This string holds the Git Based Version Number of the app.

Definition at line 32 of file version.c.

11.14.2.4 _app_version_string [2/2]

```
const char* _app_version_string = "!  version not set !"
```

This string holds the Git Based Version Number of the app.

Definition at line 32 of file version.c.

11.15 RendererSettingsDialog

Collaboration diagram for RendererSettingsDialog:

Data Structures

- struct render_settings

  This struct holds the renderer configuration.
- struct _RendererSettingsDialog

Macros

- #define RENDERER_TYPE_SETTINGS_DIALOG (renderer_settings_dialog_get_type())

Enumerations

- enum output_renderer { RENDERER_LATEX_TIKZ , RENDERER_CAIROGRAPHICS_PDF , RENDERER_CAIROGRAPHICS_ }

  return type of the RedererSettingsDialog
- enum { PROP_CELL_NAME = 1 , PROP_COUNT }

Functions

- RendererSettingsDialog * renderer_settings_dialog_new (GtkWindow *parent)

  Create a new RedererSettingsDialog GObject.
- G_END_DECLS void renderer_settings_dialog_set_settings (RendererSettingsDialog *dialog, struct render_settings *settings)

  Apply settings to dialog.
- void renderer_settings_dialog_get_settings (RendererSettingsDialog *dialog, struct render_settings *settings)

  Get the settings configured in the dialog.
- void renderer_settings_dialog_set_cell_width (RendererSettingsDialog *dialog, unsigned int width)

  renderer_settings_dialog_set_cell_width Set width for rendered cell
- void renderer_settings_dialog_set_cell_height (RendererSettingsDialog *dialog, unsigned int height)

  renderer_settings_dialog_set_cell_height Set height for rendered cell
- void renderer_settings_dialog_set_database_unit_scale (RendererSettingsDialog *dialog, double unit_in_↩ meters)

  renderer_settings_dialog_set_database_unit_scale Set database scale
- static void renderer_settings_dialog_set_property (GObject *object, guint property_id, const GValue *value, GParamSpec *pspec)
- static void renderer_settings_dialog_get_property (GObject *object, guint property_id, GValue *value, GParamSpec *pspec)
- static void renderer_settings_dialog_class_init (RendererSettingsDialogClass *klass)
- static void show_tex_options (RendererSettingsDialog *self)
- static void hide_tex_options (RendererSettingsDialog *self)
- static void latex_render_callback (GtkToggleButton *radio, RendererSettingsDialog *dialog)
- static gboolean shape_drawer_drawing_callback (GtkWidget *widget, cairo_t *cr, gpointer data)
- static double convert_number_to_engineering (double input, const char **out_prefix)
- static void renderer_settings_dialog_update_labels (RendererSettingsDialog *self)
- static void scale_value_changed (GtkRange *range, gpointer user_data)
- static void renderer_settings_dialog_init (RendererSettingsDialog *self)

Variables

- static GParamSpec * properties [PROP_COUNT]

11.15.1 Detailed Description

11.15.2 Macro Definition Documentation

11.15.2.1 RENDERER_TYPE_SETTINGS_DIALOG

```
#define RENDERER_TYPE_SETTINGS_DIALOG (renderer_settings_dialog_get_type())
```

Definition at line 51 of file conv-settings-dialog.h.

11.15.3 Enumeration Type Documentation

11.15.3.1 anonymous enum

```
anonymous enum
```

Enumerator

PROP_CELL_NAME	
PROP_COUNT	

Definition at line 58 of file conv-settings-dialog.c.

11.15.3.2 output_renderer

```
enum output_renderer
```

return type of the RedererSettingsDialog

Enumerator

RENDERER_LATEX_TIKZ	
RENDERER_CAIROGRAPHICS_PDF	
RENDERER_CAIROGRAPHICS_SVG	

Definition at line 40 of file conv-settings-dialog.h.

11.15.4 Function Documentation

11.15.4.1 convert_number_to_engineering()

```
static double convert_number_to_engineering (
            double input,
            const char ** out_prefix )  [static]
```

Definition at line 185 of file conv-settings-dialog.c.

Here is the caller graph for this function:

11.15.4.2 hide_tex_options()

```
static void hide_tex_options (
            RendererSettingsDialog * self )  [static]
```

Definition at line 121 of file conv-settings-dialog.c.

Here is the caller graph for this function:

11.15.4.3 latex_render_callback()

```
static void latex_render_callback (
            GtkToggleButton * radio,
            RendererSettingsDialog * dialog )  [static]
```

Definition at line 127 of file conv-settings-dialog.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.15.4.4 renderer_settings_dialog_class_init()

```
static void renderer_settings_dialog_class_init (
            RendererSettingsDialogClass * klass )  [static]
```

Definition at line 98 of file conv-settings-dialog.c.

Here is the call graph for this function:

11.15.4.5 renderer_settings_dialog_get_property()

```
static void renderer_settings_dialog_get_property (
            GObject * object,
            guint property_id,
            GValue * value,
            GParamSpec * pspec )  [static]
```

Definition at line 82 of file conv-settings-dialog.c.

Here is the caller graph for this function:

11.15.4.6 renderer_settings_dialog_get_settings()

```
void renderer_settings_dialog_get_settings (
            RendererSettingsDialog * dialog,
            struct render_settings * settings )
```

Get the settings configured in the dialog.

Parameters

dialog	
settings	

Definition at line 321 of file conv-settings-dialog.c.

Here is the caller graph for this function:

11.15.4.7 renderer_settings_dialog_init()

```
static void renderer_settings_dialog_init (
            RendererSettingsDialog * self )  [static]
```

Definition at line 269 of file conv-settings-dialog.c.

Here is the call graph for this function:

11.15.4.8 renderer_settings_dialog_new()

```
RendererSettingsDialog * renderer_settings_dialog_new (
            GtkWindow * parent )
```

Create a new RedererSettingsDialog GObject.

Parameters

parent	Parent window

Returns

Created dialog object

Definition at line 310 of file conv-settings-dialog.c.

Here is the caller graph for this function:

11.15.4.9 renderer_settings_dialog_set_cell_height()

```
void renderer_settings_dialog_set_cell_height (
            RendererSettingsDialog * dialog,
            unsigned int height )
```

renderer_settings_dialog_set_cell_height Set height for rendered cell

Parameters

dialog	
height	Height in database units

Definition at line 377 of file conv-settings-dialog.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.15.4.10 renderer_settings_dialog_set_cell_width()

```
void renderer_settings_dialog_set_cell_width (
            RendererSettingsDialog * dialog,
            unsigned int width )
```

renderer_settings_dialog_set_cell_width Set width for rendered cell

Parameters

dialog	
width	Width in database units

Definition at line 365 of file conv-settings-dialog.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.15.4.11 renderer_settings_dialog_set_database_unit_scale()

```
void renderer_settings_dialog_set_database_unit_scale (
            RendererSettingsDialog * dialog,
            double unit_in_meters )
```

renderer_settings_dialog_set_database_unit_scale Set database scale

Parameters

dialog	dialog element
unit_in_meters	Database unit in meters

Definition at line 389 of file conv-settings-dialog.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.15.4.12 renderer_settings_dialog_set_property()

```
static void renderer_settings_dialog_set_property (
            GObject * object,
            guint property_id,
            const GValue * value,
            GParamSpec * pspec )  [static]
```

Definition at line 65 of file conv-settings-dialog.c.

Here is the caller graph for this function:

11.15.4.13 renderer_settings_dialog_set_settings()

```
void renderer_settings_dialog_set_settings (
            RendererSettingsDialog * dialog,
            struct render_settings * settings )
```

Apply settings to dialog.

Parameters

dialog	
settings	

Definition at line 340 of file conv-settings-dialog.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.15.4.14 renderer_settings_dialog_update_labels()

```
static void renderer_settings_dialog_update_labels (
            RendererSettingsDialog * self )  [static]
```

Definition at line 224 of file conv-settings-dialog.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.15.4.15 scale_value_changed()

```
static void scale_value_changed (
            GtkRange * range,
            gpointer user_data )  [static]
```

Definition at line 260 of file conv-settings-dialog.c.

Here is the call graph for this function: Here is the caller graph for this function:

11.15.4.16 shape_drawer_drawing_callback()

```
static gboolean shape_drawer_drawing_callback (
            GtkWidget * widget,
            cairo_t * cr,
            gpointer data )  [static]
```

Definition at line 135 of file conv-settings-dialog.c.

Here is the caller graph for this function:

11.15.4.17 show_tex_options()

```
static void show_tex_options (
            RendererSettingsDialog * self )  [static]
```

Definition at line 114 of file conv-settings-dialog.c.

Here is the caller graph for this function:

11.15.5 Variable Documentation

11.15.5.1 properties

```
GParamSpec* properties[PROP_COUNT]  [static]
```

Definition at line 63 of file conv-settings-dialog.c.

11.16 LayerElement

Collaboration diagram for LayerElement:

Data Structures

- struct _LayerElementPriv
- struct _LayerElement
- struct layer_element_dnd_data

  This structure holds the necessary data to set up a LayerElement for Drag'n'Drop.

Macros

- #define TYPE_LAYER_ELEMENT (layer_element_get_type())

Typedefs

- typedef struct _LayerElementPriv LayerElementPriv

Functions

- GtkWidget * layer_element_new (void)

  Create new layer element object.
- const char * layer_element_get_name (LayerElement *elem)

  get name of the layer
- void layer_element_set_name (LayerElement *elem, const char *name)

  layer_element_set_name
- void layer_element_set_layer (LayerElement *elem, int layer)

  Set layer number for this layer.
- int layer_element_get_layer (LayerElement *elem)

  Get layer number.
- void layer_element_set_export (LayerElement *elem, gboolean export)

  Set export flag for this layer.
- gboolean layer_element_get_export (LayerElement *elem)

  Get export flag of layer.
- void layer_element_get_color (LayerElement *elem, GdkRGBA *rgba)

  Get color of layer.
- void layer_element_set_color (LayerElement *elem, GdkRGBA *rgba)

  Set color of layer.
- void layer_element_set_dnd_callbacks (LayerElement *elem, struct layer_element_dnd_data *data)

  Setup drag and drop of `elem` for use in the LayerSelector.
- static void layer_element_dispose (GObject *obj)
- static void layer_element_constructed (GObject *obj)
- static void layer_element_class_init (LayerElementClass *klass)
- static void layer_element_init (LayerElement *self)

11.16.1 Detailed Description

11.16.2 Macro Definition Documentation

11.16.2.1 TYPE_LAYER_ELEMENT

```
#define TYPE_LAYER_ELEMENT (layer_element_get_type())
```

Definition at line 42 of file layer-element.h.

11.16.3 Typedef Documentation

11.16.3.1 LayerElementPriv

```
typedef struct _LayerElementPriv LayerElementPriv
```

11.16.4 Function Documentation

11.16.4.1 layer_element_class_init()

```
static void layer_element_class_init (
            LayerElementClass * klass )  [static]
```

Definition at line 55 of file layer-element.c.

Here is the call graph for this function:

11.16.4.2 layer_element_constructed()

```
static void layer_element_constructed (
            GObject * obj )  [static]
```

Definition at line 50 of file layer-element.c.

Here is the caller graph for this function:

11.16.4.3 layer_element_dispose()

```
static void layer_element_dispose (
            GObject * obj )  [static]
```

Definition at line 44 of file layer-element.c.

Here is the caller graph for this function:

11.16.4.4 layer_element_get_color()

```
void layer_element_get_color (
            LayerElement * elem,
            GdkRGBA * rgba )
```

Get color of layer.

Parameters

elem	Layer Element
rgba	RGBA color

Definition at line 123 of file layer-element.c.

Here is the caller graph for this function:

11.16.4.5 layer_element_get_export()

```
gboolean layer_element_get_export (
            LayerElement * elem )
```

Get export flag of layer.

Parameters

elem	Layer Element

Returns

Definition at line 118 of file layer-element.c.

Here is the caller graph for this function:

11.16.4.6 layer_element_get_layer()

```
int layer_element_get_layer (
            LayerElement * elem )
```

Get layer number.

Parameters

elem	Layer Element

Returns

Number of this layer

Definition at line 108 of file layer-element.c.

Here is the caller graph for this function:

11.16.4.7 layer_element_get_name()

```
const char * layer_element_get_name (
            LayerElement * elem )
```

get name of the layer

Parameters

elem	Layer element

Returns

Name. Must not be changed, freed or anything else.

Definition at line 87 of file layer-element.c.

Here is the caller graph for this function:

11.16.4.8 layer_element_init()

```
static void layer_element_init (
            LayerElement * self )  [static]
```

Definition at line 63 of file layer-element.c.

11.16.4.9 layer_element_new()

```
GtkWidget * layer_element_new (
            void  )
```

Create new layer element object.

Returns

new object

Definition at line 82 of file layer-element.c.

Here is the caller graph for this function:

11.16.4.10 layer_element_set_color()

```
void layer_element_set_color (
            LayerElement * elem,
            GdkRGBA * rgba )
```

Set color of layer.

Parameters

elem	Layer Element
rgba	RGBA color

Definition at line 131 of file layer-element.c.

Here is the caller graph for this function:

11.16.4.11 layer_element_set_dnd_callbacks()

```
void layer_element_set_dnd_callbacks (
            LayerElement * elem,
            struct layer_element_dnd_data * data )
```

Setup drag and drop of `elem` for use in the LayerSelector.

Parameters

elem	Layer element to set up
data	Data array containing the necessary callbacks etc. for drag and drop.

Definition at line 139 of file layer-element.c.

Here is the caller graph for this function:

11.16.4.12 layer_element_set_export()

```
void layer_element_set_export (
            LayerElement * elem,
            gboolean export )
```

Set export flag for this layer.

Parameters

elem	Layer Element
export	flag

Definition at line 113 of file layer-element.c.

Here is the caller graph for this function:

11.16.4.13 layer_element_set_layer()

```
void layer_element_set_layer (
            LayerElement * elem,
            int layer )
```

Set layer number for this layer.

Parameters

elem	Layer element
layer	Layer number

Definition at line 97 of file layer-element.c.

Here is the caller graph for this function:

11.16.4.14 layer_element_set_name()

```
void layer_element_set_name (
            LayerElement * elem,
            const char * name )
```

layer_element_set_name

Parameters

elem	set the name of the layer
name	Name. Can be freed after call to this function

Definition at line 92 of file layer-element.c.

Here is the caller graph for this function:

11.17 Example Plugin for External Renderer

Collaboration diagram for Example Plugin for External Renderer:

Functions

- int EXPORTED_FUNC_DECL() EXTERNAL_LIBRARY_RENDER_FUNCTION (struct gds_cell *toplevel, GList *layer_info_list, const char *output_file_name, double scale)
- int EXPORTED_FUNC_DECL() EXTERNAL_LIBRARY_INIT_FUNCTION (const char *params, const char *version)

11.17.1 Detailed Description

This is a template / example for an external renderer plugin

11.17.2 Function Documentation

11.17.2.1 EXTERNAL_LIBRARY_INIT_FUNCTION()

```
int EXPORTED_FUNC_DECL() EXTERNAL_LIBRARY_INIT_FUNCTION (
             const char * params,
             const char * version )
```

Definition at line 43 of file plugin-main.c.

Here is the caller graph for this function:

11.17.2.2 EXTERNAL_LIBRARY_RENDER_FUNCTION()

```
int EXPORTED_FUNC_DECL() EXTERNAL_LIBRARY_RENDER_FUNCTION (
             struct gds_cell * toplevel,
             GList * layer_info_list,
             const char * output_file_name,
             double scale )
```

Definition at line 34 of file plugin-main.c.

Here is the caller graph for this function:

Chapter 12

Data Structure Documentation

12.1 _ActivityBar Struct Reference

Opaque ActivityBar object. Not viewable outside this source file.

Data Fields

- GtkBox super
- GtkWidget * spinner
- GtkWidget * label

12.1.1 Detailed Description

Opaque ActivityBar object. Not viewable outside this source file.

Definition at line 43 of file activity-bar.c.

12.1.2 Field Documentation

12.1.2.1 label

```
GtkWidget* label
```

Definition at line 47 of file activity-bar.c.

12.1.2.2 spinner

```
GtkWidget* spinner
```

Definition at line 46 of file activity-bar.c.

12.1.2.3 super

```
GtkBox super
```

Definition at line 44 of file activity-bar.c.

The documentation for this struct was generated from the following file:

- activity-bar.c

12.2 _CairoRenderer Struct Reference

Data Fields

- GdsOutputRenderer parent
- gboolean svg

  TRUE: SVG output, FALSE: PDF output.

12.2.1 Detailed Description

Definition at line 40 of file cairo-renderer.c.

12.2.2 Field Documentation

12.2.2.1 parent

```
GdsOutputRenderer parent
```

Definition at line 41 of file cairo-renderer.c.

12.2.2.2 svg

```
gboolean svg
```

TRUE: SVG output, FALSE: PDF output.

Definition at line 42 of file cairo-renderer.c.

The documentation for this struct was generated from the following file:

- cairo-renderer.c

12.3 gds_cell_checks::_check_internals Struct Reference

For the internal use of the checker.

```
#include <gds-types.h>
```

Data Fields

- int marker

12.3.1 Detailed Description

For the internal use of the checker.

Warning

Do not use this structure and its contents!

Definition at line 78 of file gds-types.h.

12.3.2 Field Documentation

12.3.2.1 marker

```
int marker
```

Definition at line 79 of file gds-types.h.

The documentation for this struct was generated from the following file:

- gds-types.h

12.4 _ColorPalette Struct Reference

Data Fields

- GObject parent
- GdkRGBA * color_array

  The internal array to store the colors.
- unsigned int color_array_length

  The length of the _ColorPalette::color_array array.
- gpointer dummy [4]

12.4.1 Detailed Description

Definition at line 28 of file color-palette.c.

12.4.2 Field Documentation

12.4.2.1 color_array

```
GdkRGBA* color_array
```

The internal array to store the colors.

Definition at line 34 of file color-palette.c.

12.4.2.2 color_array_length

```
unsigned int color_array_length
```

The length of the _ColorPalette::color_array array.

Definition at line 36 of file color-palette.c.

12.4.2.3 dummy

```
gpointer dummy[4]
```

Definition at line 39 of file color-palette.c.

12.4.2.4 parent

```
GObject parent
```

Definition at line 30 of file color-palette.c.

The documentation for this struct was generated from the following file:

- color-palette.c

12.5 _ExternalRenderer Struct Reference

Data Fields

- GdsOutputRenderer parent
- char * shared_object_path
- char * cli_param_string

12.5.1 Detailed Description

Definition at line 41 of file external-renderer.c.

12.5.2 Field Documentation

12.5.2.1 cli_param_string

```
char* cli_param_string
```

Definition at line 44 of file external-renderer.c.

12.5.2.2 parent

```
GdsOutputRenderer parent
```

Definition at line 42 of file external-renderer.c.

12.5.2.3 shared_object_path

```
char* shared_object_path
```

Definition at line 43 of file external-renderer.c.

The documentation for this struct was generated from the following file:

- external-renderer.c

12.6 _GdsOutputRendererClass Struct Reference

Base output renderer class structure.

```
#include <gds-output-renderer.h>
```

Data Fields

- GObjectClass parent_class
- int(* render_output)(GdsOutputRenderer *renderer, struct gds_cell *cell, double scale)

  Virtual render output function. Overwritten by final class implementation.
- gpointer padding [4]

12.6.1 Detailed Description

Base output renderer class structure.

Note

This structure is only used for internal inheritance of GObjects. Do not use in code outside of these classes.

Definition at line 49 of file gds-output-renderer.h.

12.6.2 Field Documentation

12.6.2.1 padding

```
gpointer padding[4]
```

Definition at line 58 of file gds-output-renderer.h.

12.6.2.2 parent_class

```
GObjectClass parent_class
```

Definition at line 50 of file gds-output-renderer.h.

12.6.2.3 render_output

```
int(* render_output) (GdsOutputRenderer *renderer, struct gds_cell *cell, double scale)
```

Virtual render output function. Overwritten by final class implementation.

Definition at line 55 of file gds-output-renderer.h.

The documentation for this struct was generated from the following file:

- gds-output-renderer.h

12.7 _GdsRenderGui Struct Reference

Collaboration diagram for _GdsRenderGui:

Data Fields

- GObject parent
- GtkWindow * main_window
- GtkWidget * convert_button
- GtkWidget * open_button
- GtkWidget * load_layer_button
- GtkWidget * save_layer_button
- GtkWidget * select_all_button
- GtkTreeStore * cell_tree_store
- GtkTreeModelFilter * cell_filter
- GtkWidget * cell_search_entry
- LayerSelector * layer_selector
- GtkTreeView * cell_tree_view
- GList * gds_libraries
- ActivityBar * activity_status_bar
- struct render_settings render_dialog_settings
- ColorPalette * palette
- struct gui_button_states button_state_data

12.7.1 Detailed Description

Definition at line 63 of file gds-render-gui.c.

12.7.2 Field Documentation

12.7.2.1 activity_status_bar

```
ActivityBar* activity_status_bar
```

Definition at line 80 of file gds-render-gui.c.

12.7.2.2 button_state_data

```
struct gui_button_states button_state_data
```

Definition at line 83 of file gds-render-gui.c.

12.7.2.3 cell_filter

```
GtkTreeModelFilter* cell_filter
```

Definition at line 75 of file gds-render-gui.c.

12.7.2.4 cell_search_entry

```
GtkWidget* cell_search_entry
```

Definition at line 76 of file gds-render-gui.c.

12.7.2.5 cell_tree_store

```
GtkTreeStore* cell_tree_store
```

Definition at line 74 of file gds-render-gui.c.

12.7.2.6 cell_tree_view

```
GtkTreeView* cell_tree_view
```

Definition at line 78 of file gds-render-gui.c.

12.7.2.7 convert_button

```
GtkWidget* convert_button
```

Definition at line 69 of file gds-render-gui.c.

12.7.2.8 gds_libraries

```
GList* gds_libraries
```

Definition at line 79 of file gds-render-gui.c.

12.7.2.9 layer_selector

```
LayerSelector* layer_selector
```

Definition at line 77 of file gds-render-gui.c.

12.7.2.10 load_layer_button

```
GtkWidget* load_layer_button
```

Definition at line 71 of file gds-render-gui.c.

12.7.2.11 main_window

```
GtkWindow* main_window
```

Definition at line 68 of file gds-render-gui.c.

12.7.2.12 open_button

```
GtkWidget* open_button
```

Definition at line 70 of file gds-render-gui.c.

12.7.2.13 palette

```
ColorPalette* palette
```

Definition at line 82 of file gds-render-gui.c.

12.7.2.14 parent

```
GObject parent
```

Definition at line 65 of file gds-render-gui.c.

12.7.2.15 render_dialog_settings

```
struct render_settings render_dialog_settings
```

Definition at line 81 of file gds-render-gui.c.

12.7.2.16 save_layer_button

```
GtkWidget* save_layer_button
```

Definition at line 72 of file gds-render-gui.c.

12.7.2.17 select_all_button

```
GtkWidget* select_all_button
```

Definition at line 73 of file gds-render-gui.c.

The documentation for this struct was generated from the following file:

- gds-render-gui.c

12.8 _LatexRenderer Struct Reference

Struct representing the LaTeX-Renderer object.

Data Fields

- GdsOutputRenderer parent
- gboolean tex_standalone
- gboolean pdf_layers

12.8.1 Detailed Description

Struct representing the LaTeX-Renderer object.

This struct holds the LaTeX renderer internal data. It is only used inside the LaTeX / TikZ Renderer class.

Definition at line 42 of file latex-renderer.c.

12.8.2 Field Documentation

12.8.2.1 parent

```
GdsOutputRenderer parent
```

Definition at line 43 of file latex-renderer.c.

12.8.2.2 pdf_layers

```
gboolean pdf_layers
```

Definition at line 45 of file latex-renderer.c.

12.8.2.3 tex_standalone

```
gboolean tex_standalone
```

Definition at line 44 of file latex-renderer.c.

The documentation for this struct was generated from the following file:

- latex-renderer.c

12.9 _LayerElement Struct Reference

`#include <layer-element.h>`

Collaboration diagram for _LayerElement:

Data Fields

- GtkListBoxRow parent
- LayerElementPriv priv

12.9.1 Detailed Description

Definition at line 53 of file layer-element.h.

12.9.2 Field Documentation

12.9.2.1 parent

`GtkListBoxRow parent`

Definition at line 55 of file layer-element.h.

12.9.2.2 priv

`LayerElementPriv priv`

Definition at line 57 of file layer-element.h.

The documentation for this struct was generated from the following file:

- layer-element.h

12.10 _LayerElementPriv Struct Reference

`#include <layer-element.h>`

Data Fields

- GtkEntry * name
- GtkLabel * layer
- int layer_num
- GtkEventBox * event_handle
- GtkColorButton * color
- GtkCheckButton * export

12.10.1 Detailed Description

Definition at line 44 of file layer-element.h.

12.10.2 Field Documentation

12.10.2.1 color

```
GtkColorButton* color
```

Definition at line 49 of file layer-element.h.

12.10.2.2 event_handle

```
GtkEventBox* event_handle
```

Definition at line 48 of file layer-element.h.

12.10.2.3 export

```
GtkCheckButton* export
```

Definition at line 50 of file layer-element.h.

12.10.2.4 layer

```
GtkLabel* layer
```

Definition at line 46 of file layer-element.h.

12.10.2.5 layer_num

```
int layer_num
```

Definition at line 47 of file layer-element.h.

12.10.2.6 name

```
GtkEntry* name
```

Definition at line 45 of file layer-element.h.

The documentation for this struct was generated from the following file:

- layer-element.h

12.11 _LayerSelector Struct Reference

Data Fields

- GObject parent
- GtkWidget * associated_load_button
- GtkWidget * associated_save_button
- GtkWindow * load_parent_window
- GtkWindow * save_parent_window
- GtkListBox * list_box
- GtkTargetEntry dnd_target
- gpointer dummy [4]

12.11.1 Detailed Description

Definition at line 40 of file layer-selector.c.

12.11.2 Field Documentation

12.11.2.1 associated_load_button

```
GtkWidget* associated_load_button
```

Definition at line 44 of file layer-selector.c.

12.11.2.2 associated_save_button

```
GtkWidget* associated_save_button
```

Definition at line 45 of file layer-selector.c.

12.11.2.3 dnd_target

```
GtkTargetEntry dnd_target
```

Definition at line 50 of file layer-selector.c.

12.11.2.4 dummy

```
gpointer dummy[4]
```

Definition at line 52 of file layer-selector.c.

12.11.2.5 list_box

```
GtkListBox* list_box
```

Definition at line 48 of file layer-selector.c.

12.11.2.6 load_parent_window

```
GtkWindow* load_parent_window
```

Definition at line 46 of file layer-selector.c.

12.11.2.7 parent

```
GObject parent
```

Definition at line 42 of file layer-selector.c.

12.11.2.8 save_parent_window

```
GtkWindow* save_parent_window
```

Definition at line 47 of file layer-selector.c.

The documentation for this struct was generated from the following file:

- layer-selector.c

12.12 _LayerSettings Struct Reference

Data Fields

- GObject parent
- GList * layer_infos
- gpointer padding [12]

12.12.1 Detailed Description

Definition at line 29 of file layer-settings.c.

12.12.2 Field Documentation

12.12.2.1 layer_infos

```
GList* layer_infos
```

Definition at line 31 of file layer-settings.c.

12.12.2.2 padding

```
gpointer padding[12]
```

Definition at line 32 of file layer-settings.c.

12.12.2.3 parent

```
GObject parent
```

Definition at line 30 of file layer-settings.c.

The documentation for this struct was generated from the following file:

- layer-settings.c

12.13 _LibCellRenderer Struct Reference

```
#include <lib-cell-renderer.h>
```

Data Fields

- GtkCellRendererText super

12.13.1 Detailed Description

Definition at line 48 of file lib-cell-renderer.h.

12.13.2 Field Documentation

12.13.2.1 super

```
GtkCellRendererText super
```

Definition at line 50 of file lib-cell-renderer.h.

The documentation for this struct was generated from the following file:

- lib-cell-renderer.h

12.14 _RendererSettingsDialog Struct Reference

Data Fields

- GtkDialog parent
- GtkWidget * radio_latex
- GtkWidget * radio_cairo_pdf
- GtkWidget * radio_cairo_svg
- GtkWidget * scale
- GtkWidget * layer_check
- GtkWidget * standalone_check
- GtkDrawingArea * shape_drawing
- GtkLabel * x_label
- GtkLabel * y_label
- GtkLabel * x_output_label
- GtkLabel * y_output_label
- unsigned int cell_height
- unsigned int cell_width
- double unit_in_meters

12.14.1 Detailed Description

Definition at line 35 of file conv-settings-dialog.c.

12.14.2 Field Documentation

12.14.2.1 cell_height

```
unsigned int cell_height
```

Definition at line 51 of file conv-settings-dialog.c.

12.14.2.2 cell_width

```
unsigned int cell_width
```

Definition at line 52 of file conv-settings-dialog.c.

12.14.2.3 layer_check

```
GtkWidget* layer_check
```

Definition at line 42 of file conv-settings-dialog.c.

12.14.2.4 parent

```
GtkDialog parent
```

Definition at line 36 of file conv-settings-dialog.c.

12.14.2.5 radio_cairo_pdf

```
GtkWidget* radio_cairo_pdf
```

Definition at line 39 of file conv-settings-dialog.c.

12.14.2.6 radio_cairo_svg

```
GtkWidget* radio_cairo_svg
```

Definition at line 40 of file conv-settings-dialog.c.

12.14.2.7 radio_latex

```
GtkWidget* radio_latex
```

Definition at line 38 of file conv-settings-dialog.c.

12.14.2.8 scale

```
GtkWidget* scale
```

Definition at line 41 of file conv-settings-dialog.c.

12.14.2.9 shape_drawing

```
GtkDrawingArea* shape_drawing
```

Definition at line 44 of file conv-settings-dialog.c.

12.14.2.10 standalone_check

```
GtkWidget* standalone_check
```

Definition at line 43 of file conv-settings-dialog.c.

12.14.2.11 unit_in_meters

```
double unit_in_meters
```

Definition at line 53 of file conv-settings-dialog.c.

12.14.2.12 x_label

```
GtkLabel* x_label
```

Definition at line 45 of file conv-settings-dialog.c.

12.14.2.13 x_output_label

```
GtkLabel* x_output_label
```

Definition at line 48 of file conv-settings-dialog.c.

12.14.2.14 y_label

```
GtkLabel* y_label
```

Definition at line 46 of file conv-settings-dialog.c.

12.14.2.15 y_output_label

```
GtkLabel* y_output_label
```

Definition at line 49 of file conv-settings-dialog.c.

The documentation for this struct was generated from the following file:

- conv-settings-dialog.c

12.15 bounding_box::_vectors Struct Reference

Location vectors of upper right and lower left bounding box points.

```
#include <bounding-box.h>
```

Collaboration diagram for bounding_box::_vectors:

Data Fields

- struct vector_2d lower_left
  Lower left point of the bounding box.
- struct vector_2d upper_right
  Upper right point of the bounding box.

12.15.1 Detailed Description

Location vectors of upper right and lower left bounding box points.

Note

Coordinate System is (y up | x right)

Definition at line 60 of file bounding-box.h.

12.15.2 Field Documentation

12.15.2.1 lower_left

```
struct vector_2d lower_left
```

Lower left point of the bounding box.

Definition at line 62 of file bounding-box.h.

12.15.2.2 upper_right

```
struct vector_2d upper_right
```

Upper right point of the bounding box.

Definition at line 64 of file bounding-box.h.

The documentation for this struct was generated from the following file:

- bounding-box.h

12.16 application_data Struct Reference

Structure containing The GtkApplication and a list containing the GdsRenderGui objects.

Data Fields

- GtkApplication * app
- GList * gui_list

12.16.1 Detailed Description

Structure containing The GtkApplication and a list containing the GdsRenderGui objects.

Definition at line 40 of file main.c.

12.16.2 Field Documentation

12.16.2.1 app

```
GtkApplication* app
```

Definition at line 41 of file main.c.

12.16.2.2 gui_list

```
GList* gui_list
```

Definition at line 42 of file main.c.

The documentation for this struct was generated from the following file:

- main.c

12.17 bounding_box Union Reference

Union describing a bounding box.

`#include <bounding-box.h>`

Collaboration diagram for bounding_box:

Data Structures

- struct _vectors

  Location vectors of upper right and lower left bounding box points.

Data Fields

- struct bounding_box::_vectors vectors
- struct vector_2d vector_array [2]

  Array of vectors representing a bounding box.

12.17.1 Detailed Description

Union describing a bounding box.

Two ways of accessing a bounding box are possible.

Either, use the "named" vectors struct to specifically access the points

```
lower_left = box.vectors.lower_left;
upper right = box.vectors.upper_right;
```

or use the iterable vector array:

```
for (i = 0; i < 2; i++)
        box.vector_array[i] = points[i];
```

Definition at line 55 of file bounding-box.h.

12.17.2 Field Documentation

12.17.2.1 vector_array

```
struct vector_2d vector_array[2]
```

Array of vectors representing a bounding box.

Note

This is more convenient for iterating

Definition at line 70 of file bounding-box.h.

12.17.2.2 vectors

```
struct bounding_box::_vectors vectors
```

The documentation for this union was generated from the following file:

- bounding-box.h

12.18 cairo_layer Struct Reference

The cairo_layer struct Each rendered layer is represented by this struct.

Collaboration diagram for cairo_layer:

Data Fields

- cairo_t * cr
  cairo context for layer
- cairo_surface_t * rec
  Recording surface to hold the layer.
- struct layer_info * linfo
  Reference to layer information.

12.18.1 Detailed Description

The cairo_layer struct Each rendered layer is represented by this struct.

Definition at line 51 of file cairo-renderer.c.

12.18.2 Field Documentation

12.18.2.1 cr

```
cairo_t* cr
```

cairo context for layer

Definition at line 52 of file cairo-renderer.c.

12.18.2.2 linfo

```
struct layer_info* linfo
```

Reference to layer information.

Definition at line 54 of file cairo-renderer.c.

12.18.2.3 rec

```
cairo_surface_t* rec
```

Recording surface to hold the layer.

Definition at line 53 of file cairo-renderer.c.

The documentation for this struct was generated from the following file:

- cairo-renderer.c

12.19 external_renderer_params Struct Reference

External renderer paramameters to command line renderer.

```
#include <command-line.h>
```

Data Fields

- char * so_path
  Path to shared object.
- char * cli_params
  Command line parameters given.

12.19.1 Detailed Description

External renderer paramameters to command line renderer.

Definition at line 39 of file command-line.h.

12.19.2 Field Documentation

12.19.2.1 cli_params

```
char* cli_params
```

Command line parameters given.

Definition at line 48 of file command-line.h.

12.19.2.2 so_path

```
char* so_path
```

Path to shared object.

Definition at line 43 of file command-line.h.

The documentation for this struct was generated from the following file:

- command-line.h

12.20 gds_cell Struct Reference

A Cell inside a gds_library.

```
#include <gds-types.h>
```

Collaboration diagram for gds_cell:

Data Fields

- char name [CELL_NAME_MAX]
- struct gds_time_field mod_time
- struct gds_time_field access_time
- GList * child_cells

  List of gds_cell_instance elements.
- GList * graphic_objs

  List of gds_graphics.
- struct gds_library * parent_library

  Pointer to parent library.
- struct gds_cell_checks checks

  Checking results.

12.20.1 Detailed Description

A Cell inside a gds_library.

Definition at line 122 of file gds-types.h.

12.20.2 Field Documentation

12.20.2.1 access_time

```
struct gds_time_field access_time
```

Definition at line 125 of file gds-types.h.

12.20.2.2 checks

```
struct gds_cell_checks checks
```

Checking results.

Definition at line 129 of file gds-types.h.

12.20.2.3 child_cells

```
GList* child_cells
```

List of gds_cell_instance elements.

Definition at line 126 of file gds-types.h.

12.20.2.4 graphic_objs

```
GList* graphic_objs
```

List of gds_graphics.

Definition at line 127 of file gds-types.h.

12.20.2.5 mod_time

```
struct gds_time_field mod_time
```

Definition at line 124 of file gds-types.h.

12.20.2.6 name

```
char name[CELL_NAME_MAX]
```

Definition at line 123 of file gds-types.h.

12.20.2.7 parent_library

```
struct gds_library* parent_library
```

Pointer to parent library.

Definition at line 128 of file gds-types.h.

The documentation for this struct was generated from the following file:

- gds-types.h

12.21 gds_cell_array_instance Struct Reference

Struct representing an array instantiation.

Collaboration diagram for gds_cell_array_instance:

Data Fields

- char ref_name [CELL_NAME_MAX]
  Name of referenced cell.
- struct gds_cell * cell_ref
  Referenced gds_cell structure.
- struct gds_point control_points [3]
  The three control points.
- int flipped
  Mirror each instance on x-axis before rotation.
- double angle
  Angle of rotation for each instance (counter clockwise) in degrees.
- double magnification
  Magnification of each instance.
- int columns
  Column count.
- int rows
  Row count.

12.21.1 Detailed Description

Struct representing an array instantiation.

This struct is defined locally because it is not exposed to the outside of the parser. Array references are internally converted to a bunch of standard gds_cell_instance elements.

Definition at line 96 of file gds-parser.c.

12.21.2 Field Documentation

12.21.2.1 angle

```
double angle
```

Angle of rotation for each instance (counter clockwise) in degrees.

Definition at line 101 of file gds-parser.c.

12.21.2.2 cell_ref

```
struct gds_cell* cell_ref
```

Referenced gds_cell structure.

Definition at line 98 of file gds-parser.c.

12.21.2.3 columns

```
int columns
```

Column count.

Definition at line 103 of file gds-parser.c.

12.21.2.4 control_points

```
struct gds_point control_points[3]
```

The three control points.

Definition at line 99 of file gds-parser.c.

12.21.2.5 flipped

```
int flipped
```

Mirror each instance on x-axis before rotation.

Definition at line 100 of file gds-parser.c.

12.21.2.6 magnification

```
double magnification
```

Magnification of each instance.

Definition at line 102 of file gds-parser.c.

12.21.2.7 ref_name

```
char ref_name[CELL_NAME_MAX]
```

Name of referenced cell.

Definition at line 97 of file gds-parser.c.

12.21.2.8 rows

```
int rows
```

Row count.

Definition at line 104 of file gds-parser.c.

The documentation for this struct was generated from the following file:

- gds-parser.c

12.22 gds_cell_checks Struct Reference

Stores the result of the cell checks.

```
#include <gds-types.h>
```

Collaboration diagram for gds_cell_checks:

Data Structures

- struct _check_internals

  For the internal use of the checker.

Data Fields

- int unresolved_child_count

  Number of unresolved cell instances inside this cell. Default: GDS_CELL_CHECK_NOT_RUN.
- int affected_by_reference_loop

  1 if the cell is affected by a reference loop and therefore not renderable. Default: GDS_CELL_CHECK_NOT_RUN
- struct gds_cell_checks::_check_internals _internal

12.22.1 Detailed Description

Stores the result of the cell checks.

Definition at line 71 of file gds-types.h.

12.22.2 Field Documentation

12.22.2.1 _internal

```
struct gds_cell_checks::_check_internals _internal
```

12.22.2.2 affected_by_reference_loop

```
int affected_by_reference_loop
```

1 if the cell is affected by a reference loop and therefore not renderable. Default: GDS_CELL_CHECK_NOT_RUN

Definition at line 73 of file gds-types.h.

12.22.2.3 unresolved_child_count

```
int unresolved_child_count
```

Number of unresolved cell instances inside this cell. Default: GDS_CELL_CHECK_NOT_RUN.

Definition at line 72 of file gds-types.h.

The documentation for this struct was generated from the following file:

- gds-types.h

12.23 gds_cell_instance Struct Reference

This represents an instanc of a cell inside another cell.

`#include <gds-types.h>`

Collaboration diagram for gds_cell_instance:

Data Fields

- char ref_name [CELL_NAME_MAX]
  Name of referenced cell.
- struct gds_cell * cell_ref
  Referenced gds_cell structure.
- struct gds_point origin
  Origin.
- int flipped
  Mirrored on x-axis before rotation.
- double angle
  Angle of rotation (counter clockwise) in degrees.
- double magnification
  magnification

12.23.1 Detailed Description

This represents an instanc of a cell inside another cell.

Definition at line 110 of file gds-types.h.

12.23.2 Field Documentation

12.23.2.1 angle

`double angle`

Angle of rotation (counter clockwise) in degrees.

Definition at line 115 of file gds-types.h.

12.23.2.2 cell_ref

```
struct gds_cell* cell_ref
```

Referenced gds_cell structure.

Definition at line 112 of file gds-types.h.

12.23.2.3 flipped

```
int flipped
```

Mirrored on x-axis before rotation.

Definition at line 114 of file gds-types.h.

12.23.2.4 magnification

```
double magnification
```

magnification

Definition at line 116 of file gds-types.h.

12.23.2.5 origin

```
struct gds_point origin
```

Origin.

Definition at line 113 of file gds-types.h.

12.23.2.6 ref_name

```
char ref_name[CELL_NAME_MAX]
```

Name of referenced cell.

Definition at line 111 of file gds-types.h.

The documentation for this struct was generated from the following file:

- gds-types.h

12.24 gds_graphics Struct Reference

A GDS graphics object.

`#include <gds-types.h>`

Data Fields

- enum graphics_type gfx_type
  Type of graphic.
- GList * vertices
  List of gds_point.
- enum path_type path_render_type
  Line cap.
- int width_absolute
  Width. Not used for objects other than paths.
- int16_t layer
  Layer the graphic object is on.
- int16_t datatype
  Data type of graphic object.

12.24.1 Detailed Description

A GDS graphics object.

Definition at line 98 of file gds-types.h.

12.24.2 Field Documentation

12.24.2.1 datatype

`int16_t datatype`

Data type of graphic object.

Definition at line 104 of file gds-types.h.

12.24.2.2 gfx_type

`enum graphics_type gfx_type`

Type of graphic.

Definition at line 99 of file gds-types.h.

12.24.2.3 layer

```
int16_t layer
```

Layer the graphic object is on.

Definition at line 103 of file gds-types.h.

12.24.2.4 path_render_type

```
enum path_type path_render_type
```

Line cap.

Definition at line 101 of file gds-types.h.

12.24.2.5 vertices

```
GList* vertices
```

List of gds_point.

Definition at line 100 of file gds-types.h.

12.24.2.6 width_absolute

```
int width_absolute
```

Width. Not used for objects other than paths.

Definition at line 102 of file gds-types.h.

The documentation for this struct was generated from the following file:

- gds-types.h

12.25 gds_library Struct Reference

GDS Toplevel library.

```
#include <gds-types.h>
```

Collaboration diagram for gds_library:

Data Fields

- char name [CELL_NAME_MAX]
- struct gds_time_field mod_time
- struct gds_time_field access_time
- double unit_in_meters
- GList ∗ cells
- GList ∗ cell_names

12.25.1 Detailed Description

GDS Toplevel library.

Definition at line 135 of file gds-types.h.

12.25.2 Field Documentation

12.25.2.1 access_time

```
struct gds_time_field access_time
```

Definition at line 138 of file gds-types.h.

12.25.2.2 cell_names

```
GList* cell_names
```

< List of strings that contains all cell names

Definition at line 141 of file gds-types.h.

12.25.2.3 cells

```
GList* cells
```

List of gds_cell that contains all cells in this library

Definition at line 140 of file gds-types.h.

12.25.2.4 mod_time

```
struct gds_time_field mod_time
```

Definition at line 137 of file gds-types.h.

12.25.2.5 name

```
char name[CELL_NAME_MAX]
```

Definition at line 136 of file gds-types.h.

12.25.2.6 unit_in_meters

```
double unit_in_meters
```

Length of a database unit in meters

Definition at line 139 of file gds-types.h.

The documentation for this struct was generated from the following file:

- gds-types.h

12.26 gds_point Struct Reference

A point in the 2D plane. Sometimes referred to as vertex.

```
#include <gds-types.h>
```

Data Fields

- int x
- int y

12.26.1 Detailed Description

A point in the 2D plane. Sometimes referred to as vertex.

Definition at line 63 of file gds-types.h.

12.26.2 Field Documentation

12.26.2.1 x

```
int x
```

Definition at line 64 of file gds-types.h.

12.26.2.2 y

```
int y
```

Definition at line 65 of file gds-types.h.

The documentation for this struct was generated from the following file:

- gds-types.h

12.27 gds_time_field Struct Reference

Date information for cells and libraries.

```
#include <gds-types.h>
```

Data Fields

- uint16_t year
- uint16_t month
- uint16_t day
- uint16_t hour
- uint16_t minute
- uint16_t second

12.27.1 Detailed Description

Date information for cells and libraries.

Definition at line 86 of file gds-types.h.

12.27.2 Field Documentation

12.27.2.1 day

```
uint16_t day
```

Definition at line 89 of file gds-types.h.

12.27.2.2 hour

```
uint16_t hour
```

Definition at line 90 of file gds-types.h.

12.27.2.3 minute

```
uint16_t minute
```

Definition at line 91 of file gds-types.h.

12.27.2.4 month

```
uint16_t month
```

Definition at line 88 of file gds-types.h.

12.27.2.5 second

```
uint16_t second
```

Definition at line 92 of file gds-types.h.

12.27.2.6 year

```
uint16_t year
```

Definition at line 87 of file gds-types.h.

The documentation for this struct was generated from the following file:

- gds-types.h

12.28 GdsOutputRendererPrivate Struct Reference

Collaboration diagram for GdsOutputRendererPrivate:

Data Fields

- gchar * output_file
- LayerSettings * layer_settings
- GMutex settings_lock
- gboolean mutex_init_status
- GTask * task
- GMainContext * main_context
- struct renderer_params async_params
- struct idle_function_params idle_function_parameters
- gpointer padding [11]

12.28.1 Detailed Description

Definition at line 43 of file gds-output-renderer.c.

12.28.2 Field Documentation

12.28.2.1 async_params

```
struct renderer_params async_params
```

Definition at line 50 of file gds-output-renderer.c.

12.28.2.2 idle_function_parameters

```
struct idle_function_params idle_function_parameters
```

Definition at line 51 of file gds-output-renderer.c.

12.28.2.3 layer_settings

```
LayerSettings* layer_settings
```

Definition at line 45 of file gds-output-renderer.c.

12.28.2.4 main_context

```
GMainContext* main_context
```

Definition at line 49 of file gds-output-renderer.c.

12.28.2.5 mutex_init_status

```
gboolean mutex_init_status
```

Definition at line 47 of file gds-output-renderer.c.

12.28.2.6 output_file

```
gchar* output_file
```

Definition at line 44 of file gds-output-renderer.c.

12.28.2.7 padding

```
gpointer padding[11]
```

Definition at line 52 of file gds-output-renderer.c.

12.28.2.8 settings_lock

```
GMutex settings_lock
```

Definition at line 46 of file gds-output-renderer.c.

12.28.2.9 task

```
GTask* task
```

Definition at line 48 of file gds-output-renderer.c.

The documentation for this struct was generated from the following file:

- gds-output-renderer.c

12.29 gui_button_states Struct Reference

Data Fields

- gboolean rendering_active
- gboolean valid_cell_selected

12.29.1 Detailed Description

Definition at line 58 of file gds-render-gui.c.

12.29.2 Field Documentation

12.29.2.1 rendering_active

```
gboolean rendering_active
```

Definition at line 59 of file gds-render-gui.c.

12.29.2.2 valid_cell_selected

```
gboolean valid_cell_selected
```

Definition at line 60 of file gds-render-gui.c.

The documentation for this struct was generated from the following file:

- gds-render-gui.c

12.30 idle_function_params Struct Reference

Data Fields

- GMutex message_lock
- char * status_message

12.30.1 Detailed Description

Definition at line 38 of file gds-output-renderer.c.

12.30.2 Field Documentation

12.30.2.1 message_lock

```
GMutex message_lock
```

Definition at line 39 of file gds-output-renderer.c.

12.30.2.2 status_message

```
char* status_message
```

Definition at line 40 of file gds-output-renderer.c.

The documentation for this struct was generated from the following file:

- gds-output-renderer.c

12.31 layer_element_dnd_data Struct Reference

This structure holds the necessary data to set up a LayerElement for Drag'n'Drop.

```
#include <layer-element.h>
```

Data Fields

- GtkTargetEntry * entries
  Array of target entries for the DnD operation.
- int entry_count
  Count of elements in layer_element_dnd_data::entries array.
- void(* drag_begin)(GtkWidget *, GdkDragContext *, gpointer)
  Callback function for drag_begin event.
- void(* drag_data_get)(GtkWidget *, GdkDragContext *, GtkSelectionData *, guint, guint, gpointer)
  Callback fucktion for data_get event.
- void(* drag_end)(GtkWidget *, GdkDragContext *, gpointer)
  Callback function for drag_end event.

12.31.1 Detailed Description

This structure holds the necessary data to set up a LayerElement for Drag'n'Drop.

Definition at line 63 of file layer-element.h.

12.31.2 Field Documentation

12.31.2.1 drag_begin

```
void(* drag_begin) (GtkWidget *, GdkDragContext *, gpointer)
```

Callback function for drag_begin event.

Definition at line 69 of file layer-element.h.

12.31.2.2 drag_data_get

```
void(* drag_data_get) (GtkWidget *, GdkDragContext *, GtkSelectionData *, guint, guint, gpointer)
```

Callback fucktion for data_get event.

Definition at line 71 of file layer-element.h.

12.31.2.3 drag_end

```
void(* drag_end) (GtkWidget *, GdkDragContext *, gpointer)
```

Callback function for drag_end event.

Definition at line 73 of file layer-element.h.

12.31.2.4 entries

```
GtkTargetEntry* entries
```

Array of target entries for the DnD operation.

Definition at line 65 of file layer-element.h.

12.31.2.5 entry_count

```
int entry_count
```

Count of elements in layer_element_dnd_data::entries array.

Definition at line 67 of file layer-element.h.

The documentation for this struct was generated from the following file:

- layer-element.h

12.32 layer_info Struct Reference

Layer information.

```
#include <layer-settings.h>
```

Data Fields

- int layer
  Layer number.
- char * name
  Layer name.
- int stacked_position
  Position of layer in output.
- GdkRGBA color
  RGBA color used to render this layer.
- int render
  true: Render to output

12.32.1 Detailed Description

Layer information.

This structs contains information on how to render a layer

Note

You probably don't want to use this struct standalone but in combination with a LayerSettings object.

Definition at line 40 of file layer-settings.h.

12.32.2 Field Documentation

12.32.2.1 color

```
GdkRGBA color
```

RGBA color used to render this layer.

Definition at line 45 of file layer-settings.h.

12.32.2.2 layer

```
int layer
```

Layer number.

Definition at line 42 of file layer-settings.h.

12.32.2.3 name

```
char* name
```

Layer name.

Definition at line 43 of file layer-settings.h.

12.32.2.4 render

```
int render
```

true: Render to output

Definition at line 46 of file layer-settings.h.

12.32.2.5 stacked_position

```
int stacked_position
```

Position of layer in output.

Warning

This parameter is not used by any renderer so far

Note

Lower is bottom, higher is top

Definition at line 44 of file layer-settings.h.

The documentation for this struct was generated from the following file:

- layer-settings.h

12.33 render_settings Struct Reference

This struct holds the renderer configuration.

`#include <conv-settings-dialog.h>`

Data Fields

- double scale

  Scale image down by this factor.
- enum output_renderer renderer
- gboolean tex_pdf_layers
- gboolean tex_standalone

12.33.1 Detailed Description

This struct holds the renderer configuration.

Definition at line 56 of file conv-settings-dialog.h.

12.33.2 Field Documentation

12.33.2.1 renderer

`enum output_renderer renderer`

The renderer to use

Definition at line 58 of file conv-settings-dialog.h.

12.33.2.2 scale

`double scale`

Scale image down by this factor.

Note

Used to keep image in bound of maximum coordinate limit

Definition at line 57 of file conv-settings-dialog.h.

12.33.2.3 tex_pdf_layers

```
gboolean tex_pdf_layers
```

Create OCG layers when rendering with TikZ

Definition at line 59 of file conv-settings-dialog.h.

12.33.2.4 tex_standalone

```
gboolean tex_standalone
```

Create a standalone compile TeX file

Definition at line 60 of file conv-settings-dialog.h.

The documentation for this struct was generated from the following file:

- conv-settings-dialog.h

12.34 renderer_params Struct Reference

Collaboration diagram for renderer_params:

Data Fields

- struct gds_cell * cell
- double scale

12.34.1 Detailed Description

Definition at line 33 of file gds-output-renderer.c.

12.34.2 Field Documentation

12.34.2.1 cell

```
struct gds_cell* cell
```

Definition at line 34 of file gds-output-renderer.c.

12.34.2.2 scale

```
double scale
```

Definition at line 35 of file gds-output-renderer.c.

The documentation for this struct was generated from the following file:

- gds-output-renderer.c

12.35 vector_2d Struct Reference

```
#include <vector-operations.h>
```

Data Fields

- double x
- double y

12.35.1 Detailed Description

Definition at line 37 of file vector-operations.h.

12.35.2 Field Documentation

12.35.2.1 x

```
double x
```

Definition at line 38 of file vector-operations.h.

12.35.2.2 y

```
double y
```

Definition at line 39 of file vector-operations.h.

The documentation for this struct was generated from the following file:

- vector-operations.h

Chapter 13

File Documentation

13.1 lib-cell-renderer.c File Reference

LibCellRenderer GObject Class.

```
#include <gds-render/cell-selector/lib-cell-renderer.h>
#include <gds-render/gds-utils/gds-types.h>
```

Include dependency graph for lib-cell-renderer.c:

Enumerations

- enum { PROP_LIB = 1 , PROP_CELL , PROP_ERROR_LEVEL , PROP_COUNT }

Functions

- void lib_cell_renderer_init (LibCellRenderer *self)
- static void lib_cell_renderer_constructed (GObject *obj)
- static void convert_error_level_to_color (GdkRGBA *color, unsigned int error_level)
- static void lib_cell_renderer_set_property (GObject *object, guint param_id, const GValue *value, GParam↩ Spec *pspec)
- static void lib_cell_renderer_get_property (GObject *object, guint param_id, GValue *value, GParamSpec *pspec)
- void lib_cell_renderer_class_init (LibCellRendererClass *klass)
- GtkCellRenderer * lib_cell_renderer_new (void)

  Create a new renderer for renderering gds_cell and gds_library elements.

Variables

- static GParamSpec * properties [PROP_COUNT]

13.1.1 Detailed Description

LibCellRenderer GObject Class.

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file lib-cell-renderer.c.

13.2 lib-cell-renderer.c

Go to the documentation of this file.

```
00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018  Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter.  If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00031 #include <gds-render/cell-selector/lib-cell-renderer.h>
00032 #include <gds-render/gds-utils/gds-types.h>
00033
00034 G_DEFINE_TYPE(LibCellRenderer, lib_cell_renderer, GTK_TYPE_CELL_RENDERER_TEXT)
00035
00036 enum {
00037         PROP_LIB = 1,
00038         PROP_CELL,
00039         PROP_ERROR_LEVEL,
00040         PROP_COUNT
00041 };
00042
00043 void lib_cell_renderer_init(LibCellRenderer *self)
00044 {
00045         (void)self;
00046         /* Nothing to do */
00047 }
00048
00049 static void lib_cell_renderer_constructed(GObject *obj)
00050 {
00051         G_OBJECT_CLASS(lib_cell_renderer_parent_class)->constructed(obj);
00052 }
00053
00054 static void convert_error_level_to_color(GdkRGBA *color, unsigned int error_level)
00055 {
00056
00057         /* Always use no transparency */
00058         color->alpha = 1.0;
00059
00060         if (error_level & LIB_CELL_RENDERER_ERROR_ERR) {
00061                 /* Error set. Color cell red */
00062                 color->red = 1.0;
00063                 color->blue = 0.0;
00064                 color->green = 0.0;
00065         } else if (error_level & LIB_CELL_RENDERER_ERROR_WARN) {
00066                 /* Only warning set; orange color */
00067                 color->red = 1.0;
00068                 color->blue = 0.0;
00069                 color->green = 0.6;
00070         } else {
00071                 /* Everything okay; green color */
00072                 color->red = (double)61.0/(double)255.0;
00073                 color->green = (double)152.0/(double)255.0;
```

```
00074                 color->blue = 0.0;
00075         }
00076 }
00077
00078 static void lib_cell_renderer_set_property(GObject      *object,
00079                                            guint        param_id,
00080                                            const GValue *value,
00081                                            GParamSpec   *pspec)
00082 {
00083         GValue val = G_VALUE_INIT;
00084         GdkRGBA color;
00085
00086         switch (param_id) {
00087         case PROP_LIB:
00088                 g_value_init(&val, G_TYPE_STRING);
00089                 g_value_set_string(&val, ((struct gds_library *)g_value_get_pointer(value))->name);
00090                 g_object_set_property(object, "text", &val);
00091                 g_value_unset(&val);
00092                 break;
00093         case PROP_CELL:
00094                 g_value_init(&val, G_TYPE_STRING);
00095                 g_value_set_string(&val, ((struct gds_cell *)g_value_get_pointer(value))->name);
00096                 g_object_set_property(object, "text", &val);
00097                 g_value_unset(&val);
00098                 break;
00099         case PROP_ERROR_LEVEL:
00100                 /* Set cell color according to error level */
00101                 g_value_init(&val, GDK_TYPE_RGBA);
00102                 convert_error_level_to_color(&color, g_value_get_uint(value));
00103                 g_value_set_boxed(&val, &color);
00104                 g_object_set_property(object, "foreground-rgba", &val);
00105                 g_value_unset(&val);
00106                 break;
00107         default:
00108                 G_OBJECT_WARN_INVALID_PROPERTY_ID(object, param_id, pspec);
00109                 break;
00110         }
00111 }
00112
00113 static void lib_cell_renderer_get_property(GObject      *object,
00114                                            guint        param_id,
00115                                            GValue       *value,
00116                                            GParamSpec   *pspec)
00117 {
00118         (void)value;
00119
00120         switch (param_id) {
00121         default:
00122                 G_OBJECT_WARN_INVALID_PROPERTY_ID(object, param_id, pspec);
00123                 break;
00124         }
00125 }
00126
00127 static GParamSpec *properties[PROP_COUNT];
00128
00129 void lib_cell_renderer_class_init(LibCellRendererClass *klass)
00130 {
00131         GObjectClass *oclass = G_OBJECT_CLASS(klass);
00132
00133         oclass->constructed = lib_cell_renderer_constructed;
00134         oclass->set_property = lib_cell_renderer_set_property;
00135         oclass->get_property = lib_cell_renderer_get_property;
00136
00137         properties[PROP_LIB] = g_param_spec_pointer("gds-lib", "gds-lib",
00138                                                     "Library reference to be displayed",
00139                                                     G_PARAM_WRITABLE);
00140         properties[PROP_CELL] = g_param_spec_pointer("gds-cell", "gds-cell",
00141                                                      "Cell reference to be displayed",
00142                                                      G_PARAM_WRITABLE);
00143         properties[PROP_ERROR_LEVEL] = g_param_spec_uint("error-level", "error-level",
00144                                                          "Error level of this cell", 0, 255, 0,
      G_PARAM_WRITABLE);
00145
00146         g_object_class_install_properties(oclass, PROP_COUNT, properties);
00147 }
00148
00149 GtkCellRenderer *lib_cell_renderer_new()
00150 {
00151         return GTK_CELL_RENDERER(g_object_new(TYPE_LIB_CELL_RENDERER, NULL));
00152 }
00153
```

13.3 command-line.c File Reference

Function to render according to command line parameters.

```
#include <stdio.h>
#include <glib/gi18n.h>
#include <gds-render/command-line.h>
#include <gds-render/gds-utils/gds-parser.h>
#include <gds-render/layer/layer-settings.h>
#include <gds-render/output-renderers/cairo-renderer.h>
#include <gds-render/output-renderers/latex-renderer.h>
#include <gds-render/output-renderers/external-renderer.h>
#include <gds-render/gds-utils/gds-tree-checker.h>
```

Include dependency graph for command-line.c:

Functions

- static int string_array_count (char **string_array)
- static int create_renderers (char **renderers, char **output_file_names, gboolean tex_layers, gboolean tex_standalone, const struct external_renderer_params *ext_params, GList **renderer_list, LayerSettings *layer_settings)
- static struct gds_cell * find_gds_cell_in_lib (struct gds_library *lib, const char *cell_name)
- int command_line_convert_gds (const char *gds_name, const char *cell_name, char **renderers, char **output_file_names, const char *layer_file, struct external_renderer_params *ext_param, gboolean tex_standalone, gboolean tex_layers, double scale)

  Convert GDS according to command line parameters.

13.3.1 Detailed Description

Function to render according to command line parameters.

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file command-line.c.

13.4 command-line.c

Go to the documentation of this file.

```
00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018  Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
```

```
00017  * along with GDSII-Converter.  If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00031 #include <stdio.h>
00032 #include <glib/gi18n.h>
00033
00034 #include <gds-render/command-line.h>
00035 #include <gds-render/gds-utils/gds-parser.h>
00036 #include <gds-render/layer/layer-settings.h>
00037 #include <gds-render/output-renderers/cairo-renderer.h>
00038 #include <gds-render/output-renderers/latex-renderer.h>
00039 #include <gds-render/output-renderers/external-renderer.h>
00040 #include <gds-render/gds-utils/gds-tree-checker.h>
00041
00042 static int string_array_count(char **string_array)
00043 {
00044         int count;
00045
00046         if (!string_array)
00047                 return 0;
00048
00049         for (count = 0; *string_array; string_array++)
00050                 count++;
00051
00052         return count;
00053 }
00054
00055 static int create_renderers(char **renderers,
00056                             char **output_file_names,
00057                             gboolean tex_layers,
00058                             gboolean tex_standalone,
00059                             const struct external_renderer_params *ext_params,
00060                             GList **renderer_list,
00061                             LayerSettings *layer_settings)
00062 {
00063         char **renderer_iter;
00064         char *current_renderer;
00065         int idx;
00066         char *current_out_file;
00067         int count_render, count_out;
00068         GdsOutputRenderer *output_renderer;
00069
00070         if (!renderer_list)
00071                 return -1;
00072
00073         if (!renderers || !output_file_names) {
00074                 fprintf(stderr, _("Please specify renderers and file names\n"));
00075                 return -1;
00076         }
00077
00078         count_render = string_array_count(renderers);
00079         count_out = string_array_count(output_file_names);
00080         if (count_render != count_out) {
00081                 fprintf(stderr, _("Count of renderers %d does not match count of output file names
      %d\n"),
00082                         count_render, count_out);
00083                 return -1;
00084         }
00085
00086         /* Parse cmd line parameters */
00087         for (renderer_iter = renderers, idx = 0; *renderer_iter; renderer_iter++, idx++) {
00088                 current_renderer = *renderer_iter;
00089                 current_out_file = output_file_names[idx];
00090
00091                 /* File valid ? */
00092                 if (!current_out_file || !current_out_file[0])
00093                         continue;
00094
00095                 if (!strcmp(current_renderer, "tikz")) {
00096                         output_renderer =
      GDS_RENDER_OUTPUT_RENDERER(latex_renderer_new_with_options(tex_layers,
00097
      tex_standalone));
00098                 } else if (!strcmp(current_renderer, "pdf")) {
00099                         output_renderer = GDS_RENDER_OUTPUT_RENDERER(cairo_renderer_new_pdf());
00100                 } else if (!strcmp(current_renderer, "svg")) {
00101                         output_renderer = GDS_RENDER_OUTPUT_RENDERER(cairo_renderer_new_svg());
00102                 } else if (!strcmp(current_renderer, "ext")) {
00103                         if (!ext_params->so_path) {
00104                                 fprintf(stderr, _("Please specify shared object for external renderer.
      Will ignore this renderer.\n"));
00105                                 continue;
00106                         }
00107                         output_renderer = GDS_RENDER_OUTPUT_RENDERER(
00108
      external_renderer_new_with_so_and_param(ext_params->so_path,
00109
```

```
      ext_params->cli_params));
00110                 } else {
00111                         continue;
00112                 }
00113
00114                 gds_output_renderer_set_output_file(output_renderer, current_out_file);
00115                 gds_output_renderer_set_layer_settings(output_renderer, layer_settings);
00116                 *renderer_list = g_list_append(*renderer_list, output_renderer);
00117         }
00118
00119         return 0;
00120 }
00121
00122 static struct gds_cell *find_gds_cell_in_lib(struct gds_library *lib, const char *cell_name)
00123 {
00124         GList *cell_list;
00125         struct gds_cell *return_cell = NULL;
00126         struct gds_cell *temp_cell;
00127
00128         for (cell_list = lib->cells; cell_list; cell_list = g_list_next(cell_list)) {
00129                 temp_cell = (struct gds_cell *)cell_list->data;
00130                 if (!strncmp(temp_cell->name, cell_name, CELL_NAME_MAX)) {
00131                         return_cell = temp_cell;
00132                         break;
00133                 }
00134         }
00135         return return_cell;
00136 }
00137
00138 int command_line_convert_gds(const char *gds_name,
00139                              const char *cell_name,
00140                              char **renderers,
00141                              char **output_file_names,
00142                              const char *layer_file,
00143                              struct external_renderer_params *ext_param,
00144                              gboolean tex_standalone,
00145                              gboolean tex_layers,
00146                              double scale)
00147 {
00148         int ret = -1;
00149         GList *libs = NULL;
00150         int res;
00151         GList *renderer_list = NULL;
00152         GList *list_iter;
00153         struct gds_library *first_lib;
00154         struct gds_cell *toplevel_cell = NULL;
00155         LayerSettings *layer_sett;
00156         GdsOutputRenderer *current_renderer;
00157
00158         /* Check if parameters are valid */
00159         if (!gds_name || !cell_name || !output_file_names || !layer_file || !renderers) {
00160                 printf(_("Probably missing argument. Check --help option\n"));
00161                 return -2;
00162         }
00163
00164         /* Load layer_settings */
00165         layer_sett = layer_settings_new();
00166         layer_settings_load_from_csv(layer_sett, layer_file);
00167
00168         /* Create renderers */
00169         if (create_renderers(renderers, output_file_names, tex_layers, tex_standalone,
00170                              ext_param, &renderer_list, layer_sett))
00171                 goto ret_destroy_layer_mapping;
00172
00173
00174         /* Load GDS */
00175         clear_lib_list(&libs);
00176         res = parse_gds_from_file(gds_name, &libs);
00177         if (res)
00178                 goto ret_destroy_library_list;
00179
00180         /* find_cell in first library. */
00181         if (!libs)
00182                 goto ret_clear_renderers;
00183
00184         first_lib = (struct gds_library *)libs->data;
00185         if (!first_lib) {
00186                 fprintf(stderr, _("No library in library list. This should not happen.\n"));
00187                 /* This is safe. Library destruction can handle an empty list element */
00188                 goto ret_destroy_library_list;
00189         }
00190
00191         /* Find cell in first library */
00192         toplevel_cell = find_gds_cell_in_lib(first_lib, cell_name);
00193
00194         if (!toplevel_cell) {
00195                 printf(_("Couldn't find cell in first library!\n"));
```

```
00196                 goto ret_destroy_library_list;
00197         }
00198
00199         /* Check if cell passes vital checks */
00200         res = gds_tree_check_reference_loops(toplevel_cell->parent_library);
00201         if (res < 0) {
00202                 fprintf(stderr, _("Checking library %s failed.\n"), first_lib->name);
00203                 goto ret_destroy_library_list;
00204         } else if (res > 0) {
00205                 fprintf(stderr, _("%d reference loops found.\n"), res);
00206
00207                 /* do further checking if the specified cell and/or its subcells are affected */
00208                 if (toplevel_cell->checks.affected_by_reference_loop == 1) {
00209                         fprintf(stderr, _("Cell is affected by reference loop. Abort!\n"));
00210                         goto ret_destroy_library_list;
00211                 }
00212         }
00213
00214         if (toplevel_cell->checks.affected_by_reference_loop == GDS_CELL_CHECK_NOT_RUN)
00215                 fprintf(stderr, _("Cell was not checked. This should not happen. Please report this
      issue. Will continue either way.\n"));
00216
00217         /* Note: unresolved references are not an abort condition.
00218          * Deal with it.
00219          */
00220
00221         /* Execute all rendererer instances */
00222         for (list_iter = renderer_list; list_iter; list_iter = list_iter->next) {
00223                 current_renderer = GDS_RENDER_OUTPUT_RENDERER(list_iter->data);
00224                 gds_output_renderer_render_output(current_renderer, toplevel_cell, scale);
00225         }
00226
00227 ret_destroy_library_list:
00228         clear_lib_list(&libs);
00229 ret_clear_renderers:
00230         for (list_iter = renderer_list; list_iter; list_iter = list_iter->next)
00231                 g_object_unref(list_iter->data);
00232 ret_destroy_layer_mapping:
00233         g_object_unref(layer_sett);
00234         return ret;
00235 }
00236
```

13.5 activity-bar.dox File Reference

13.6 cairo-renderer.dox File Reference

13.7 command-line.dox File Reference

13.8 compilation.dox File Reference

13.9 external-renderer.dox File Reference

13.10 gds-output-renderer.dox File Reference

13.11 geometric.dox File Reference

13.12 gui.dox File Reference

13.13 latex-renderer.dox File Reference

13.14 layer-selector.dox File Reference

13.15 lib-cell-renderer.dox File Reference

13.16 lmf-spec.dox File Reference

13.17 main-page.dox File Reference

13.18 plugins.dox File Reference

13.19 usage.dox File Reference

13.20 versioning.dox File Reference

13.21 widgets.dox File Reference

13.22 gds-render-gui.c File Reference

Handling of GUI.

```
#include <stdio.h>
#include <gtk/gtk.h>
#include <glib/gi18n.h>
#include <gds-render/gds-render-gui.h>
#include <gds-render/gds-utils/gds-parser.h>
#include <gds-render/gds-utils/gds-tree-checker.h>
#include <gds-render/layer/layer-selector.h>
#include <gds-render/widgets/activity-bar.h>
#include <gds-render/cell-selector/lib-cell-renderer.h>
#include <gds-render/output-renderers/latex-renderer.h>
#include <gds-render/output-renderers/cairo-renderer.h>
#include <gds-render/widgets/conv-settings-dialog.h>
#include <gds-render/geometric/cell-geometrics.h>
#include <gds-render/version.h>
```

Include dependency graph for gds-render-gui.c:

Data Structures

- struct gui_button_states
- struct _GdsRenderGui

Enumerations

- enum cell_store_columns { CELL_SEL_LIBRARY = 0 , CELL_SEL_CELL , CELL_SEL_CELL_ERROR_STATE , CELL_SEL_COLUMN_COUNT }

  Columns of selection tree view.
- enum gds_render_gui_signal_sig_ids { SIGNAL_WINDOW_CLOSED = 0 , SIGNAL_COUNT }

Functions

- static gboolean on_window_close (gpointer window, GdkEvent *event, gpointer user)

  Main window close event.
- static gboolean tree_sel_func (GtkTreeSelection *selection, GtkTreeModel *model, GtkTreePath *path, gboolean path_currently_selected, gpointer data)

  This function only allows valid cells to be selected.
- static void cell_tree_view_change_filter (GtkWidget *entry, gpointer data)

  Trigger refiltering of cell filter.
- static gboolean cell_store_filter_visible_func (GtkTreeModel *model, GtkTreeIter *iter, gpointer data)

  cell_store_filter_visible_func Decides whether an element of the tree model `model` is visible.
- int gds_render_gui_setup_cell_selector (GdsRenderGui *self)

  Setup a GtkTreeView with the necessary columns.
- static void on_load_gds (gpointer button, gpointer user)

  Callback function of Load GDS button.
- static void process_button_state_changes (GdsRenderGui *self)
- static void on_auto_color_clicked (gpointer button, gpointer user)

  Callback for auto coloring button.
- static void async_rendering_finished_callback (GdsOutputRenderer *renderer, gpointer gui)
- static void async_rendering_status_update_callback (GdsOutputRenderer *renderer, const char *status_↩ message, gpointer data)
- static void on_convert_clicked (gpointer button, gpointer user)

  Convert button callback.

- static void cell_tree_view_activated (gpointer tree_view, GtkTreePath *path, GtkTreeViewColumn *column, gpointer user)

  cell_tree_view_activated Callback for 'double click' on cell selector element
- static void cell_selection_changed (GtkTreeSelection *sel, GdsRenderGui *self)

  Callback for cell-selection change event.
- static void sort_up_callback (GtkWidget *widget, gpointer user)
- static void sort_down_callback (GtkWidget *widget, gpointer user)
- static void gds_render_gui_dispose (GObject *gobject)
- static void gds_render_gui_class_init (GdsRenderGuiClass *klass)
- static void on_select_all_layers_clicked (GtkWidget *button, gpointer user_data)

  Callback for the 'select all layers'-button.
- static gboolean auto_naming_ask_for_override (GdsRenderGui *gui)
- static void auto_naming_clicked (GtkWidget *button, gpointer user_data)
- GtkWindow * gds_render_gui_get_main_window (GdsRenderGui *gui)

  Get main window.
- static void gds_render_gui_init (GdsRenderGui *self)
- GdsRenderGui * gds_render_gui_new ()

  Create new GdsRenderGui Object.

Variables

- static guint gds_render_gui_signals [SIGNAL_COUNT]

13.22.1 Detailed Description

Handling of GUI.

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file gds-render-gui.c.

13.23 gds-render-gui.c

Go to the documentation of this file.

```
00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018  Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter.  If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00030 #include <stdio.h>
00031 #include <gtk/gtk.h>
```

```
00032 #include <glib/gi18n.h>
00033
00034 #include <gds-render/gds-render-gui.h>
00035 #include <gds-render/gds-utils/gds-parser.h>
00036 #include <gds-render/gds-utils/gds-tree-checker.h>
00037 #include <gds-render/layer/layer-selector.h>
00038 #include <gds-render/widgets/activity-bar.h>
00039 #include <gds-render/cell-selector/lib-cell-renderer.h>
00040 #include <gds-render/output-renderers/latex-renderer.h>
00041 #include <gds-render/output-renderers/cairo-renderer.h>
00042 #include <gds-render/widgets/conv-settings-dialog.h>
00043 #include <gds-render/geometric/cell-geometrics.h>
00044 #include <gds-render/version.h>
00045
00047 enum cell_store_columns {
00048         CELL_SEL_LIBRARY = 0,
00049         CELL_SEL_CELL,
00050         CELL_SEL_CELL_ERROR_STATE,
00051         CELL_SEL_COLUMN_COUNT
00052 };
00053
00054 enum gds_render_gui_signal_sig_ids {SIGNAL_WINDOW_CLOSED = 0, SIGNAL_COUNT};
00055
00056 static guint gds_render_gui_signals[SIGNAL_COUNT];
00057
00058 struct gui_button_states {
00059         gboolean rendering_active;
00060         gboolean valid_cell_selected;
00061 };
00062
00063 struct _GdsRenderGui {
00064         /* Parent GObject */
00065         GObject parent;
00066
00067         /* Custom fields */
00068         GtkWindow *main_window;
00069         GtkWidget *convert_button;
00070         GtkWidget *open_button;
00071         GtkWidget *load_layer_button;
00072         GtkWidget *save_layer_button;
00073         GtkWidget *select_all_button;
00074         GtkTreeStore *cell_tree_store;
00075         GtkTreeModelFilter *cell_filter;
00076         GtkWidget *cell_search_entry;
00077         LayerSelector *layer_selector;
00078         GtkTreeView *cell_tree_view;
00079         GList *gds_libraries;
00080         ActivityBar *activity_status_bar;
00081         struct render_settings render_dialog_settings;
00082         ColorPalette *palette;
00083         struct gui_button_states button_state_data;
00084 };
00085
00086 G_DEFINE_TYPE(GdsRenderGui, gds_render_gui, G_TYPE_OBJECT)
00087
00088
00095 static gboolean on_window_close(gpointer window, GdkEvent *event, gpointer user)
00096 {
00097         GdsRenderGui *self;
00098         (void)event;
00099
00100         self = RENDERER_GUI(user);
00101         /* Don't close window in case of error */
00102         if (!self)
00103                 return TRUE;
00104
00105         /* Close Window. Leads to termination of the program/the current instance */
00106         g_clear_object(&self->main_window);
00107         gtk_widget_destroy(GTK_WIDGET(window));
00108
00109         /* Delete loaded library data */
00110         clear_lib_list(&self->gds_libraries);
00111
00112         g_signal_emit(self, gds_render_gui_signals[SIGNAL_WINDOW_CLOSED], 0);
00113
00114         return TRUE;
00115 }
00116
00126 static gboolean tree_sel_func(GtkTreeSelection *selection,
00127                               GtkTreeModel *model,
00128                               GtkTreePath *path,
00129                               gboolean path_currently_selected,
00130                               gpointer data)
00131 {
00132         GtkTreeIter iter;
00133         struct gds_cell *cell;
00134         unsigned int error_level;
```

```
00135         gboolean ret = FALSE;
00136         (void)selection;
00137         (void)path_currently_selected;
00138         (void)data;
00139 
00140         gtk_tree_model_get_iter(model, &iter, path);
00141         gtk_tree_model_get(model, &iter, CELL_SEL_CELL, &cell, CELL_SEL_CELL_ERROR_STATE,
      &error_level, -1);
00142 
00143         /* Allow only rows with _valid_ cell to be selected */
00144         if (cell) {
00145                 /* Cell available. Check if it passed the critical checks */
00146                 if (!(error_level & LIB_CELL_RENDERER_ERROR_ERR))
00147                         ret = TRUE;
00148         }
00149 
00150         return ret;
00151 }
00152 
00158 static void cell_tree_view_change_filter(GtkWidget *entry, gpointer data)
00159 {
00160         GdsRenderGui *self = RENDERER_GUI(data);
00161         (void)entry;
00162 
00163         gtk_tree_model_filter_refilter(self->cell_filter);
00164 }
00165 
00174 static gboolean cell_store_filter_visible_func(GtkTreeModel *model, GtkTreeIter *iter, gpointer data)
00175 {
00176         GdsRenderGui *self;
00177         struct gds_cell *cell;
00178         struct gds_library *lib;
00179         gboolean result = FALSE;
00180         const char *search_string;
00181 
00182         self = RENDERER_GUI(data);
00183         g_return_val_if_fail(RENDERER_IS_GUI(self), FALSE);
00184 
00185         if (!model || !iter)
00186                 goto exit_filter;
00187 
00188         gtk_tree_model_get(model, iter, CELL_SEL_CELL, &cell, CELL_SEL_LIBRARY, &lib, -1);
00189 
00190     /* Show always, if this is a pure lib entry */
00191     if (lib && !cell) {
00192                 result = TRUE;
00193                 goto exit_filter;
00194         }
00195 
00196         if (!cell)
00197                 goto exit_filter;
00198 
00199         search_string = gtk_entry_get_text(GTK_ENTRY(self->cell_search_entry));
00200 
00201         /* Show all, if field is empty */
00202         if (!strlen(search_string))
00203                 result = TRUE;
00204 
00205         if (strstr(cell->name, search_string))
00206                 result = TRUE;
00207 
00208         gtk_tree_view_expand_all(self->cell_tree_view);
00209 
00210 exit_filter:
00211         return result;
00212 }
00213 
00218 int gds_render_gui_setup_cell_selector(GdsRenderGui *self)
00219 {
00220         GtkCellRenderer *render_cell;
00221         GtkCellRenderer *render_lib;
00222         GtkTreeViewColumn *column;
00223 
00224         self->cell_tree_store = gtk_tree_store_new(CELL_SEL_COLUMN_COUNT, G_TYPE_POINTER,
00225                                                    G_TYPE_POINTER, G_TYPE_UINT);
00226 
00227         /* Searching */
00228         self->cell_filter = GTK_TREE_MODEL_FILTER(
00229                                 gtk_tree_model_filter_new(GTK_TREE_MODEL(self->cell_tree_store),
      NULL));
00230 
00231         gtk_tree_model_filter_set_visible_func(self->cell_filter,
00232 
      (GtkTreeModelFilterVisibleFunc)cell_store_filter_visible_func,
00233                                                self, NULL);
00234         g_signal_connect(GTK_SEARCH_ENTRY(self->cell_search_entry), "search-changed",
00235                          G_CALLBACK(cell_tree_view_change_filter), self);
```

```
00236
00237         gtk_tree_view_set_model(self->cell_tree_view, GTK_TREE_MODEL(self->cell_filter));
00238
00239         render_cell = lib_cell_renderer_new();
00240         render_lib = lib_cell_renderer_new();
00241
00242         column = gtk_tree_view_column_new_with_attributes(_("Library"), render_lib, "gds-lib",
      CELL_SEL_LIBRARY, NULL);
00243         gtk_tree_view_append_column(self->cell_tree_view, column);
00244
00245         column = gtk_tree_view_column_new_with_attributes(_("Cell"), render_cell, "gds-cell",
      CELL_SEL_CELL,
00246                                                           "error-level", CELL_SEL_CELL_ERROR_STATE,
      NULL);
00247         gtk_tree_view_append_column(self->cell_tree_view, column);
00248
00249         /* Callback for selection
00250          * This prevents selecting a library
00251          */
00252         gtk_tree_selection_set_select_function(gtk_tree_view_get_selection(self->cell_tree_view),
00253                                                tree_sel_func, NULL, NULL);
00254
00255         return 0;
00256 }
00257
00263 static void on_load_gds(gpointer button, gpointer user)
00264 {
00265         GList *cell;
00266         GtkTreeIter libiter;
00267         GtkTreeIter celliter;
00268         GList *lib;
00269         struct gds_library *gds_lib;
00270         struct gds_cell *gds_c;
00271         GdsRenderGui *self;
00272         GtkWidget *open_dialog;
00273         GtkFileChooser *file_chooser;
00274         GtkFileFilter *filter;
00275         GtkStyleContext *button_style;
00276         gint dialog_result;
00277         int gds_result;
00278         char *filename;
00279         unsigned int cell_error_level;
00280
00281         self = RENDERER_GUI(user);
00282         if (!self)
00283                 return;
00284
00285         open_dialog = gtk_file_chooser_dialog_new(_("Open GDSII File"), self->main_window,
00286                                                   GTK_FILE_CHOOSER_ACTION_OPEN,
00287                                                   _("Cancel"), GTK_RESPONSE_CANCEL,
00288                                                   _("Open GDSII"), GTK_RESPONSE_ACCEPT,
00289                                                   NULL);
00290         file_chooser = GTK_FILE_CHOOSER(open_dialog);
00291
00292         /* Add GDS II Filter */
00293         filter = gtk_file_filter_new();
00294         gtk_file_filter_add_pattern(filter, "*.gds");
00295         gtk_file_filter_set_name(filter, _("GDSII-Files"));
00296         gtk_file_chooser_add_filter(file_chooser, filter);
00297
00298         dialog_result = gtk_dialog_run(GTK_DIALOG(open_dialog));
00299
00300         if (dialog_result != GTK_RESPONSE_ACCEPT)
00301                 goto end_destroy;
00302
00303         /* Get File name */
00304         filename = gtk_file_chooser_get_filename(file_chooser);
00305
00306         gtk_tree_store_clear(self->cell_tree_store);
00307         clear_lib_list(&self->gds_libraries);
00308
00309         /* Parse new GDSII file */
00310         gds_result = parse_gds_from_file(filename, &self->gds_libraries);
00311
00312         /* Delete file name afterwards */
00313         g_free(filename);
00314         if (gds_result)
00315                 goto end_destroy;
00316
00317         /* remove suggested action from Open button */
00318         button_style = gtk_widget_get_style_context(GTK_WIDGET(button));
00319         gtk_style_context_remove_class(button_style, "suggested-action");
00320
00321         for (lib = self->gds_libraries; lib != NULL; lib = lib->next) {
00322                 gds_lib = (struct gds_library *)lib->data;
00323                 /* Create top level iter */
00324                 gtk_tree_store_append(self->cell_tree_store, &libiter, NULL);
```

```
00325
00326                 gtk_tree_store_set(self->cell_tree_store, &libiter,
00327                                    CELL_SEL_LIBRARY, gds_lib,
00328                                    -1);
00329
00330                 /* Check this library. This might take a while */
00331                 (void)gds_tree_check_cell_references(gds_lib);
00332                 (void)gds_tree_check_reference_loops(gds_lib);
00333
00334                 for (cell = gds_lib->cells; cell != NULL; cell = cell->next) {
00335                         gds_c = (struct gds_cell *)cell->data;
00336                         gtk_tree_store_append(self->cell_tree_store, &celliter, &libiter);
00337
00338                         /* Get the checking results for this cell */
00339                         cell_error_level = 0;
00340                         if (gds_c->checks.unresolved_child_count)
00341                                 cell_error_level |= LIB_CELL_RENDERER_ERROR_WARN;
00342
00343                         /* Check if it is completely b0rken */
00344                         if (gds_c->checks.affected_by_reference_loop)
00345                                 cell_error_level |= LIB_CELL_RENDERER_ERROR_ERR;
00346
00347                         /* Add cell to tree store model */
00348                         gtk_tree_store_set(self->cell_tree_store, &celliter,
00349                                            CELL_SEL_CELL, gds_c,
00350                                            CELL_SEL_CELL_ERROR_STATE, cell_error_level,
00351                                            CELL_SEL_LIBRARY, gds_c->parent_library,
00352                                            -1);
00353                 } /* for cells */
00354         } /* for libraries */
00355
00356         /* Create Layers in Layer Box */
00357         layer_selector_generate_layer_widgets(self->layer_selector, self->gds_libraries);
00358
00359 end_destroy:
00360         /* Destroy dialog and filter */
00361         gtk_widget_destroy(open_dialog);
00362 }
00363
00364 static void process_button_state_changes(GdsRenderGui *self)
00365 {
00366         gboolean convert_button_state = FALSE;
00367         gboolean open_gds_button_state = FALSE;
00368
00369         /* Calculate states */
00370         if (!self->button_state_data.rendering_active) {
00371                 open_gds_button_state = TRUE;
00372                 if (self->button_state_data.valid_cell_selected)
00373                         convert_button_state = TRUE;
00374         }
00375
00376         /* Apply states */
00377         gtk_widget_set_sensitive(self->convert_button, convert_button_state);
00378         gtk_widget_set_sensitive(self->open_button, open_gds_button_state);
00379 }
00380
00386 static void on_auto_color_clicked(gpointer button, gpointer user)
00387 {
00388         GdsRenderGui *self;
00389         (void)button;
00390
00391         self = RENDERER_GUI(user);
00392         layer_selector_auto_color_layers(self->layer_selector, self->palette, 1.0);
00393 }
00394
00395 static void async_rendering_finished_callback(GdsOutputRenderer *renderer, gpointer gui)
00396 {
00397         GdsRenderGui *self;
00398
00399         self = RENDERER_GUI(gui);
00400
00401         self->button_state_data.rendering_active = FALSE;
00402         process_button_state_changes(self);
00403         activity_bar_set_ready(self->activity_status_bar);
00404
00405         g_object_unref(renderer);
00406 }
00407
00408 static void async_rendering_status_update_callback(GdsOutputRenderer *renderer,
00409                                                    const char *status_message,
00410                                                    gpointer data)
00411 {
00412         GdsRenderGui *gui;
00413         (void)renderer;
00414
00415         gui = RENDERER_GUI(data);
00416
```

```
00417         activity_bar_set_busy(gui->activity_status_bar, status_message);
00418 }
00419
00425 static void on_convert_clicked(gpointer button, gpointer user)
00426 {
00427         (void)button;
00428         GdsRenderGui *self;
00429         GtkTreeSelection *selection;
00430         GtkTreeIter iter;
00431         GtkTreeModel *model;
00432         struct gds_cell *cell_to_render;
00433         GtkWidget *dialog;
00434         RendererSettingsDialog *settings;
00435         GtkFileFilter *filter;
00436         gint res;
00437         char *file_name;
00438         union bounding_box cell_box;
00439         unsigned int height, width;
00440         struct render_settings *sett;
00441         LayerSettings *layer_settings;
00442         GdsOutputRenderer *render_engine;
00443
00444         self = RENDERER_GUI(user);
00445
00446         if (!self)
00447                 return;
00448
00449         /* Abort if rendering is already active */
00450         if (self->button_state_data.rendering_active == TRUE)
00451                 return;
00452
00453         sett = &self->render_dialog_settings;
00454
00455         /* Get selected cell */
00456         selection = gtk_tree_view_get_selection(self->cell_tree_view);
00457         if (gtk_tree_selection_get_selected(selection, &model, &iter) == FALSE)
00458                 return;
00459
00460         gtk_tree_model_get(model, &iter, CELL_SEL_CELL, &cell_to_render, -1);
00461
00462         if (!cell_to_render)
00463                 return;
00464
00465         /* Get layers that are rendered */
00466         layer_settings = layer_selector_export_rendered_layer_info(self->layer_selector);
00467
00468         /* Calculate cell size in DB units */
00469         bounding_box_prepare_empty(&cell_box);
00470         calculate_cell_bounding_box(&cell_box, cell_to_render);
00471
00472         /* Calculate size in database units
00473          * Note that the results are bound to be positive,
00474          * so casting them to unsigned int is absolutely valid
00475          */
00476         height = (unsigned int)(cell_box.vectors.upper_right.y - cell_box.vectors.lower_left.y);
00477         width = (unsigned int)(cell_box.vectors.upper_right.x - cell_box.vectors.lower_left.x);
00478
00479         /* Show settings dialog */
00480         settings = renderer_settings_dialog_new(GTK_WINDOW(self->main_window));
00481         renderer_settings_dialog_set_settings(settings, sett);
00482         renderer_settings_dialog_set_database_unit_scale(settings,
      cell_to_render->parent_library->unit_in_meters);
00483         renderer_settings_dialog_set_cell_height(settings, height);
00484         renderer_settings_dialog_set_cell_width(settings, width);
00485         g_object_set(G_OBJECT(settings), "cell-name", cell_to_render->name, NULL);
00486
00487         res = gtk_dialog_run(GTK_DIALOG(settings));
00488         if (res == GTK_RESPONSE_OK) {
00489                 renderer_settings_dialog_get_settings(settings, sett);
00490                 gtk_widget_destroy(GTK_WIDGET(settings));
00491         } else {
00492                 gtk_widget_destroy(GTK_WIDGET(settings));
00493                 goto ret_layer_destroy;
00494         }
00495
00496         /* save file dialog */
00497         dialog = gtk_file_chooser_dialog_new((sett->renderer == RENDERER_LATEX_TIKZ
00498                                               ? "Save LaTeX File" : "Save PDF"),
00499                                              GTK_WINDOW(self->main_window),
      GTK_FILE_CHOOSER_ACTION_SAVE,
00500                                              "Cancel", GTK_RESPONSE_CANCEL, "Save",
      GTK_RESPONSE_ACCEPT, NULL);
00501         /* Set file filter according to settings */
00502         filter = gtk_file_filter_new();
00503         switch (sett->renderer) {
00504         case RENDERER_LATEX_TIKZ:
00505                 gtk_file_filter_add_pattern(filter, "*.tex");
```

```
00506                 gtk_file_filter_set_name(filter, "LaTeX-Files");
00507                 break;
00508         case RENDERER_CAIROGRAPHICS_PDF:
00509                 gtk_file_filter_add_pattern(filter, "*.pdf");
00510                 gtk_file_filter_set_name(filter, "PDF-Files");
00511                 break;
00512         case RENDERER_CAIROGRAPHICS_SVG:
00513                 gtk_file_filter_add_pattern(filter, "*.svg");
00514                 gtk_file_filter_set_name(filter, "SVG-Files");
00515                 break;
00516         }
00517
00518         gtk_file_chooser_add_filter(GTK_FILE_CHOOSER(dialog), filter);
00519
00520         gtk_file_chooser_set_do_overwrite_confirmation(GTK_FILE_CHOOSER(dialog), TRUE);
00521
00522         res = gtk_dialog_run(GTK_DIALOG(dialog));
00523         if (res == GTK_RESPONSE_ACCEPT) {
00524                 file_name = gtk_file_chooser_get_filename(GTK_FILE_CHOOSER(dialog));
00525                 gtk_widget_destroy(dialog);
00526
00527                 switch (sett->renderer) {
00528                 case RENDERER_LATEX_TIKZ:
00529                         render_engine =
00530
      GDS_RENDER_OUTPUT_RENDERER(latex_renderer_new_with_options(sett->tex_pdf_layers,
00531
      sett->tex_standalone));
00532                         break;
00533                 case RENDERER_CAIROGRAPHICS_SVG:
00534                         render_engine = GDS_RENDER_OUTPUT_RENDERER(cairo_renderer_new_svg());
00535                         break;
00536                 case RENDERER_CAIROGRAPHICS_PDF:
00537                         render_engine = GDS_RENDER_OUTPUT_RENDERER(cairo_renderer_new_pdf());
00538                         break;
00539                 default:
00540                         /* Abort rendering */
00541                         render_engine = NULL;
00542                         break;
00543                 }
00544
00545                 if (render_engine) {
00546                         gds_output_renderer_set_output_file(render_engine, file_name);
00547                         gds_output_renderer_set_layer_settings(render_engine, layer_settings);
00548                         /* Prevent user from overwriting library or triggering additional conversion
      */
00549                         self->button_state_data.rendering_active = TRUE;
00550                         process_button_state_changes(self);
00551
00552                         g_signal_connect(render_engine, "async-finished",
      G_CALLBACK(async_rendering_finished_callback),
00553                                          self);
00554
00555                         activity_bar_set_busy(self->activity_status_bar, _("Rendering cell..."));
00556
00557                         g_signal_connect(render_engine, "progress-changed",
00558                                          G_CALLBACK(async_rendering_status_update_callback), self);
00559                         gds_output_renderer_render_output_async(render_engine, cell_to_render,
      sett->scale);
00560                 }
00561                 g_free(file_name);
00562         } else {
00563                 gtk_widget_destroy(dialog);
00564         }
00565 ret_layer_destroy:
00566         g_object_unref(layer_settings);
00567 }
00568
00576 static void cell_tree_view_activated(gpointer tree_view, GtkTreePath *path,
00577                                      GtkTreeViewColumn *column, gpointer user)
00578 {
00579         (void)tree_view;
00580         (void)path;
00581         (void)column;
00582
00583         on_convert_clicked(NULL, user);
00584 }
00585
00594 static void cell_selection_changed(GtkTreeSelection *sel, GdsRenderGui *self)
00595 {
00596         GtkTreeModel *model = NULL;
00597         GtkTreeIter iter;
00598
00599         if (gtk_tree_selection_get_selected(sel, &model, &iter)) {
00600                 /* Node selected. Show button */
00601                 self->button_state_data.valid_cell_selected = TRUE;
00602         } else {
```

```
00603                   self->button_state_data.valid_cell_selected = FALSE;
00604           }
00605
00606           process_button_state_changes(self);
00607   }
00608
00609   static void sort_up_callback(GtkWidget *widget, gpointer user)
00610   {
00611           (void)widget;
00612           GdsRenderGui *self;
00613
00614           self = RENDERER_GUI(user);
00615           if (!self)
00616                   return;
00617           layer_selector_force_sort(self->layer_selector, LAYER_SELECTOR_SORT_UP);
00618   }
00619
00620   static void sort_down_callback(GtkWidget *widget, gpointer user)
00621   {
00622           (void)widget;
00623           GdsRenderGui *self;
00624
00625           self = RENDERER_GUI(user);
00626           if (!self)
00627                   return;
00628           layer_selector_force_sort(self->layer_selector, LAYER_SELECTOR_SORT_DOWN);
00629   }
00630
00631   static void gds_render_gui_dispose(GObject *gobject)
00632   {
00633           GdsRenderGui *self;
00634
00635           self = RENDERER_GUI(gobject);
00636
00637           clear_lib_list(&self->gds_libraries);
00638
00639           g_clear_object(&self->cell_tree_view);
00640           g_clear_object(&self->convert_button);
00641           g_clear_object(&self->layer_selector);
00642           g_clear_object(&self->cell_tree_store);
00643           g_clear_object(&self->cell_filter);
00644           g_clear_object(&self->cell_search_entry);
00645           g_clear_object(&self->activity_status_bar);
00646           g_clear_object(&self->palette);
00647           g_clear_object(&self->load_layer_button);
00648           g_clear_object(&self->save_layer_button);
00649           g_clear_object(&self->open_button);
00650           g_clear_object(&self->select_all_button);
00651
00652           if (self->main_window) {
00653                   g_signal_handlers_destroy(self->main_window);
00654                   gtk_widget_destroy(GTK_WIDGET(self->main_window));
00655                   self->main_window = NULL;
00656           }
00657
00658           /* Chain up */
00659           G_OBJECT_CLASS(gds_render_gui_parent_class)->dispose(gobject);
00660   }
00661
00662   static void gds_render_gui_class_init(GdsRenderGuiClass *klass)
00663   {
00664           GObjectClass *gobject_class = G_OBJECT_CLASS(klass);
00665
00666           gds_render_gui_signals[SIGNAL_WINDOW_CLOSED] =
00667                           g_signal_newv("window-closed", RENDERER_TYPE_GUI,
00668                                         G_SIGNAL_RUN_LAST | G_SIGNAL_NO_RECURSE,
00669                                         NULL,
00670                                         NULL,
00671                                         NULL,
00672                                         NULL,
00673                                         G_TYPE_NONE,
00674                                         0,
00675                                         NULL);
00676
00677           gobject_class->dispose = gds_render_gui_dispose;
00678   }
00679
00685   static void on_select_all_layers_clicked(GtkWidget *button, gpointer user_data)
00686   {
00687           GdsRenderGui *gui;
00688           (void)button;
00689
00690           gui = RENDERER_GUI(user_data);
00691           layer_selector_select_all_layers(gui->layer_selector, TRUE);
00692   }
00693
00694   static gboolean auto_naming_ask_for_override(GdsRenderGui *gui)
```

```
00695 {
00696         GtkDialog *dialog;
00697         gint dialog_result;
00698         gboolean overwrite = FALSE;
00699 
00700         g_return_val_if_fail(RENDERER_IS_GUI(gui), FALSE);
00701 
00702         /* Ask for overwrite */
00703         dialog = GTK_DIALOG(gtk_message_dialog_new(gui->main_window, GTK_DIALOG_USE_HEADER_BAR,
      GTK_MESSAGE_QUESTION,
00704                                                    GTK_BUTTONS_YES_NO, "Overwrite existing layer
      names?"));
00705         dialog_result = gtk_dialog_run(dialog);
00706         switch (dialog_result) {
00707         case GTK_RESPONSE_YES:
00708                 overwrite = TRUE;
00709                 break;
00710         case GTK_RESPONSE_NO: /* Expected fallthrough */
00711         default:
00712                 overwrite = FALSE;
00713                 break;
00714         }
00715         gtk_widget_destroy(GTK_WIDGET(dialog));
00716 
00717         return overwrite;
00718 }
00719 
00720 static void auto_naming_clicked(GtkWidget *button, gpointer user_data)
00721 {
00722         GdsRenderGui *gui;
00723         gboolean overwrite = FALSE;
00724         (void)button;
00725 
00726         gui = RENDERER_GUI(user_data);
00727 
00728         /* Don't do anything if the selector is empty. */
00729         if (!layer_selector_contains_elements(gui->layer_selector))
00730                 return;
00731 
00732         /* Ask, if names shall be overwritten, if they are not empty */
00733         if (layer_selector_num_of_named_elements(gui->layer_selector) > 0)
00734                 overwrite = auto_naming_ask_for_override(gui);
00735 
00736         layer_selector_auto_name_layers(gui->layer_selector, overwrite);
00737 }
00738 
00739 GtkWindow *gds_render_gui_get_main_window(GdsRenderGui *gui)
00740 {
00741         return gui->main_window;
00742 }
00743 
00744 static void gds_render_gui_init(GdsRenderGui *self)
00745 {
00746         GtkBuilder *main_builder;
00747         GtkWidget *listbox;
00748         GtkHeaderBar *header_bar;
00749         GtkWidget *sort_up_button;
00750         GtkWidget *sort_down_button;
00751         GtkWidget *activity_bar_box;
00752         GtkWidget *auto_color_button;
00753         GtkWidget *auto_naming_button;
00754 
00755         main_builder = gtk_builder_new_from_resource("/gui/main.glade");
00756 
00757         self->cell_tree_view = GTK_TREE_VIEW(gtk_builder_get_object(main_builder, "cell-tree"));
00758         self->cell_search_entry = GTK_WIDGET(gtk_builder_get_object(main_builder, "cell-search"));
00759 
00760         gds_render_gui_setup_cell_selector(self);
00761 
00762         self->main_window = GTK_WINDOW(gtk_builder_get_object(main_builder, "main-window"));
00763         self->open_button = GTK_WIDGET(gtk_builder_get_object(main_builder, "button-load-gds"));
00764         g_signal_connect(self->open_button,
00765                          "clicked", G_CALLBACK(on_load_gds), (gpointer)self);
00766 
00767         self->convert_button = GTK_WIDGET(gtk_builder_get_object(main_builder, "convert-button"));
00768         g_signal_connect(self->convert_button, "clicked", G_CALLBACK(on_convert_clicked),
      (gpointer)self);
00769 
00770         listbox = GTK_WIDGET(gtk_builder_get_object(main_builder, "layer-list"));
00771         /* Create layer selector */
00772         self->layer_selector = layer_selector_new(GTK_LIST_BOX(listbox));
00773 
00774         activity_bar_box = GTK_WIDGET(gtk_builder_get_object(main_builder, "activity-bar"));
00775 
00776         /* Callback for selection change of cell selector */
00777         g_signal_connect(G_OBJECT(gtk_tree_view_get_selection(self->cell_tree_view)), "changed",
00778                          G_CALLBACK(cell_selection_changed), self);
```

```
00779         g_signal_connect(self->cell_tree_view, "row-activated", G_CALLBACK(cell_tree_view_activated),
      self);
00780
00781         /* Set version in main window subtitle */
00782         header_bar = GTK_HEADER_BAR(gtk_builder_get_object(main_builder, "header-bar"));
00783         gtk_header_bar_set_subtitle(header_bar, _app_version_string);
00784
00785         /* Get layer sorting buttons and set callbacks */
00786         sort_up_button = GTK_WIDGET(gtk_builder_get_object(main_builder, "button-up-sort"));
00787         sort_down_button = GTK_WIDGET(gtk_builder_get_object(main_builder, "button-down-sort"));
00788
00789         g_signal_connect(sort_up_button, "clicked", G_CALLBACK(sort_up_callback), self);
00790         g_signal_connect(sort_down_button, "clicked", G_CALLBACK(sort_down_callback), self);
00791
00792         /* Set buttons for loading and saving */
00793         self->load_layer_button = GTK_WIDGET(gtk_builder_get_object(main_builder,
      "button-load-mapping"));
00794         self->save_layer_button = GTK_WIDGET(gtk_builder_get_object(main_builder,
      "button-save-mapping"));
00795         layer_selector_set_load_mapping_button(self->layer_selector, self->load_layer_button,
      self->main_window);
00796         layer_selector_set_save_mapping_button(self->layer_selector, self->save_layer_button,
      self->main_window);
00797
00798         /* Connect delete-event */
00799         g_signal_connect(GTK_WIDGET(self->main_window), "delete-event",
00800                          G_CALLBACK(on_window_close), self);
00801
00802         /* Create and apply ActivityBar */
00803         self->activity_status_bar = activity_bar_new();
00804         gtk_container_add(GTK_CONTAINER(activity_bar_box), GTK_WIDGET(self->activity_status_bar));
00805         gtk_widget_show(GTK_WIDGET(self->activity_status_bar));
00806
00807         /* Create color palette */
00808         self->palette = color_palette_new_from_resource("/data/color-palette.txt");
00809         auto_color_button = GTK_WIDGET(gtk_builder_get_object(main_builder, "auto-color-button"));
00810         g_signal_connect(auto_color_button, "clicked", G_CALLBACK(on_auto_color_clicked), self);
00811
00812
00813         /* Set default conversion/rendering settings */
00814         self->render_dialog_settings.scale = 1000;
00815         self->render_dialog_settings.renderer = RENDERER_LATEX_TIKZ;
00816         self->render_dialog_settings.tex_pdf_layers = FALSE;
00817         self->render_dialog_settings.tex_standalone = FALSE;
00818
00819         /* Get select all button and connect callback */
00820         self->select_all_button = GTK_WIDGET(gtk_builder_get_object(main_builder,
      "button-select-all"));
00821         g_signal_connect(self->select_all_button, "clicked", G_CALLBACK(on_select_all_layers_clicked),
      self);
00822
00823         /* Setup auto naming button */
00824         auto_naming_button = GTK_WIDGET(gtk_builder_get_object(main_builder, "button-auto-name"));
00825         g_signal_connect(auto_naming_button, "clicked", G_CALLBACK(auto_naming_clicked), self);
00826
00827         g_object_unref(main_builder);
00828
00829         /* Setup default button sensibility data */
00830         self->button_state_data.rendering_active = FALSE;
00831         self->button_state_data.valid_cell_selected = FALSE;
00832
00833         /* Reference all objects referenced by this object */
00834         g_object_ref(self->activity_status_bar);
00835         g_object_ref(self->main_window);
00836         g_object_ref(self->cell_tree_view);
00837         g_object_ref(self->convert_button);
00838         /* g_object_ref(self->layer_selector); <= This is already referenced by the _new() function */
00839         g_object_ref(self->cell_search_entry);
00840         /* g_object_ref(self->palette); */
00841         g_object_ref(self->open_button);
00842         g_object_ref(self->load_layer_button);
00843         g_object_ref(self->save_layer_button);
00844         g_object_ref(self->select_all_button);
00845 }
00846
00847 GdsRenderGui *gds_render_gui_new()
00848 {
00849         return RENDERER_GUI(g_object_new(RENDERER_TYPE_GUI, NULL));
00850 }
00851
```

13.24 gds-parser.c File Reference

Implementation of the GDS-Parser.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
#include <math.h>
#include <cairo.h>
#include <glib/gi18n.h>
#include <gds-render/gds-utils/gds-parser.h>
```

Include dependency graph for gds-parser.c:

Data Structures

- struct gds_cell_array_instance

  Struct representing an array instantiation.

Macros

- #define GDS_DEFAULT_UNITS (10E-9)

  Default units assumed for library.
- #define GDS_ERROR(fmt, ...) printf("[PARSE_ERROR] " fmt "\n", ##__VA_ARGS__)

  Print GDS error.
- #define GDS_WARN(fmt, ...) printf("[PARSE_WARNING] " fmt "\n", ##__VA_ARGS__)

  Print GDS warning.
- #define GDS_INF(fmt, ...)

Enumerations

- enum gds_record {
  INVALID = 0x0000 , HEADER = 0x0002 , BGNLIB = 0x0102 , LIBNAME = 0x0206 ,
  UNITS = 0x0305 , ENDLIB = 0x0400 , BGNSTR = 0x0502 , STRNAME = 0x0606 ,
  ENDSTR = 0x0700 , BOUNDARY = 0x0800 , PATH = 0x0900 , SREF = 0x0A00 ,
  ENDEL = 0x1100 , XY = 0x1003 , MAG = 0x1B05 , ANGLE = 0x1C05 ,
  SNAME = 0x1206 , STRANS = 0x1A01 , BOX = 0x2D00 , LAYER = 0x0D02 ,
  DATATYPE = 0x0E02 , WIDTH = 0x0F03 , PATHTYPE = 0x2102 , COLROW = 0x1302 ,
  AREF = 0x0B00 }

Functions

- static int name_cell_ref (struct gds_cell_instance *cell_inst, unsigned int bytes, char *data)

  Name cell reference.
- static int name_array_cell_ref (struct gds_cell_array_instance *cell_inst, unsigned int bytes, char *data)

  Name cell reference.
- static double gds_convert_double (const char *data)

  Convert GDS 8-byte real to double.
- static signed int gds_convert_signed_int (const char *data)

  Convert GDS INT32 to int.

- static int16_t gds_convert_signed_int16 (const char *data)

  Convert GDS INT16 to int16.
- static uint16_t gds_convert_unsigned_int16 (const char *data)

  Convert GDS UINT16 String to uint16.
- static GList * append_library (GList *curr_list, struct gds_library **library_ptr)

  Append library to list.
- static GList * prepend_graphics (GList *curr_list, enum graphics_type type, struct gds_graphics **graphics_ptr)

  Prepend graphics to list.
- static GList * append_vertex (GList *curr_list, int x, int y)

  Appends vertext List.
- static GList * append_cell (GList *curr_list, struct gds_cell **cell_ptr)

  append_cell Append a gds_cell to a list
- static GList * append_cell_ref (GList *curr_list, struct gds_cell_instance **instance_ptr)

  Append a cell reference to the reference GList.
- static int name_library (struct gds_library *current_library, unsigned int bytes, char *data)

  Name a gds_library.
- static int name_cell (struct gds_cell *cell, unsigned int bytes, char *data, struct gds_library *lib)

  Names a gds_cell.
- static void parse_reference_list (gpointer gcell_ref, gpointer glibrary)

  Search for cell reference `gcell_ref` in `glibrary`.
- static void scan_cell_reference_dependencies (gpointer gcell, gpointer library)

  Scans cell references inside cell This function searches all the references in `gcell` and updates the gds_cell_instance::cell_ref field in each instance.
- static void scan_library_references (gpointer library_list_item, gpointer user)

  Scans library's cell references.
- static void gds_parse_date (const char *buffer, int length, struct gds_time_field *mod_date, struct gds_time_field *access_date)

  gds_parse_date
- static void convert_aref_to_sref (struct gds_cell_array_instance *aref, struct gds_cell *container_cell)

  Convert AREF to a bunch of SREFs and append them to `container_cell`.
- int parse_gds_from_file (const char *filename, GList **library_array)

  Parse a GDS file.
- static void delete_cell_inst_element (struct gds_cell_instance *cell_inst)

  delete_cell_inst_element
- static void delete_vertex (struct gds_point *vertex)

  delete_vertex
- static void delete_graphics_obj (struct gds_graphics *gfx)

  delete_graphics_obj
- static void delete_cell_element (struct gds_cell *cell)

  delete_cell_element
- static void delete_library_element (struct gds_library *lib)

  delete_library_element
- int clear_lib_list (GList **library_list)

  Deletes all libraries including cells, references etc.

13.24.1 Detailed Description

Implementation of the GDS-Parser.

Author

Mario Hüttel mario.huettel@gmx.net

What's missing? - A lot: Support for 4 Byte real Support for pathtypes Support for datatypes (only layer so far) etc...

Definition in file gds-parser.c.

13.25 gds-parser.c

Go to the documentation of this file.

```
00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018  Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter.  If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00037 #include <stdlib.h>
00038 #include <stdio.h>
00039 #include <string.h>
00040 #include <stdbool.h>
00041 #include <math.h>
00042 #include <cairo.h>
00043 #include <glib/gi18n.h>
00044
00045 #include <gds-render/gds-utils/gds-parser.h>
00046
00051 #define GDS_DEFAULT_UNITS (10E-9)
00052
00053 #define GDS_ERROR(fmt, ...) printf("[PARSE_ERROR] " fmt "\n", ##__VA_ARGS__)
00054 #define GDS_WARN(fmt, ...) printf("[PARSE_WARNING] " fmt "\n", ##__VA_ARGS__)
00056 #if GDS_PRINT_DEBUG_INFOS
00058         #define GDS_INF(fmt, ...) printf(fmt, ##__VA_ARGS__)
00059 #else
00060         #define GDS_INF(fmt, ...)
00061 #endif
00062 enum gds_record {
00063         INVALID = 0x0000,
00064         HEADER = 0x0002,
00065         BGNLIB = 0x0102,
00066         LIBNAME = 0x0206,
00067         UNITS = 0x0305,
00068         ENDLIB = 0x0400,
00069         BGNSTR = 0x0502,
00070         STRNAME = 0x0606,
00071         ENDSTR = 0x0700,
00072         BOUNDARY = 0x0800,
00073         PATH = 0x0900,
00074         SREF = 0x0A00,
00075         ENDEL = 0x1100,
00076         XY = 0x1003,
00077         MAG = 0x1B05,
00078         ANGLE = 0x1C05,
00079         SNAME = 0x1206,
00080         STRANS = 0x1A01,
00081         BOX = 0x2D00,
00082         LAYER = 0x0D02,
```

```
00083         DATATYPE = 0x0E02,
00084         WIDTH = 0x0F03,
00085         PATHTYPE = 0x2102,
00086         COLROW = 0x1302,
00087         AREF = 0x0B00
00088 };
00089
00096 struct gds_cell_array_instance {
00097         char ref_name[CELL_NAME_MAX];
00098         struct gds_cell *cell_ref;
00099         struct gds_point control_points[3];
00100         int flipped;
00101         double angle;
00102         double magnification;
00103         int columns;
00104         int rows;
00105 };
00106
00114 static int name_cell_ref(struct gds_cell_instance *cell_inst,
00115                          unsigned int bytes, char *data)
00116 {
00117         int len;
00118
00119         if (cell_inst == NULL) {
00120                 GDS_ERROR("Naming cell ref with no opened cell ref");
00121                 return -1;
00122         }
00123         data[bytes] = 0; // Append '0'
00124         len = (int)strlen(data);
00125         if (len > CELL_NAME_MAX-1) {
00126                 GDS_ERROR("Cell name '%s' too long: %d\n", data, len);
00127                 return -1;
00128         }
00129
00130         /* else: */
00131         strcpy(cell_inst->ref_name, data);
00132         GDS_INF("\tCell referenced: %s\n", cell_inst->ref_name);
00133
00134         return 0;
00135 }
00136
00144 static int name_array_cell_ref(struct gds_cell_array_instance *cell_inst,
00145                                unsigned int bytes, char *data)
00146 {
00147         int len;
00148
00149         if (cell_inst == NULL) {
00150                 GDS_ERROR("Naming array cell ref with no opened cell ref");
00151                 return -1;
00152         }
00153         data[bytes] = 0; // Append '0'
00154         len = (int)strlen(data);
00155         if (len > CELL_NAME_MAX-1) {
00156                 GDS_ERROR("Cell name '%s' too long: %d\n", data, len);
00157                 return -1;
00158         }
00159
00160         /* else: */
00161         strcpy(cell_inst->ref_name, data);
00162         GDS_INF("\tCell referenced: %s\n", cell_inst->ref_name);
00163
00164         return 0;
00165 }
00166
00172 static double gds_convert_double(const char *data)
00173 {
00174         bool sign_bit;
00175         int i;
00176         double ret_val;
00177         char current_byte;
00178         int bit = 0;
00179         int exponent;
00180
00181         sign_bit = ((data[0] & 0x80) ? true : false);
00182
00183         /* Check for real 0 */
00184         for (i = 0; i < 8; i++) {
00185                 if (data[i] != 0)
00186                         break;
00187                 if (i == 7) {
00188                         /* All 8 bytes are 0 */
00189                         return 0.0;
00190                 }
00191         }
00192
00193         /* Value is other than 0 */
00194         ret_val = 0.0;
```

```
00195         for (i = 8; i < 64; i++) {
00196                 current_byte = data[i/8];
00197                 bit = i % 8;
00198                 /* isolate bit */
00199                 if ((current_byte & (0x80 » bit)))
00200                         ret_val += pow(2, ((double)(-i+7)));
00201
00202         }
00203
00204         /* Parse exponent and sign bit */
00205         exponent = (int)(data[0] & 0x7F);
00206         exponent -= 64;
00207         ret_val *= pow(16, exponent) * (sign_bit == true ? -1 : 1);
00208
00209         return ret_val;
00210 }
00211
00217 static signed int gds_convert_signed_int(const char *data)
00218 {
00219         int ret;
00220
00221         if (!data) {
00222                 GDS_ERROR("Conversion from GDS data to signed int failed.");
00223                 return 0;
00224         }
00225
00226         ret =   (signed int)(((((int)data[0]) & 0xFF) « 24) |
00227                         ((((int)data[1]) & 0xFF) « 16) |
00228                         (((int)(data[2]) & 0xFF) «  8) |
00229                         (((int)(data[3]) & 0xFF) «  0));
00230         return ret;
00231 }
00232
00238 static int16_t gds_convert_signed_int16(const char *data)
00239 {
00240         if (!data) {
00241                 GDS_ERROR("This should not happen");
00242                 return 0;
00243         }
00244         return (int16_t)((((int16_t)(data[0]) & 0xFF) «  8) |
00245                         (((int16_t)(data[1]) & 0xFF) «  0));
00246 }
00247
00253 static uint16_t gds_convert_unsigned_int16(const char *data)
00254 {
00255         if (!data) {
00256                 GDS_ERROR("This should not happen");
00257                 return 0;
00258         }
00259         return (uint16_t)((((uint16_t)(data[0]) & 0xFF) «  8) |
00260                         (((uint16_t)(data[1]) & 0xFF) «  0));
00261 }
00262
00269 static GList *append_library(GList *curr_list, struct gds_library **library_ptr)
00270 {
00271         struct gds_library *lib;
00272
00273         lib = (struct gds_library *)malloc(sizeof(struct gds_library));
00274         if (lib) {
00275                 lib->cells = NULL;
00276                 lib->name[0] = 0;
00277                 lib->unit_in_meters = GDS_DEFAULT_UNITS; // Default. Will be overwritten
00278                 lib->cell_names = NULL;
00279         } else
00280                 return NULL;
00281         if (library_ptr)
00282                 *library_ptr = lib;
00283
00284         return g_list_append(curr_list, lib);
00285 }
00286
00294 static __attribute__((warn_unused_result)) GList *prepend_graphics(GList *curr_list, enum
      graphics_type type,
00295                                 struct gds_graphics **graphics_ptr)
00296 {
00297         struct gds_graphics *gfx;
00298
00299         gfx = (struct gds_graphics *)malloc(sizeof(struct gds_graphics));
00300         if (gfx) {
00301                 gfx->datatype = 0;
00302                 gfx->layer = 0;
00303                 gfx->vertices = NULL;
00304                 gfx->width_absolute = 0;
00305                 gfx->gfx_type = type;
00306                 gfx->path_render_type = PATH_FLUSH;
00307         } else
00308                 return NULL;
```

```
00309
00310         if (graphics_ptr)
00311                 *graphics_ptr = gfx;
00312
00313         return g_list_prepend(curr_list, gfx);
00314 }
00315
00323 static GList *append_vertex(GList *curr_list, int x, int y)
00324 {
00325         struct gds_point *vertex;
00326
00327         vertex = (struct gds_point *)malloc(sizeof(struct gds_point));
00328         if (vertex) {
00329                 vertex->x = x;
00330                 vertex->y = y;
00331         } else
00332                 return NULL;
00333         return g_list_append(curr_list, vertex);
00334 }
00335
00344 static GList *append_cell(GList *curr_list, struct gds_cell **cell_ptr)
00345 {
00346         struct gds_cell *cell;
00347
00348         cell = (struct gds_cell *)malloc(sizeof(struct gds_cell));
00349         if (cell) {
00350                 cell->child_cells = NULL;
00351                 cell->graphic_objs = NULL;
00352                 cell->name[0] = 0;
00353                 cell->parent_library = NULL;
00354                 cell->checks.unresolved_child_count = GDS_CELL_CHECK_NOT_RUN;
00355                 cell->checks.affected_by_reference_loop = GDS_CELL_CHECK_NOT_RUN;
00356         } else
00357                 return NULL;
00358         /* return cell */
00359         if (cell_ptr)
00360                 *cell_ptr = cell;
00361
00362         return g_list_append(curr_list, cell);
00363 }
00364
00373 static GList *append_cell_ref(GList *curr_list, struct gds_cell_instance **instance_ptr)
00374 {
00375         struct gds_cell_instance *inst;
00376
00377         inst = (struct gds_cell_instance *)
00378                         malloc(sizeof(struct gds_cell_instance));
00379         if (inst) {
00380                 inst->cell_ref = NULL;
00381                 inst->ref_name[0] = 0;
00382                 inst->magnification = 1.0;
00383                 inst->flipped = 0;
00384                 inst->angle = 0.0;
00385         } else
00386                 return NULL;
00387
00388         if (instance_ptr)
00389                 *instance_ptr = inst;
00390
00391         return g_list_append(curr_list, inst);
00392 }
00393
00401 static int name_library(struct gds_library *current_library,
00402                         unsigned int bytes, char *data)
00403 {
00404         int len;
00405
00406         if (current_library == NULL) {
00407                 GDS_ERROR("Naming cell with no opened library");
00408                 return -1;
00409         }
00410
00411         data[bytes] = 0; // Append '0'
00412         len = (int)strlen(data);
00413         if (len > CELL_NAME_MAX-1) {
00414                 GDS_ERROR("Library name '%s' too long: %d\n", data, len);
00415                 return -1;
00416         }
00417
00418         strcpy(current_library->name, data);
00419         GDS_INF("Named library: %s\n", current_library->name);
00420
00421         return 0;
00422 }
00423
00432 static int name_cell(struct gds_cell *cell, unsigned int bytes,
00433                      char *data, struct gds_library *lib)
```

```
00434 {
00435         int len;
00436
00437         if (cell == NULL) {
00438                 GDS_ERROR("Naming library with no opened library");
00439                 return -1;
00440         }
00441         data[bytes] = 0; // Append '0'
00442         len = (int)strlen(data);
00443         if (len > CELL_NAME_MAX-1) {
00444                 GDS_ERROR("Cell name '%s' too long: %d\n", data, len);
00445                 return -1;
00446         }
00447
00448         strcpy(cell->name, data);
00449         GDS_INF("Named cell: %s\n", cell->name);
00450
00451         /* Append cell name to lib's list of names */
00452         lib->cell_names = g_list_append(lib->cell_names, cell->name);
00453
00454         return 0;
00455 }
00456
00464 static void parse_reference_list(gpointer gcell_ref, gpointer glibrary)
00465 {
00466         struct gds_cell_instance *inst = (struct gds_cell_instance *)gcell_ref;
00467         struct gds_library *lib = (struct gds_library *)glibrary;
00468         GList *cell_item;
00469         struct gds_cell *cell;
00470
00471         GDS_INF("\t\t\tReference: %s: ", inst->ref_name);
00472         /* Find cell */
00473         for (cell_item = lib->cells; cell_item != NULL;
00474              cell_item = cell_item->next) {
00475
00476                 cell = (struct gds_cell *)cell_item->data;
00477                 /* Check if cell is found */
00478                 if (!strcmp(cell->name, inst->ref_name)) {
00479                         GDS_INF("found\n");
00480                         /* update reference link */
00481                         inst->cell_ref = cell;
00482                         return;
00483                 }
00484         }
00485
00486         GDS_INF("MISSING!\n");
00487         GDS_WARN("referenced cell could not be found in library");
00488 }
00489
00496 static void scan_cell_reference_dependencies(gpointer gcell, gpointer library)
00497 {
00498         struct gds_cell *cell = (struct gds_cell *)gcell;
00499
00500         GDS_INF("\tScanning cell: %s\n", cell->name);
00501
00502         /* Scan all library references */
00503         g_list_foreach(cell->child_cells, parse_reference_list, library);
00504
00505 }
00506
00514 static void scan_library_references(gpointer library_list_item, gpointer user)
00515 {
00516         struct gds_library *lib = (struct gds_library *)library_list_item;
00517         (void)user;
00518
00519         GDS_INF("Scanning Library: %s\n", lib->name);
00520         g_list_foreach(lib->cells, scan_cell_reference_dependencies, lib);
00521 }
00522
00530 static void gds_parse_date(const char *buffer, int length, struct gds_time_field *mod_date, struct
       gds_time_field *access_date)
00531 {
00532
00533         struct gds_time_field *temp_date;
00534
00535         if (!access_date || !mod_date) {
00536                 GDS_WARN("Date structures invalid");
00537                 return;
00538         }
00539
00540         if (length != (2*6*2)) {
00541                 GDS_WARN("Could not parse date field! Not the specified length");
00542                 return;
00543         }
00544
00545         for (temp_date = mod_date; 1; temp_date = access_date) {
00546                 temp_date->year = gds_convert_unsigned_int16(buffer);
```

```
00547                 buffer += 2;
00548                 temp_date->month = gds_convert_unsigned_int16(buffer);
00549                 buffer += 2;
00550                 temp_date->day = gds_convert_unsigned_int16(buffer);
00551                 buffer += 2;
00552                 temp_date->hour = gds_convert_unsigned_int16(buffer);
00553                 buffer += 2;
00554                 temp_date->minute = gds_convert_unsigned_int16(buffer);
00555                 buffer += 2;
00556                 temp_date->second = gds_convert_unsigned_int16(buffer);
00557                 buffer += 2;
00558 
00559                 if (temp_date == access_date)
00560                         break;
00561         }
00562 }
00563 
00575 static void convert_aref_to_sref(struct gds_cell_array_instance *aref, struct gds_cell
       *container_cell)
00576 {
00577         struct gds_point origin;
00578         struct gds_point row_shift_vector;
00579         struct gds_point col_shift_vector;
00580         struct gds_cell_instance *sref_inst = NULL;
00581         int col;
00582         int row;
00583 
00584         if (!aref || !container_cell)
00585                 return;
00586 
00587         if (aref->columns == 0 || aref->rows == 0) {
00588                 GDS_ERROR("Conversion of array instance aborted. No rows / columns.");
00589                 return;
00590         }
00591         origin.x = aref->control_points[0].x;
00592         origin.y = aref->control_points[0].y;
00593 
00594         row_shift_vector.x = (aref->control_points[2].x - origin.x) / aref->rows;
00595         row_shift_vector.y = (aref->control_points[2].y - origin.y) / aref->rows;
00596         col_shift_vector.x = (aref->control_points[1].x - origin.x) / aref->columns;
00597         col_shift_vector.y = (aref->control_points[1].y - origin.y) / aref->columns;
00598 
00599         /* Iterate over columns and rows */
00600         for (col = 0; col < aref->columns; col++) {
00601                 for (row = 0; row < aref->rows; row++) {
00602                         /* Create new instance for this row/column and configure data */
00603                         container_cell->child_cells = append_cell_ref(container_cell->child_cells,
       &sref_inst);
00604                         if (!sref_inst) {
00605                                 GDS_ERROR("Appending cell ref failed!");
00606                                 continue;
00607                         }
00608 
00609                         sref_inst->angle = aref->angle;
00610                         sref_inst->magnification = aref->magnification;
00611                         sref_inst->flipped = aref->flipped;
00612                         strncpy(sref_inst->ref_name, aref->ref_name, CELL_NAME_MAX);
00613                         sref_inst->origin.x = origin.x + row_shift_vector.x * row + col_shift_vector.x
       * col;
00614                         sref_inst->origin.y = origin.y + row_shift_vector.y * row + col_shift_vector.y
       * col;
00615                 }
00616         }
00617         GDS_INF("Converted AREF to SREFs\n");
00618 }
00619 
00620 int parse_gds_from_file(const char *filename, GList **library_list)
00621 {
00622         char *workbuff;
00623         int read;
00624         int i;
00625         int run = 1;
00626         FILE *gds_file = NULL;
00627         uint16_t rec_data_length;
00628         enum gds_record rec_type;
00629         struct gds_library *current_lib = NULL;
00630         struct gds_cell *current_cell = NULL;
00631         struct gds_graphics *current_graphics = NULL;
00632         struct gds_cell_instance *current_s_reference = NULL;
00633         struct gds_cell_array_instance *current_a_reference = NULL;
00634         struct gds_cell_array_instance temp_a_reference;
00635         int x, y;
00637         GList *lib_list;
00638 
00639         lib_list = *library_list;
00640 
00641         /* Allocate working buffer */
```

```
00642         workbuff = (char *)malloc(sizeof(char)*128*1024);
00643
00644         if(!workbuff)
00645                 return -100;
00646
00647         /* open File */
00648         gds_file = fopen(filename, "rb");
00649         if (gds_file == NULL) {
00650                 GDS_ERROR("Could not open File %s", filename);
00651                 return -1;
00652         }
00653
00654         /* Record parser */
00655         while (run == 1) {
00656                 rec_type = INVALID;
00657                 read = fread(workbuff, sizeof(char), 2, gds_file);
00658                 if (read != 2 && (current_cell != NULL ||
00659                                   current_graphics != NULL ||
00660                                   current_lib != NULL ||
00661                                   current_s_reference != NULL)) {
00662                         GDS_ERROR("End of File. with openend structs/libs");
00663                         run = -2;
00664                         break;
00665                 } else if (read != 2) {
00666                         /* EOF */
00667                         run = 0;
00668                         break;
00669                 }
00670
00671                 rec_data_length = gds_convert_unsigned_int16(workbuff);
00672
00673                 if (rec_data_length < 4) {
00674                         /* Possible Zero-Padding: */
00675                         run = 0;
00676                         GDS_WARN("Zero Padding detected!");
00677                         if (current_cell != NULL ||
00678                                         current_graphics != NULL ||
00679                                         current_lib != NULL ||
00680                                         current_s_reference != NULL) {
00681                                 GDS_ERROR("Not all structures closed");
00682                                 run = -2;
00683                         }
00684                         break;
00685                 }
00686                 rec_data_length -= 4;
00687
00688                 read = fread(workbuff, sizeof(char), 2, gds_file);
00689                 if (read != 2) {
00690                         run = -2;
00691                         GDS_ERROR("Unexpected end of file");
00692                         break;
00693                 }
00694                 rec_type = gds_convert_unsigned_int16(workbuff);
00695
00696
00697                 /* if begin: Allocate structures */
00698                 switch (rec_type) {
00699                 case BGNLIB:
00700                         lib_list = append_library(lib_list, &current_lib);
00701                         if (lib_list == NULL) {
00702                                 GDS_ERROR("Allocating memory failed");
00703                                 run = -3;
00704                                 break;
00705
00706                         }
00707                         GDS_INF("Entering Lib\n");
00708                         break;
00709                 case ENDLIB:
00710                         if (current_lib == NULL) {
00711                                 run = -4;
00712                                 GDS_ERROR("Closing Library with no opened library");
00713                                 break;
00714                         }
00715
00716                         /* Check for open Cells */
00717                         if (current_cell != NULL) {
00718                                 run = -4;
00719                                 GDS_ERROR("Closing Library with opened cells");
00720                                 break;
00721                         }
00722                         current_lib = NULL;
00723                         GDS_INF("Leaving Library\n");
00724                         break;
00725                 case BGNSTR:
00726                         if (current_lib == NULL) {
00727                                 GDS_ERROR("Defining Cell outside of library!\n");
00728                                 run = -4;
```

```
00729                                   break;
00730                           }
00731                           current_lib->cells = append_cell(current_lib->cells, &current_cell);
00732                           if (current_lib->cells == NULL) {
00733                                   GDS_ERROR("Allocating memory failed");
00734                                   run = -3;
00735                                   break;
00736                           }
00737
00738                           current_cell->parent_library = current_lib;
00739
00740                           GDS_INF("Entering cell\n");
00741                           break;
00742                   case ENDSTR:
00743                           if (current_cell == NULL) {
00744                                   run = -4;
00745                                   GDS_ERROR("Closing cell with no opened cell");
00746                                   break;
00747                           }
00748                           /* Check for open Elements */
00749                           if (current_graphics != NULL || current_s_reference != NULL) {
00750                                   run = -4;
00751                                   GDS_ERROR("Closing cell with opened Elements");
00752                                   break;
00753                           }
00754                           current_cell = NULL;
00755                           GDS_INF("Leaving Cell\n");
00756                           break;
00757                   case BOX:
00758                   case BOUNDARY:
00759                           if (current_cell == NULL) {
00760                                   GDS_ERROR("Boundary/Box outside of cell");
00761                                   run = -3;
00762                                   break;
00763                           }
00764                           current_cell->graphic_objs = prepend_graphics(current_cell->graphic_objs,
00765                                                                         (rec_type == BOUNDARY
00766                                                                            ? GRAPHIC_POLYGON
00767                                                                            : GRAPHIC_BOX),
00768                                                                         &current_graphics);
00769                           if (current_cell->graphic_objs == NULL) {
00770                                   GDS_ERROR("Memory allocation failed");
00771                                   run = -4;
00772                                   break;
00773                           }
00774                           GDS_INF("\tEntering boundary/Box\n");
00775                           break;
00776                   case SREF:
00777                           if (current_cell == NULL) {
00778                                   GDS_ERROR("Cell Reference outside of cell");
00779                                   run = -3;
00780                                   break;
00781                           }
00782                           current_cell->child_cells = append_cell_ref(current_cell->child_cells,
00783                                                                       &current_s_reference);
00784                           if (current_cell->child_cells == NULL) {
00785                                   GDS_ERROR("Memory allocation failed");
00786                                   run = -4;
00787                                   break;
00788                           }
00789
00790                           GDS_INF("\tEntering reference\n");
00791                           break;
00792                   case PATH:
00793                           if (current_cell == NULL) {
00794                                   GDS_ERROR("Path outside of cell");
00795                                   run = -3;
00796                                   break;
00797                           }
00798                           current_cell->graphic_objs = prepend_graphics(current_cell->graphic_objs,
00799                                                                         GRAPHIC_PATH, &current_graphics);
00800                           if (current_cell->graphic_objs == NULL) {
00801                                   GDS_ERROR("Memory allocation failed");
00802                                   run = -4;
00803                                   break;
00804                           }
00805                           GDS_INF("\tEntering Path\n");
00806                           break;
00807                   case ENDEL:
00808                           if (current_graphics != NULL) {
00809                                   GDS_INF("\tLeaving %s\n", (current_graphics->gfx_type ==
    GRAPHIC_POLYGON ? "boundary" : "path"));
00810                                   current_graphics = NULL;
00811                           }
00812                           if (current_s_reference != NULL) {
00813                                   GDS_INF("\tLeaving Reference\n");
00814                                   current_s_reference = NULL;
```

```
00815                         }
00816                         if (current_a_reference != NULL) {
00817                                 GDS_INF("\tLeaving Array Reference\n");
00818                                 convert_aref_to_sref(current_a_reference, current_cell);
00819                                 current_a_reference = NULL;
00820                         }
00821
00822                         break;
00823                 case XY:
00824                         if (current_graphics) {
00825
00826                         } else if (current_s_reference) {
00827                                 if (rec_data_length != 8) {
00828                                         GDS_WARN("Instance has weird coordinates. Rendered output
      might be screwed!");
00829                                 }
00830                         } else if (current_a_reference) {
00831                                 if (rec_data_length != (3*(4+4)))
00832                                         GDS_WARN("Array instance has weird coordinates. Rendered
      output might be screwed!");
00833                         }
00834                         break;
00835                 case AREF:
00836                         if (current_cell == NULL) {
00837                                 GDS_ERROR("Cell array reference outside of cell");
00838                                 run = -3;
00839                                 break;
00840                         }
00841
00842                         if (current_a_reference != NULL) {
00843                                 GDS_ERROR("Recursive cell array reference");
00844                                 run = -3;
00845                                 break;
00846                         }
00847
00848                         GDS_INF("Entering Array Reference\n");
00849
00850                         /* Array references are coverted after fully declared. Therefore,
00851                          * only a static buffer is needed
00852                          */
00853                         current_a_reference = &temp_a_reference;
00854                         current_a_reference->ref_name[0] = '\0';
00855                         current_a_reference->angle = 0.0;
00856                         current_a_reference->magnification = 1.0;
00857                         current_a_reference->flipped = 0;
00858                         current_a_reference->rows = 0;
00859                         current_a_reference->columns = 0;
00860                         break;
00861                 case COLROW:
00862                 case MAG:
00863                 case ANGLE:
00864                 case STRANS:
00865                 case WIDTH:
00866                 case PATHTYPE:
00867                 case UNITS:
00868                 case LIBNAME:
00869                 case SNAME:
00870                 case LAYER:
00871                 case DATATYPE:
00872                 case STRNAME:
00873                         break;
00874                 default:
00875                         GDS_INF("Unhandled Record: %04x, len: %u\n", (unsigned int)rec_type, (unsigned
      int)rec_data_length);
00876                         break;
00877                 } /* switch(rec_type) */
00878
00879
00880                 /* No Data -> No Processing, go back to top */
00881                 if (!rec_data_length || run != 1) continue;
00882
00883                 read = fread(workbuff, sizeof(char), rec_data_length, gds_file);
00884
00885                 if (read != rec_data_length) {
00886                         GDS_ERROR("Could not read enough data: requested: %u, read: %u | Type:
      0x%04x\n",
00887                                         (unsigned int)rec_data_length, (unsigned int)read, (unsigned
      int)rec_type);
00888                         run = -5;
00889                         break;
00890                 }
00891
00892                 switch (rec_type) {
00893                 case AREF:
00894                 case HEADER:
00895                 case ENDLIB:
00896                 case ENDSTR:
```

```
00897                 case BOUNDARY:
00898                 case PATH:
00899                 case SREF:
00900                 case ENDEL:
00901                 case BOX:
00902                 case INVALID:
00903                         break;
00904
00905                 case COLROW:
00906                         if (!current_a_reference) {
00907                                 GDS_ERROR("COLROW record defined outside of array instance");
00908                                 break;
00909                         }
00910                         if (rec_data_length != 4 || read != 4) {
00911                                 GDS_ERROR("COLUMN/ROW count record contains too few data. Won't set
      column and row counts (%d, %d)",
00912                                                 rec_data_length, read);
00913                                 break;
00914                         }
00915                         current_a_reference->columns = (int)gds_convert_signed_int16(&workbuff[0]);
00916                         current_a_reference->rows = (int)gds_convert_signed_int16(&workbuff[2]);
00917                         GDS_INF("\tRows: %d\n\tColumns: %d\n", current_a_reference->rows,
      current_a_reference->columns);
00918                         break;
00919                 case UNITS:
00920                         if (!current_lib) {
00921                                 GDS_WARN("Units defined outside of library!\n");
00922                                 break;
00923                         }
00924
00925                         if (rec_data_length != 16) {
00926                                 GDS_WARN("Unit define incomplete. Will assume database unit of %E
      meters\n", current_lib->unit_in_meters);
00927                                 break;
00928                         }
00929
00930                         current_lib->unit_in_meters = gds_convert_double(&workbuff[8]);
00931                         GDS_INF("Length of database unit: %E meters\n", current_lib->unit_in_meters);
00932                         break;
00933                 case BGNLIB:
00934                         /* Parse date record */
00935                         gds_parse_date(workbuff, read, &current_lib->mod_time,
      &current_lib->access_time);
00936                         break;
00937                 case BGNSTR:
00938                         gds_parse_date(workbuff, read, &current_cell->mod_time,
      &current_cell->access_time);
00939                         break;
00940                 case LIBNAME:
00941                         name_library(current_lib, (unsigned int)read, workbuff);
00942                         break;
00943                 case STRNAME:
00944                         name_cell(current_cell, (unsigned int)read, workbuff, current_lib);
00945                         break;
00946                 case XY:
00947                         if (current_s_reference) {
00948                                 /* Get origin of reference */
00949                                 current_s_reference->origin.x = gds_convert_signed_int(workbuff);
00950                                 current_s_reference->origin.y = gds_convert_signed_int(&workbuff[4]);
00951                                 GDS_INF("\t\tSet origin to: %d/%d\n", current_s_reference->origin.x,
00952                                                 current_s_reference->origin.y);
00953                         } else if (current_graphics) {
00954                                 for (i = 0; i < read/8; i++) {
00955                                         x = gds_convert_signed_int(&workbuff[i*8]);
00956                                         y = gds_convert_signed_int(&workbuff[i*8+4]);
00957                                         current_graphics->vertices =
00958                                                         append_vertex(current_graphics->vertices, x,
      y);
00959                                         GDS_INF("\t\tSet coordinate: %d/%d\n", x, y);
00960
00961                                 }
00962                         } else if (current_a_reference) {
00963                                 for (i = 0; i < 3; i++) {
00964                                         x = gds_convert_signed_int(&workbuff[i*8]);
00965                                         y = gds_convert_signed_int(&workbuff[i*8+4]);
00966                                         current_a_reference->control_points[i].x = x;
00967                                         current_a_reference->control_points[i].y = y;
00968                                         GDS_INF("\tSet control point %d: %d/%d\n", i, x, y);
00969                                 }
00970                         }
00971                         break;
00972                 case STRANS:
00973                         if (current_s_reference) {
00974                                 current_s_reference->flipped = ((workbuff[0] & 0x80) ? 1 : 0);
00975                         } else if (current_a_reference) {
00976                                 current_a_reference->flipped = ((workbuff[0] & 0x80) ? 1 : 0);
00977                         } else {
```

```
00978                                   GDS_ERROR("Transformation defined outside of instance");
00979                                   break;
00980                           }
00981                           break;
00982                   case SNAME:
00983                           if (current_s_reference) {
00984                                   name_cell_ref(current_s_reference, (unsigned int)read, workbuff);
00985                           } else if (current_a_reference) {
00986                                   name_array_cell_ref(current_a_reference, (unsigned int)read,
      workbuff);
00987                           } else {
00988                                   GDS_ERROR("Reference name set outside of cell reference");
00989                           }
00990                           break;
00991                   case WIDTH:
00992                           if (!current_graphics) {
00993                                   GDS_WARN("Width defined outside of path element");
00994                           }
00995                           current_graphics->width_absolute = gds_convert_signed_int(workbuff);
00996                           break;
00997                   case LAYER:
00998                           if (!current_graphics) {
00999                                   GDS_WARN("Layer has to be defined inside graphics object. Probably
      unknown object. Implement it yourself!");
01000                                   break;
01001                           }
01002                           current_graphics->layer = gds_convert_signed_int16(workbuff);
01003                           if (current_graphics->layer < 0) {
01004                                   GDS_WARN("Layer negative!\n");
01005                           }
01006                           GDS_INF("\t\tAdded layer %d\n", (int)current_graphics->layer);
01007                           break;
01008                   case DATATYPE:
01009                           if (!current_graphics) {
01010                                   GDS_WARN("Datatype has to be defined inside graphics object. Probably
      unknown object. Implement it yourself!");
01011                                   break;
01012                           }
01013                           current_graphics->datatype = gds_convert_signed_int16(workbuff);
01014                           if (current_graphics->datatype < 0)
01015                                   GDS_WARN("Datatype negative!");
01016                           GDS_INF("\t\tAdded datatype %d\n", (int)current_graphics->datatype);
01017                           break;
01018                   case MAG:
01019                           if (rec_data_length != 8) {
01020                                   GDS_WARN("Magnification is not an 8 byte real. Results may be wrong");
01021                           }
01022                           if (current_graphics != NULL && current_s_reference != NULL) {
01023                                   GDS_ERROR("Open Graphics and Cell Reference\n\tMissing ENDEL?");
01024                                   run = -6;
01025                                   break;
01026                           }
01027                           if (current_s_reference != NULL) {
01028                                   current_s_reference->magnification = gds_convert_double(workbuff);
01029                                   GDS_INF("\t\tMagnification defined: %lf\n",
      current_s_reference->magnification);
01030                           }
01031                           if (current_a_reference != NULL) {
01032                                   current_a_reference->magnification = gds_convert_double(workbuff);
01033                                   GDS_INF("\t\tMagnification defined: %lf\n",
      current_a_reference->magnification);
01034                           }
01035                           break;
01036                   case ANGLE:
01037                           if (rec_data_length != 8) {
01038                                   GDS_WARN("Angle is not an 8 byte real. Results may be wrong");
01039                           }
01040                           if (current_graphics != NULL && current_s_reference != NULL &&
      current_a_reference != NULL) {
01041                                   GDS_ERROR("Open Graphics and Cell Reference\n\tMissing ENDEL?");
01042                                   run = -6;
01043                                   break;
01044                           }
01045                           if (current_s_reference != NULL) {
01046                                   current_s_reference->angle = gds_convert_double(workbuff);
01047                                   GDS_INF("\t\tAngle defined: %lf\n", current_s_reference->angle);
01048                           }
01049                           if (current_a_reference != NULL) {
01050                                   current_a_reference->angle = gds_convert_double(workbuff);
01051                                   GDS_INF("\t\tAngle defined: %lf\n", current_a_reference->angle);
01052                           }
01053                           break;
01054                   case PATHTYPE:
01055                           if (current_graphics == NULL) {
01056                                   GDS_WARN("Path type defined outside of path. Ignoring");
01057                                   break;
01058                           }
```

```
01059                         if (current_graphics->gfx_type == GRAPHIC_PATH) {
01060                                 current_graphics->path_render_type = (enum
      path_type)gds_convert_signed_int16(workbuff);
01061                                 GDS_INF("\t\tPathtype: %d\n", current_graphics->path_render_type);
01062                         } else {
01063                                 GDS_WARN("Path type defined inside non-path graphics object.
      Ignoring");
01064                         }
01065                         break;
01066
01067                 }
01068
01069         } /* while(run == 1) */
01070
01071         fclose(gds_file);
01072
01073         if (!run) {
01074                 /* Iterate and find references to cells */
01075                 g_list_foreach(lib_list, scan_library_references, NULL);
01076         }
01077
01078         *library_list = lib_list;
01079
01080         free(workbuff);
01081
01082         return run;
01083 }
01084
01089 static void delete_cell_inst_element(struct gds_cell_instance *cell_inst)
01090 {
01091         if (cell_inst)
01092                 free(cell_inst);
01093 }
01094
01099 static void delete_vertex(struct gds_point *vertex)
01100 {
01101         if (vertex)
01102                 free(vertex);
01103 }
01104
01109 static void delete_graphics_obj(struct gds_graphics *gfx)
01110 {
01111         if (!gfx)
01112                 return;
01113
01114         g_list_free_full(gfx->vertices, (GDestroyNotify)delete_vertex);
01115         free(gfx);
01116 }
01117
01122 static void delete_cell_element(struct gds_cell *cell)
01123 {
01124         if (!cell)
01125                 return;
01126
01127         g_list_free_full(cell->child_cells, (GDestroyNotify)delete_cell_inst_element);
01128         g_list_free_full(cell->graphic_objs, (GDestroyNotify)delete_graphics_obj);
01129         free(cell);
01130 }
01131
01136 static void delete_library_element(struct gds_library *lib)
01137 {
01138         if (!lib)
01139                 return;
01140
01141         g_list_free(lib->cell_names);
01142         g_list_free_full(lib->cells, (GDestroyNotify)delete_cell_element);
01143         free(lib);
01144 }
01145
01146 int clear_lib_list(GList **library_list)
01147 {
01148         if (!library_list)
01149                 return 0;
01150
01151         if (*library_list == NULL)
01152                 return 0;
01153
01154         g_list_free_full(*library_list, (GDestroyNotify)delete_library_element);
01155         *library_list = NULL;
01156         return 0;
01157 }
01158
```

13.26 gds-tree-checker.c File Reference

Checking functions of a cell tree.

```
#include <stdio.h>
#include <glib/gi18n.h>
#include <gds-render/gds-utils/gds-tree-checker.h>
```

Include dependency graph for gds-tree-checker.c:

Functions

- int gds_tree_check_cell_references (struct gds_library *lib)

  gds_tree_check_cell_references checks if all child cell references can be resolved in the given library
- static int gds_tree_check_list_contains_cell (GList *list, struct gds_cell *cell)

  Check if list contains a cell.
- static int gds_tree_check_iterate_ref_and_check (struct gds_cell *cell_to_check, GList **visited_cells)

  This function follows down the reference list of a cell and marks each visited subcell and detects loops.
- int gds_tree_check_reference_loops (struct gds_library *lib)

  gds_tree_check_reference_loops checks if the given library contains reference loops

13.26.1 Detailed Description

Checking functions of a cell tree.

This file contains checking functions for the GDS cell tree. These functions include checks if all child references could be resolved, and if the cell tree contains loops.

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file gds-tree-checker.c.

13.27 gds-tree-checker.c

Go to the documentation of this file.

```
00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2019  Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter.  If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00036 #include <stdio.h>
00037 #include <glib/gi18n.h>
```

```
00038 #include <gds-render/gds-utils/gds-tree-checker.h>
00039
00040 int gds_tree_check_cell_references(struct gds_library *lib)
00041 {
00042         GList *cell_iter;
00043         struct gds_cell *cell;
00044         GList *instance_iter;
00045         struct  gds_cell_instance *cell_inst;
00046         int total_unresolved_count = 0;
00047
00048         if (!lib)
00049                 return -1;
00050
00051         /* Iterate over all cells in library */
00052         for (cell_iter = lib->cells; cell_iter != NULL; cell_iter = g_list_next(cell_iter)) {
00053                 cell = (struct gds_cell *)cell_iter->data;
00054
00055                 /* Check if this list element is broken. This should never happen */
00056                 if (!cell) {
00057                         fprintf(stderr, _("Broken cell list item found. Will continue.\n"));
00058                         continue;
00059                 }
00060
00061                 /* Reset the unresolved cell reference counter to 0 */
00062                 cell->checks.unresolved_child_count = 0;
00063
00064                 /* Iterate through all child cell references and check if the references are set */
00065                 for (instance_iter = cell->child_cells; instance_iter != NULL;
00066                                         instance_iter = g_list_next(instance_iter)) {
00067                         cell_inst = (struct gds_cell_instance *)instance_iter->data;
00068
00069                         /* Check if broken. This should not happen */
00070                         if (!cell_inst) {
00071                                 fprintf(stderr, _("Broken cell list item found in cell %s. Will
      continue.\n"),
00072                                                         cell->name);
00073                                 continue;
00074                         }
00075
00076                         /* Check if instance is valid; else increment "error" counter of cell */
00077                         if (!cell_inst->cell_ref) {
00078                                 total_unresolved_count++;
00079                                 cell->checks.unresolved_child_count++;
00080                         }
00081                 }
00082         }
00083
00084         return total_unresolved_count;
00085 }
00086
00093 static int gds_tree_check_list_contains_cell(GList *list, struct gds_cell *cell)
00094 {
00095         GList *iter;
00096
00097         for (iter = list; iter != NULL; iter = g_list_next(iter)) {
00098                 if ((struct gds_cell *)iter->data == cell)
00099                         return 1;
00100         }
00101
00102         return 0;
00103 }
00104
00111 static int gds_tree_check_iterate_ref_and_check(struct gds_cell *cell_to_check, GList **visited_cells)
00112 {
00113         GList *ref_iter;
00114         struct gds_cell_instance *ref;
00115         struct gds_cell *sub_cell;
00116         int res;
00117
00118         if (!cell_to_check)
00119                 return -1;
00120
00121         /* Check if this cell is already contained in visited cells. This indicates a loop */
00122         if (gds_tree_check_list_contains_cell(*visited_cells, cell_to_check))
00123                 return 1;
00124
00125         /* Add cell to visited cell list */
00126         *visited_cells = g_list_append(*visited_cells, (gpointer)cell_to_check);
00127
00128         /* Mark references and process sub cells */
00129         for (ref_iter = cell_to_check->child_cells; ref_iter != NULL; ref_iter =
      g_list_next(ref_iter)) {
00130                 ref = (struct gds_cell_instance *)ref_iter->data;
00131
00132                 if (!ref)
00133                         return -1;
00134
```

```
00135                 sub_cell = ref->cell_ref;
00136
00137                 /* If cell is not resolved, ignore. No harm there */
00138                 if (!sub_cell)
00139                         continue;
00140
00141                 res = gds_tree_check_iterate_ref_and_check(sub_cell, visited_cells);
00142                 if (res < 0) {
00143                         /* Error. return. */
00144                         return -3;
00145                 } else if (res > 0) {
00146                         /* Loop in subcell found. Propagate to top */
00147                         return 1;
00148                 }
00149         }
00150
00151         /* Remove cell from visted cells */
00152         *visited_cells = g_list_remove(*visited_cells, cell_to_check);
00153
00154         /* No error found in this chain */
00155         return 0;
00156 }
00157
00158 int gds_tree_check_reference_loops(struct gds_library *lib)
00159 {
00160         int res;
00161         int loop_count = 0;
00162         GList *cell_iter;
00163         struct gds_cell *cell_to_check;
00164         GList *visited_cells = NULL;
00165
00166
00167         if (!lib)
00168                 return -1;
00169
00170         for (cell_iter = lib->cells; cell_iter != NULL; cell_iter = g_list_next(cell_iter)) {
00171                 cell_to_check = (struct gds_cell *)cell_iter->data;
00172
00173                 /* A broken cell reference will be counted fatal in this case */
00174                 if (!cell_to_check)
00175                         return -2;
00176
00177                 /* iterate through references and check if loop exists */
00178                 res = gds_tree_check_iterate_ref_and_check(cell_to_check, &visited_cells);
00179
00180                 if (visited_cells) {
00181                         /*
00182                          * If  cell contains no loop, print error when list not empty.
00183                          * In case of a loop, it is completely normal that the list is not empty,
00184                          * due to the instant return from gds_tree_check_iterate_ref_and_check()
00185                          */
00186                         if (res == 0)
00187                                 fprintf(stderr,
00188                                         _("Visited cell list should be empty. This is a bug. Please
    report this.\n"));
00189                         g_list_free(visited_cells);
00190                         visited_cells = NULL;
00191                 }
00192
00193                 if (res < 0) {
00194                         /* Error */
00195                         return res;
00196                 } else if (res > 0) {
00197                         /* Loop found: increment loop count and flag cell */
00198                         cell_to_check->checks.affected_by_reference_loop = 1;
00199                         loop_count++;
00200                 } else if (res == 0) {
00201                         /* No error found for this cell */
00202                         cell_to_check->checks.affected_by_reference_loop = 0;
00203                 }
00204
00205         }
00206
00207
00208         return loop_count;
00209 }
00210
```

13.28 bounding-box.c File Reference

Calculation of bounding boxes.

```
#include <stdio.h>
#include <math.h>
#include <gds-render/geometric/bounding-box.h>
```
Include dependency graph for bounding-box.c:

Macros

- #define MIN(a, b) (((a) < (b)) ? (a) : (b))

  Return smaller number.
- #define MAX(a, b) (((a) > (b)) ? (a) : (b))

  Return bigger number.
- #define ABS_DBL(a) ((a) < 0 ? -(a) : (a))

Functions

- void bounding_box_calculate_from_polygon (GList *vertices, conv_generic_to_vector_2d_t conv_func, union bounding_box *box)

  Calculate bounding box of polygon.
- void bounding_box_update_with_box (union bounding_box *destination, union bounding_box *update)

  Update an exisitng bounding box with another one.
- void bounding_box_prepare_empty (union bounding_box *box)

  Prepare an empty bounding box.
- static void calculate_path_miter_points (struct vector_2d *a, struct vector_2d *b, struct vector_2d *c, struct vector_2d *m1, struct vector_2d *m2, double width)

  Calculate path miter points for a pathwith a `width` and the anchors `a b c`.
- void bounding_box_update_with_path (GList *vertices, double thickness, conv_generic_to_vector_2d_t conv_func, union bounding_box *box)

  Calculate the bounding box of a path and update the given bounding box.
- void bounding_box_update_with_point (union bounding_box *destination, conv_generic_to_vector_2d_t conv_func, void *pt)

  Update bounding box with a point.
- void bounding_box_get_all_points (struct vector_2d *points, union bounding_box *box)

  Return all four corner points of a bounding box.
- void bounding_box_apply_transform (double scale, double rotation_deg, bool flip_at_x, union bounding_box *box)

  Apply transformations onto bounding box.

13.28.1 Detailed Description

Calculation of bounding boxes.

Author

Mario Hüttel `mario.huettel@gmx.net`

Definition in file bounding-box.c.

13.29 bounding-box.c

Go to the documentation of this file.

```
00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018  Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter.  If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00031 #include <stdio.h>
00032 #include <math.h>
00033
00034 #include <gds-render/geometric/bounding-box.h>
00035
00036 #define MIN(a, b) (((a) < (b)) ? (a) : (b))
00037 #define MAX(a, b) (((a) > (b)) ? (a) : (b))
00038 #define ABS_DBL(a) ((a) < 0 ? -(a) : (a))
00039
00040 void bounding_box_calculate_from_polygon(GList *vertices, conv_generic_to_vector_2d_t conv_func, union
       bounding_box *box)
00041 {
00042         double xmin = DBL_MAX, xmax = -DBL_MAX, ymin = DBL_MAX, ymax = -DBL_MAX;
00043         struct vector_2d temp_vec;
00044         GList *list_item;
00045
00046         /* Check for errors */
00047         if (!conv_func || !box || !vertices)
00048                 return;
00049
00050         for (list_item = vertices; list_item != NULL; list_item = g_list_next(list_item)) {
00051                 /* Convert generic vertex to vector_2d */
00052                 if (conv_func)
00053                         conv_func((void *)list_item->data, &temp_vec);
00054                 else
00055                         vector_2d_copy(&temp_vec, (struct vector_2d *)list_item->data);
00056
00057                 /* Update bounding coordinates with vertex */
00058                 xmin = MIN(xmin, temp_vec.x);
00059                 xmax = MAX(xmax, temp_vec.x);
00060                 ymin = MIN(ymin, temp_vec.y);
00061                 ymax = MAX(ymax, temp_vec.y);
00062         }
00063
00064         /* Fill bounding box with results */
00065         box->vectors.lower_left.x = xmin;
00066         box->vectors.lower_left.y = ymin;
00067         box->vectors.upper_right.x = xmax;
00068         box->vectors.upper_right.y = ymax;
00069 }
00070
00071 void bounding_box_update_with_box(union bounding_box *destination, union bounding_box *update)
00072 {
00073         if (!destination || !update)
00074                 return;
00075
00076         destination->vectors.lower_left.x = MIN(destination->vectors.lower_left.x,
00077                                                 update->vectors.lower_left.x);
00078         destination->vectors.lower_left.y = MIN(destination->vectors.lower_left.y,
00079                                                 update->vectors.lower_left.y);
00080         destination->vectors.upper_right.x = MAX(destination->vectors.upper_right.x,
00081                                                 update->vectors.upper_right.x);
00082         destination->vectors.upper_right.y = MAX(destination->vectors.upper_right.y,
00083                                                 update->vectors.upper_right.y);
00084 }
00085
00086 void bounding_box_prepare_empty(union bounding_box *box)
00087 {
00088         box->vectors.lower_left.x = DBL_MAX;
00089         box->vectors.lower_left.y = DBL_MAX;
00090         box->vectors.upper_right.x = -DBL_MAX;
00091         box->vectors.upper_right.y = -DBL_MAX;
00092 }
```

```
00093
00105 static void calculate_path_miter_points(struct vector_2d *a, struct vector_2d *b, struct vector_2d *c,
00106                                         struct vector_2d *m1, struct vector_2d *m2, double width)
00107 {
00108         double angle, angle_sin, u;
00109         struct vector_2d ba, bc, u_vec, v_vec, ba_norm;
00110
00111         if (!a || !b || !c || !m1 || !m2)
00112                 return;
00113
00114         vector_2d_subtract(&ba, a, b);
00115         vector_2d_subtract(&bc, c, b);
00116
00117         angle = vector_2d_calculate_angle_between(&ba, &bc);
00118
00119         if (ABS_DBL(angle) < 0.05 || ABS_DBL(angle - M_PI) < 0.1) {
00120                 /* Specail cases Don*/
00121                 vector_2d_copy(&ba_norm, &ba);
00122                 vector_2d_rotate(&ba_norm, DEG2RAD(90));
00123                 vector_2d_normalize(&ba_norm);
00124                 vector_2d_scale(&ba_norm, width/2.0);
00125                 vector_2d_add(m1, b, &ba_norm);
00126                 vector_2d_subtract(m2, b, &ba_norm);
00127                 return;
00128         }
00129         angle_sin = sin(angle);
00130         u = width/(2*angle_sin);
00131
00132         vector_2d_copy(&u_vec, &ba);
00133         vector_2d_copy(&v_vec, &bc);
00134         vector_2d_normalize(&u_vec);
00135         vector_2d_normalize(&v_vec);
00136         vector_2d_scale(&u_vec, u);
00137         vector_2d_scale(&v_vec, u);
00138
00139         vector_2d_copy(m1, b);
00140         vector_2d_add(m1, m1, &u_vec);
00141         vector_2d_add(m1, m1, &v_vec);
00142
00143         vector_2d_copy(m2, b);
00144         vector_2d_subtract(m2, m2, &u_vec);
00145         vector_2d_subtract(m2, m2, &v_vec);
00146 }
00147
00148 void bounding_box_update_with_path(GList *vertices, double thickness,
00149                                         conv_generic_to_vector_2d_t conv_func, union bounding_box
      *box)
00150 {
00151         GList *vertex_iterator;
00152         struct vector_2d pt;
00153
00154         if (!vertices || !box)
00155                 return;
00156
00157         for (vertex_iterator = vertices; vertex_iterator != NULL; vertex_iterator =
      g_list_next(vertex_iterator)) {
00158
00159                 if (conv_func != NULL)
00160                         conv_func(vertex_iterator->data, &pt);
00161                 else
00162                         (void)vector_2d_copy(&pt, (struct vector_2d *)vertex_iterator->data);
00163
00164                 /* These are approximations.
00165                  * Used as long as miter point calculation is not fully implemented
00166                  */
00167                 box->vectors.lower_left.x = MIN(box->vectors.lower_left.x, pt.x - thickness/2);
00168                 box->vectors.lower_left.y = MIN(box->vectors.lower_left.y, pt.y - thickness/2);
00169                 box->vectors.upper_right.x = MAX(box->vectors.upper_right.x, pt.x + thickness/2);
00170                 box->vectors.upper_right.y = MAX(box->vectors.upper_right.y, pt.y + thickness/2);
00171         }
00172 }
00173
00174 void bounding_box_update_with_point(union bounding_box *destination, conv_generic_to_vector_2d_t
      conv_func, void *pt)
00175 {
00176         struct vector_2d point;
00177
00178         if (!destination || !pt)
00179                 return;
00180
00181         if (conv_func)
00182                 conv_func(pt, &point);
00183         else
00184                 (void)vector_2d_copy(&point, (struct vector_2d *)pt);
00185
00186         destination->vectors.lower_left.x = MIN(destination->vectors.lower_left.x, point.x);
00187         destination->vectors.lower_left.y = MIN(destination->vectors.lower_left.y, point.y);
```

```
00188         destination->vectors.upper_right.x = MAX(destination->vectors.upper_right.x, point.x);
00189         destination->vectors.upper_right.y = MAX(destination->vectors.upper_right.y, point.y);
00190 }
00191
00192 void bounding_box_get_all_points(struct vector_2d *points, union bounding_box *box)
00193 {
00194         if (!points || !box)
00195                 return;
00196
00197         points[0].x = box->vectors.lower_left.x;
00198         points[0].y = box->vectors.lower_left.y;
00199         points[1].x = box->vectors.upper_right.x;
00200         points[1].y = box->vectors.lower_left.y;
00201         points[2].x = box->vectors.upper_right.x;
00202         points[2].y = box->vectors.upper_right.y;
00203         points[3].x = box->vectors.lower_left.x;
00204         points[3].y = box->vectors.upper_right.y;
00205 }
00206
00207 void bounding_box_apply_transform(double scale, double rotation_deg, bool flip_at_x, union
      bounding_box *box)
00208 {
00209         int i;
00210         struct vector_2d input_points[4];
00211
00212         if (!box)
00213                 return;
00214
00215         bounding_box_get_all_points(input_points, box);
00216
00217         /* Reset box */
00218         bounding_box_prepare_empty(box);
00219
00220         for (i = 0; i < 4; i++) {
00221                 input_points[i].y *= (flip_at_x ? -1 : 1);
00222                 vector_2d_rotate(&input_points[i], rotation_deg * M_PI / 180.0);
00223                 vector_2d_scale(&input_points[i], scale);
00224
00225                 bounding_box_update_with_point(box, NULL, &input_points[i]);
00226         }
00227 }
00228
```

13.30 cell-geometrics.c File Reference

Calculation of gds_cell trigonometrics.

```
#include <math.h>
#include <gds-render/geometric/cell-geometrics.h>
```

Include dependency graph for cell-geometrics.c:

Functions

- static void convert_gds_point_to_2d_vector (struct gds_point *pt, struct vector_2d *vector)
- static void update_box_with_gfx (union bounding_box *box, struct gds_graphics *gfx)

  Update the given bounding box with the bounding box of a graphics element.
- void calculate_cell_bounding_box (union bounding_box *box, struct gds_cell *cell)

  Calculate bounding box of a gds cell.

13.30.1 Detailed Description

Calculation of gds_cell trigonometrics.

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file cell-geometrics.c.

13.31 cell-geometrics.c

Go to the documentation of this file.

```
00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018  Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter.  If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00026 #include <math.h>
00027
00028 #include <gds-render/geometric/cell-geometrics.h>
00029
00035 static void convert_gds_point_to_2d_vector(struct gds_point *pt, struct vector_2d *vector)
00036 {
00037         vector->x = pt->x;
00038         vector->y = pt->y;
00039 }
00040
00046 static void update_box_with_gfx(union bounding_box *box, struct gds_graphics *gfx)
00047 {
00048         union bounding_box current_box;
00049
00050         bounding_box_prepare_empty(&current_box);
00051
00052         switch (gfx->gfx_type) {
00053         case GRAPHIC_BOX:
00054                 /* Expected fallthrough */
00055         case GRAPHIC_POLYGON:
00056                 bounding_box_calculate_from_polygon(gfx->vertices,
00057
       (conv_generic_to_vector_2d_t)&convert_gds_point_to_2d_vector,
00058                                                     &current_box);
00059                 break;
00060         case GRAPHIC_PATH:
00061                 /*
00062                  * This is not implemented correctly.
00063                  * Please be aware if paths are the outmost elements of your cell.
00064                  * You might end up with a completely wrong calculated cell size.
00065                  */
00066                 bounding_box_update_with_path(gfx->vertices, gfx->width_absolute,
00067
       (conv_generic_to_vector_2d_t)&convert_gds_point_to_2d_vector,
00068                                                     &current_box);
00069                 break;
00070         default:
00071                 /* Unknown graphics object. */
00072                 /* Print error? Nah.. */
00073                 break;
00074         }
00075
00076         /* Update box with results */
00077         bounding_box_update_with_box(box, &current_box);
00078 }
00079
00080 void calculate_cell_bounding_box(union bounding_box *box, struct gds_cell *cell)
00081 {
00082         GList *gfx_list;
00083         struct gds_graphics *gfx;
00084         GList *sub_cell_list;
00085         struct gds_cell_instance *sub_cell;
00086         union bounding_box temp_box;
00087
00088         if (!box || !cell)
00089                 return;
00090
00091         /* Update box with graphic elements */
00092         for (gfx_list = cell->graphic_objs; gfx_list != NULL; gfx_list = gfx_list->next) {
00093                 gfx = (struct gds_graphics *)gfx_list->data;
00094                 update_box_with_gfx(box, gfx);
00095         }
00096
```

```
00097         /* Update bounding box with boxes of subcells */
00098         for (sub_cell_list = cell->child_cells; sub_cell_list != NULL;
00099                                                 sub_cell_list = sub_cell_list->next) {
00100                 sub_cell = (struct gds_cell_instance *)sub_cell_list->data;
00101                 bounding_box_prepare_empty(&temp_box);
00102                 /* Recursion Woohoo!!  This dies if your GDS is faulty and contains a reference loop
      */
00103                 calculate_cell_bounding_box(&temp_box, sub_cell->cell_ref);
00104
00105                 /* Apply transformations */
00106                 bounding_box_apply_transform(ABS(sub_cell->magnification), sub_cell->angle,
00107                                              sub_cell->flipped, &temp_box);
00108
00109                 /* Move bounding box to origin */
00110                 temp_box.vectors.lower_left.x += sub_cell->origin.x;
00111                 temp_box.vectors.upper_right.x += sub_cell->origin.x;
00112                 temp_box.vectors.lower_left.y += sub_cell->origin.y;
00113                 temp_box.vectors.upper_right.y += sub_cell->origin.y;
00114
00115                 /* update the parent's box */
00116                 bounding_box_update_with_box(box, &temp_box);
00117         }
00118 }
00119
```

13.32 vector-operations.c File Reference

2D Vector operations

```
#include <math.h>
#include <stdlib.h>
#include <gds-render/geometric/vector-operations.h>
```

Include dependency graph for vector-operations.c:

Macros

- #define ABS_DBL(a) ((a) < 0.0 ? -(a) : (a))

Functions

- double vector_2d_scalar_multipy (struct vector_2d *a, struct vector_2d *b)
- void vector_2d_normalize (struct vector_2d *vec)
- void vector_2d_rotate (struct vector_2d *vec, double angle)
- struct vector_2d * vector_2d_copy (struct vector_2d *opt_res, struct vector_2d *vec)
- struct vector_2d * vector_2d_alloc (void)
- void vector_2d_free (struct vector_2d *vec)
- void vector_2d_scale (struct vector_2d *vec, double scale)
- double vector_2d_abs (struct vector_2d *vec)
- double vector_2d_calculate_angle_between (struct vector_2d *a, struct vector_2d *b)
- void vector_2d_subtract (struct vector_2d *res, struct vector_2d *a, struct vector_2d *b)
- void vector_2d_add (struct vector_2d *res, struct vector_2d *a, struct vector_2d *b)

13.32.1 Detailed Description

2D Vector operations

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file vector-operations.c.

13.33 vector-operations.c

Go to the documentation of this file.

```
00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018  Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter.  If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00031 #include <math.h>
00032 #include <stdlib.h>
00033
00034 #include <gds-render/geometric/vector-operations.h>
00035
00036 #define ABS_DBL(a) ((a) < 0.0 ? -(a) : (a))
00037
00038 double vector_2d_scalar_multipy(struct vector_2d *a, struct vector_2d *b)
00039 {
00040         if (a && b)
00041                 return (a->x * b->x) + (a->y * b->y);
00042         else
00043                 return 0.0;
00044 }
00045
00046 void vector_2d_normalize(struct vector_2d *vec)
00047 {
00048         double len;
00049
00050         if (!vec)
00051                 return;
00052         len = sqrt(pow(vec->x, 2) + pow(vec->y, 2));
00053         vec->x = vec->x/len;
00054         vec->y = vec->y/len;
00055 }
00056
00057 void vector_2d_rotate(struct vector_2d *vec, double angle)
00058 {
00059         double sin_val, cos_val;
00060         struct vector_2d temp;
00061
00062         if (!vec)
00063                 return;
00064
00065         sin_val = sin(angle);
00066         cos_val = cos(angle);
00067
00068         (void)vector_2d_copy(&temp, vec);
00069
00070         /* Apply rotation matrix */
00071         vec->x = (cos_val * temp.x) - (sin_val * temp.y);
00072         vec->y = (sin_val * temp.x) + (cos_val * temp.y);
00073 }
00074
00075 struct vector_2d *vector_2d_copy(struct vector_2d *opt_res, struct vector_2d *vec)
00076 {
00077         struct vector_2d *res;
00078
00079         if (!vec)
00080                 return NULL;
00081
00082         if (opt_res)
00083                 res = opt_res;
00084         else
00085                 res = vector_2d_alloc();
00086
00087         if (res) {
00088                 res->x = vec->x;
00089                 res->y = vec->y;
00090         }
00091                 return res;
00092 }
00093
```

```
00094 struct vector_2d *vector_2d_alloc(void)
00095 {
00096         return (struct vector_2d *)malloc(sizeof(struct vector_2d));
00097 }
00098
00099 void vector_2d_free(struct vector_2d *vec)
00100 {
00101         if (vec)
00102                 free(vec);
00103 }
00104
00105 void vector_2d_scale(struct vector_2d *vec, double scale)
00106 {
00107         if (!vec)
00108                 return;
00109
00110         vec->x *= scale;
00111         vec->y *= scale;
00112 }
00113
00114 double vector_2d_abs(struct vector_2d *vec)
00115 {
00116         double len = 0.0;
00117
00118         if (vec)
00119                 len = sqrt(pow(vec->x, 2) + pow(vec->y, 2));
00120         return len;
00121 }
00122
00123 double vector_2d_calculate_angle_between(struct vector_2d *a, struct vector_2d *b)
00124 {
00125         double cos_angle;
00126
00127         if (!a || !b)
00128                 return 0.0;
00129
00130         cos_angle = ABS_DBL(vector_2d_scalar_multipy(a, b)) / (vector_2d_abs(a) * vector_2d_abs(b));
00131         return acos(cos_angle);
00132 }
00133
00134 void vector_2d_subtract(struct vector_2d *res, struct vector_2d *a, struct vector_2d *b)
00135 {
00136         if (res && a && b) {
00137                 res->x = a->x - b->x;
00138                 res->y = a->y - b->y;
00139         }
00140 }
00141
00142 void vector_2d_add(struct vector_2d *res, struct vector_2d *a, struct vector_2d *b)
00143 {
00144         if (res && a && b) {
00145                 res->x = a->x + b->x;
00146                 res->y = a->y + b->y;
00147         }
00148 }
00149
```

13.34 lib-cell-renderer.h File Reference

Header file for the LibCellRenderer GObject Class.

`#include <gtk/gtk.h>`
Include dependency graph for lib-cell-renderer.h: This graph shows which files directly or indirectly include this file:

Data Structures

- struct _LibCellRenderer

Macros

- #define TYPE_LIB_CELL_RENDERER (lib_cell_renderer_get_type())
- #define LIB_CELL_RENDERER_ERROR_WARN (1U<<0)
- #define LIB_CELL_RENDERER_ERROR_ERR (1U<<1)

Typedefs

- typedef struct _LibCellRenderer LibCellRenderer

Functions

- GType lib_cell_renderer_get_type (void)

  lib_cell_renderer_get_type
- GtkCellRenderer * lib_cell_renderer_new (void)

  Create a new renderer for renderering gds_cell and gds_library elements.

13.34.1 Detailed Description

Header file for the LibCellRenderer GObject Class.

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file lib-cell-renderer.h.

13.35 lib-cell-renderer.h

Go to the documentation of this file.

```
00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018  Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter.  If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00031 #ifndef __LIB_CELL_RENDERER_H__
00032 #define __LIB_CELL_RENDERER_H__
00033
00034 #include <gtk/gtk.h>
00035
00036 G_BEGIN_DECLS
00037
00038 G_DECLARE_FINAL_TYPE(LibCellRenderer, lib_cell_renderer, LIB_CELL, RENDERER, GtkCellRendererText)
00039 #define TYPE_LIB_CELL_RENDERER (lib_cell_renderer_get_type())
00040
00044 #define LIB_CELL_RENDERER_ERROR_WARN (1U<<0)
00045 #define LIB_CELL_RENDERER_ERROR_ERR (1U<<1)
00048 typedef struct _LibCellRenderer {
00049         /* Inheritance */
00050         GtkCellRendererText super;
00051         /* Custom Elements */
00052 } LibCellRenderer;
00053
00058 GType lib_cell_renderer_get_type(void);
00059
00064 GtkCellRenderer *lib_cell_renderer_new(void);
00065
00066 G_END_DECLS
00067
00068 #endif /* __LIB_CELL_RENDERER_H__ */
00069
```

13.36 command-line.h File Reference

Render according to command line parameters.

`#include <glib.h>`
Include dependency graph for command-line.h: This graph shows which files directly or indirectly include this file:

Data Structures

- struct external_renderer_params

  External renderer paramameters to command line renderer.

Functions

- int command_line_convert_gds (const char *gds_name, const char *cell_name, char **renderers, char **output_file_names, const char *layer_file, struct external_renderer_params *ext_param, gboolean tex↩_standalone, gboolean tex_layers, double scale)

  Convert GDS according to command line parameters.

13.36.1 Detailed Description

Render according to command line parameters.

Author

Mario Hüttel `mario.huettel@gmx.net`

Definition in file command-line.h.

13.37 command-line.h

Go to the documentation of this file.

```
00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018  Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter.  If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00031 #ifndef _COMMAND_LINE_H_
00032 #define _COMMAND_LINE_H_
00033
00034 #include <glib.h>
00035
00039 struct external_renderer_params {
00043         char *so_path;
```

```
00044
00048         char *cli_params;
00049 };
00050
00064 int command_line_convert_gds(const char *gds_name,
00065                              const char *cell_name,
00066                              char **renderers,
00067                              char **output_file_names,
00068                              const char *layer_file,
00069                              struct external_renderer_params *ext_param,
00070                              gboolean tex_standalone,
00071                              gboolean tex_layers,
00072                              double scale);
00073
00074 #endif /* _COMMAND_LINE_H_ */
00075
```

13.38 gds-render-gui.h File Reference

Header for GdsRenderGui Object.

`#include <gtk/gtk.h>`
Include dependency graph for gds-render-gui.h: This graph shows which files directly or indirectly include this file:

Macros

- #define RENDERER_TYPE_GUI (gds_render_gui_get_type())

Functions

- G_BEGIN_DECLS G_DECLARE_FINAL_TYPE (GdsRenderGui, gds_render_gui, RENDERER, GUI, GObject)
- GdsRenderGui * gds_render_gui_new ()

  Create new GdsRenderGui Object.
- GtkWindow * gds_render_gui_get_main_window (GdsRenderGui *gui)

  Get main window.

13.38.1 Detailed Description

Header for GdsRenderGui Object.

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file gds-render-gui.h.

13.39 gds-render-gui.h

Go to the documentation of this file.

```
00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018  Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter.  If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00026 #ifndef _GDS_RENDER_GUI_
00027 #define _GDS_RENDER_GUI_
00028
00034 #include <gtk/gtk.h>
00035
00036 G_BEGIN_DECLS
00037
00038 G_DECLARE_FINAL_TYPE(GdsRenderGui, gds_render_gui, RENDERER, GUI, GObject);
00039
00040 #define RENDERER_TYPE_GUI (gds_render_gui_get_type())
00041
00046 GdsRenderGui *gds_render_gui_new();
00047
00055 GtkWindow *gds_render_gui_get_main_window(GdsRenderGui *gui);
00056
00057 G_END_DECLS
00058
00061 #endif /* _GDS_RENDER_GUI_ */
```

13.40 gds-parser.h File Reference

Header file for the GDS-Parser.

```
#include <glib.h>
#include <gds-render/gds-utils/gds-types.h>
```

Include dependency graph for gds-parser.h: This graph shows which files directly or indirectly include this file:

Macros

- #define GDS_PRINT_DEBUG_INFOS (0)

  1: Print infos, 0: Don't print

Functions

- int parse_gds_from_file (const char *filename, GList **library_array)

  Parse a GDS file.
- int clear_lib_list (GList **library_list)

  Deletes all libraries including cells, references etc.

13.40.1 Detailed Description

Header file for the GDS-Parser.

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file gds-parser.h.

13.41 gds-parser.h

Go to the documentation of this file.

```
00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018  Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter.  If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00026 #ifndef _GDSPARSER_H_
00027 #define _GDSPARSER_H_
00028
00034 #include <glib.h>
00035
00036 #include <gds-render/gds-utils/gds-types.h>
00037
00038 #define GDS_PRINT_DEBUG_INFOS (0)
00053 int parse_gds_from_file(const char *filename, GList **library_array);
00054
00060 int clear_lib_list(GList **library_list);
00061
00064 #endif /* _GDSPARSER_H_ */
```

13.42 gds-tree-checker.h File Reference

Checking functions of a cell tree (Header)

`#include <gds-render/gds-utils/gds-types.h>`
Include dependency graph for gds-tree-checker.h: This graph shows which files directly or indirectly include this file:

Functions

- int gds_tree_check_cell_references (struct gds_library *lib)

  gds_tree_check_cell_references checks if all child cell references can be resolved in the given library
- int gds_tree_check_reference_loops (struct gds_library *lib)

  gds_tree_check_reference_loops checks if the given library contains reference loops

13.42.1 Detailed Description

Checking functions of a cell tree (Header)

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file gds-tree-checker.h.

13.43 gds-tree-checker.h

Go to the documentation of this file.

```
00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2019  Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter.  If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00031 #ifndef _GDS_TREE_CHECKER_H_
00032 #define _GDS_TREE_CHECKER_H_
00033
00034 #include <gds-render/gds-utils/gds-types.h>
00035
00049 int gds_tree_check_cell_references(struct gds_library *lib);
00050
00057 int gds_tree_check_reference_loops(struct gds_library *lib);
00058
00059 #endif /* _GDS_TREE_CHECKER_H_ */
00060
```

13.44 gds-types.h File Reference

Defines types and macros used by the GDS-Parser.

```
#include <stdint.h>
#include <glib.h>
```

Include dependency graph for gds-types.h: This graph shows which files directly or indirectly include this file:

Data Structures

- struct gds_point

  A point in the 2D plane. Sometimes referred to as vertex.
- struct gds_cell_checks

  Stores the result of the cell checks.
- struct gds_cell_checks::_check_internals

  For the internal use of the checker.

- struct gds_time_field

  Date information for cells and libraries.
- struct gds_graphics

  A GDS graphics object.
- struct gds_cell_instance

  This represents an instanc of a cell inside another cell.
- struct gds_cell

  A Cell inside a gds_library.
- struct gds_library

  GDS Toplevel library.

Macros

- #define CELL_NAME_MAX (100)

  Maximum length of a gds_cell::name or a gds_library::name.
- #define MIN(a, b) (((a) < (b)) ? (a) : (b))

  Return smaller number.
- #define MAX(a, b) (((a) > (b)) ? (a) : (b))

  Return bigger number.

Enumerations

- enum { GDS_CELL_CHECK_NOT_RUN = -1 }

  Defintion of check counter default value that indicates that the corresponding check has not yet been executed.
- enum graphics_type { GRAPHIC_PATH = 0 , GRAPHIC_POLYGON = 1 , GRAPHIC_BOX = 2 }

  Types of graphic objects.
- enum path_type { PATH_FLUSH = 0 , PATH_ROUNDED = 1 , PATH_SQUARED = 2 }

  Defines the line caps of a path.

13.44.1 Detailed Description

Defines types and macros used by the GDS-Parser.

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file gds-types.h.

13.45 gds-types.h

Go to the documentation of this file.

```
00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018  Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter.  If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00031 #ifndef __GDS_TYPES_H__
00032 #define __GDS_TYPES_H__
00033
00034 #include <stdint.h>
00035 #include <glib.h>
00036
00037 #define CELL_NAME_MAX (100)
00039 /* Maybe use the macros that ship with the compiler? */
00040 #define MIN(a,b) (((a) < (b)) ? (a) : (b))
00041 #define MAX(a,b) (((a) > (b)) ? (a) : (b))
00045 enum {GDS_CELL_CHECK_NOT_RUN = -1};
00046
00048 enum graphics_type
00049 {
00050                     GRAPHIC_PATH = 0,
00051                     GRAPHIC_POLYGON = 1,
00052                     GRAPHIC_BOX = 2
00053 };
00054
00058 enum path_type {PATH_FLUSH = 0, PATH_ROUNDED = 1, PATH_SQUARED = 2};
00063 struct gds_point {
00064         int x;
00065         int y;
00066 };
00067
00071 struct gds_cell_checks {
00072         int unresolved_child_count;
00073         int affected_by_reference_loop;
00078         struct _check_internals {
00079                 int marker;
00080         } _internal;
00081 };
00082
00086 struct gds_time_field {
00087         uint16_t year;
00088         uint16_t month;
00089         uint16_t day;
00090         uint16_t hour;
00091         uint16_t minute;
00092         uint16_t second;
00093 };
00094
00098 struct gds_graphics {
00099         enum graphics_type gfx_type;
00100         GList *vertices;
00101         enum path_type path_render_type;
00102         int width_absolute;
00103         int16_t layer;
00104         int16_t datatype;
00105 };
00106
00110 struct gds_cell_instance {
00111         char ref_name[CELL_NAME_MAX];
00112         struct gds_cell *cell_ref;
00113         struct gds_point origin;
00114         int flipped;
00115         double angle;
00116         double magnification;
00117 };
00118
00122 struct gds_cell {
00123         char name[CELL_NAME_MAX];
00124         struct gds_time_field mod_time;
```

```
00125         struct gds_time_field access_time;
00126         GList *child_cells;
00127         GList *graphic_objs;
00128         struct gds_library *parent_library;
00129         struct gds_cell_checks checks;
00130 };
00131
00135 struct gds_library {
00136         char name[CELL_NAME_MAX];
00137         struct gds_time_field mod_time;
00138         struct gds_time_field access_time;
00139         double unit_in_meters;
00140         GList *cells;
00141         GList *cell_names ;
00142 };
00143
00146 #endif /* __GDS_TYPES_H__ */
```

13.46 bounding-box.h File Reference

Header for calculation of bounding boxes.

```
#include <glib.h>
#include <gds-render/geometric/vector-operations.h>
#include <stdbool.h>
```

Include dependency graph for bounding-box.h: This graph shows which files directly or indirectly include this file:

Data Structures

- union bounding_box

  Union describing a bounding box.
- struct bounding_box::_vectors

  Location vectors of upper right and lower left bounding box points.

Typedefs

- typedef void(∗ conv_generic_to_vector_2d_t) (void ∗, struct vector_2d ∗)

Functions

- void bounding_box_calculate_from_polygon (GList ∗vertices, conv_generic_to_vector_2d_t conv_func, union bounding_box ∗box)

  Calculate bounding box of polygon.
- void bounding_box_update_with_box (union bounding_box ∗destination, union bounding_box ∗update)

  Update an exisitng bounding box with another one.
- void bounding_box_prepare_empty (union bounding_box ∗box)

  Prepare an empty bounding box.
- void bounding_box_update_with_point (union bounding_box ∗destination, conv_generic_to_vector_2d_t conv_func, void ∗pt)

  Update bounding box with a point.
- void bounding_box_get_all_points (struct vector_2d ∗points, union bounding_box ∗box)

  Return all four corner points of a bounding box.
- void bounding_box_apply_transform (double scale, double rotation_deg, bool flip_at_x, union bounding_box ∗box)

  Apply transformations onto bounding box.
- void bounding_box_update_with_path (GList ∗vertices, double thickness, conv_generic_to_vector_2d_t conv_func, union bounding_box ∗box)

  Calculate the bounding box of a path and update the given bounding box.

13.46.1 Detailed Description

Header for calculation of bounding boxes.

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file bounding-box.h.

13.47 bounding-box.h

Go to the documentation of this file.

```
00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018  Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter.  If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00031 #ifndef _BOUNDING_BOX_H_
00032 #define _BOUNDING_BOX_H_
00033
00034 #include <glib.h>
00035 #include <gds-render/geometric/vector-operations.h>
00036 #include <stdbool.h>
00037
00055 union bounding_box {
00060         struct _vectors {
00062                 struct vector_2d lower_left;
00064                 struct vector_2d upper_right;
00065         } vectors;
00070         struct vector_2d vector_array[2];
00071 };
00072
00073 /*
00074  * @brief Pointer to a function that takes any pointer and converts this object to a vector_2d struct
00075  */
00076 typedef void (*conv_generic_to_vector_2d_t)(void *, struct vector_2d *);
00077
00084 void bounding_box_calculate_from_polygon(GList *vertices, conv_generic_to_vector_2d_t conv_func, union
       bounding_box *box);
00085
00091 void bounding_box_update_with_box(union bounding_box *destination, union bounding_box *update);
00092
00100 void bounding_box_prepare_empty(union bounding_box *box);
00101
00108 void bounding_box_update_with_point(union bounding_box *destination, conv_generic_to_vector_2d_t
       conv_func, void *pt);
00109
00115 void bounding_box_get_all_points(struct vector_2d *points, union bounding_box *box);
00116
00137 void bounding_box_apply_transform(double scale, double rotation_deg, bool flip_at_x, union
       bounding_box *box);
00138
00149 void bounding_box_update_with_path(GList *vertices, double thickness, conv_generic_to_vector_2d_t
       conv_func, union bounding_box *box);
00150
00151 #endif /* _BOUNDING_BOX_H_ */
00152
```

13.48 cell-geometrics.h File Reference

Calculation of gds_cell geometrics.

```
#include <gds-render/geometric/bounding-box.h>
#include <gds-render/gds-utils/gds-types.h>
```

Include dependency graph for cell-geometrics.h: This graph shows which files directly or indirectly include this file:

Functions

- void calculate_cell_bounding_box (union bounding_box *box, struct gds_cell *cell)

  Calculate bounding box of a gds cell.

13.48.1 Detailed Description

Calculation of gds_cell geometrics.

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file cell-geometrics.h.

13.49 cell-geometrics.h

Go to the documentation of this file.

```
00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018  Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter.  If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00031 #ifndef _CELL_GEOMETRICS_H_
00032 #define _CELL_GEOMETRICS_H_
00033
00034 #include <gds-render/geometric/bounding-box.h>
00035 #include <gds-render/gds-utils/gds-types.h>
00036
00050 void calculate_cell_bounding_box(union bounding_box *box, struct gds_cell *cell);
00051
00052 #endif /* _CELL_GEOMETRICS_H_ */
00053
```

13.50 vector-operations.h File Reference

Header for 2D Vector operations.

`#include <math.h>`
Include dependency graph for vector-operations.h: This graph shows which files directly or indirectly include this file:

Data Structures

- struct vector_2d

Macros

- #define DEG2RAD(a) ((a)*M_PI/180.0)

Functions

- double vector_2d_scalar_multipy (struct vector_2d *a, struct vector_2d *b)
- void vector_2d_normalize (struct vector_2d *vec)
- void vector_2d_rotate (struct vector_2d *vec, double angle)
- struct vector_2d * vector_2d_copy (struct vector_2d *opt_res, struct vector_2d *vec)
- struct vector_2d * vector_2d_alloc (void)
- void vector_2d_free (struct vector_2d *vec)
- void vector_2d_scale (struct vector_2d *vec, double scale)
- double vector_2d_abs (struct vector_2d *vec)
- double vector_2d_calculate_angle_between (struct vector_2d *a, struct vector_2d *b)
- void vector_2d_subtract (struct vector_2d *res, struct vector_2d *a, struct vector_2d *b)
- void vector_2d_add (struct vector_2d *res, struct vector_2d *a, struct vector_2d *b)

13.50.1 Detailed Description

Header for 2D Vector operations.

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file vector-operations.h.

13.51 vector-operations.h

Go to the documentation of this file.

```
00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018  Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter.  If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00032 #ifndef _VECTOR_OPERATIONS_H_
00033 #define _VECTOR_OPERATIONS_H_
00034
00035 #include <math.h>
00036
00037 struct vector_2d {
00038     double x;
00039     double y;
00040 };
00041
00042 #define DEG2RAD(a) ((a)*M_PI/180.0)
00043
00044 double vector_2d_scalar_multipy(struct vector_2d *a, struct vector_2d *b);
00045 void vector_2d_normalize(struct vector_2d *vec);
00046 void vector_2d_rotate(struct vector_2d *vec, double angle);
00047 struct vector_2d *vector_2d_copy(struct vector_2d *opt_res, struct vector_2d *vec);
00048 struct vector_2d *vector_2d_alloc(void);
00049 void vector_2d_free(struct vector_2d *vec);
00050 void vector_2d_scale(struct vector_2d *vec, double scale);
00051 double vector_2d_abs(struct vector_2d *vec);
00052 double vector_2d_calculate_angle_between(struct vector_2d *a, struct vector_2d *b);
00053 void vector_2d_subtract(struct vector_2d *res, struct vector_2d *a, struct vector_2d *b);
00054 void vector_2d_add(struct vector_2d *res, struct vector_2d *a, struct vector_2d *b);
00055
00056 #endif /* _VECTOR_OPERATIONS_H_ */
00057
```

13.52 color-palette.h File Reference

Class representing a color palette.

```
#include <glib.h>
#include <gtk/gtk.h>
```

Include dependency graph for color-palette.h: This graph shows which files directly or indirectly include this file:

Macros

- #define TYPE_GDS_RENDER_COLOR_PALETTE (color_palette_get_type())

Functions

- G_BEGIN_DECLS G_DECLARE_FINAL_TYPE (ColorPalette, color_palette, GDS_RENDER, COLOR_↩ PALETTE, GObject)
- ColorPalette ∗ color_palette_new_from_resource (char ∗resource_name)

  Create a new object with from a resource containing the html hex color scheme.
- GdkRGBA ∗ color_palette_get_color (ColorPalette ∗palette, GdkRGBA ∗color, unsigned int index)

  Get the n-th color in the palette identified by the index.
- unsigned int color_palette_get_color_count (ColorPalette ∗palette)

  Return amount of stored colors in `palette`.

13.52.1 Detailed Description

Class representing a color palette.

Author

Mario Hüttel `mario.huettel@gmx.net`

Definition in file color-palette.h.

13.52.2 Macro Definition Documentation

13.52.2.1 TYPE_GDS_RENDER_COLOR_PALETTE

```
#define TYPE_GDS_RENDER_COLOR_PALETTE (color_palette_get_type())
```

Definition at line 36 of file color-palette.h.

13.52.3 Function Documentation

13.52.3.1 color_palette_get_color()

```
GdkRGBA * color_palette_get_color (
            ColorPalette * palette,
            GdkRGBA * color,
            unsigned int index )
```

Get the n-th color in the palette identified by the index.

This function fills the nth color into the supplied `color`. `color` is returned.

If `color` is NULL, a new GdkRGBA is created and returned. This element must be freed afterwards.

Parameters

palette	Color palette
color	GdkRGBA struct to fill data in. May be NULL.
index	Index of color. Starts at 0

Returns

GdkRGBA color. If `color` is NULL, the returned color must be freed afterwards

Definition at line 199 of file color-palette.c.

Here is the caller graph for this function:

13.52.3.2 color_palette_get_color_count()

```
unsigned int color_palette_get_color_count (
            ColorPalette * palette )
```

Return amount of stored colors in `palette`.

Parameters

palette	Color palette

Returns

Count of colors

Definition at line 223 of file color-palette.c.

Here is the caller graph for this function:

13.52.3.3 color_palette_new_from_resource()

```
ColorPalette * color_palette_new_from_resource (
            char * resource_name )
```

Create a new object with from a resource containing the html hex color scheme.

Parameters

resource_name	Name of the resource

Returns

New object

Definition at line 188 of file color-palette.c.

Here is the call graph for this function: Here is the caller graph for this function:

13.52.3.4 G_DECLARE_FINAL_TYPE()

```
G_BEGIN_DECLS G_DECLARE_FINAL_TYPE (
            ColorPalette ,
            color_palette ,
            GDS_RENDER ,
            COLOR_PALETTE ,
            GObject  )
```

13.53 color-palette.h

Go to the documentation of this file.

```
00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2019  Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter.  If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00026 #ifndef _COLOR_PALETTE_H_
00027 #define _COLOR_PALETTE_H_
00028
00029 #include <glib.h>
00030 #include <gtk/gtk.h>
00031
00032 G_BEGIN_DECLS
00033
00034 G_DECLARE_FINAL_TYPE(ColorPalette, color_palette, GDS_RENDER, COLOR_PALETTE, GObject);
00035
00036 #define TYPE_GDS_RENDER_COLOR_PALETTE (color_palette_get_type())
00037
00043 ColorPalette *color_palette_new_from_resource(char *resource_name);
00044
00059 GdkRGBA *color_palette_get_color(ColorPalette *palette, GdkRGBA *color, unsigned int index);
00060
00066 unsigned int color_palette_get_color_count(ColorPalette *palette);
00067
00068 G_END_DECLS
00069
00070 #endif /* _COLOR_PALETTE_H_ */
```

13.54 layer-selector.h File Reference

Implementation of the Layer selection list.

```
#include <gtk/gtk.h>
#include <glib.h>
#include <gds-render/layer/color-palette.h>
#include <gds-render/layer/layer-settings.h>
```

Include dependency graph for layer-selector.h: This graph shows which files directly or indirectly include this file:

Macros

- #define TYPE_LAYER_SELECTOR (layer_selector_get_type())

Enumerations

- enum layer_selector_sort_algo { LAYER_SELECTOR_SORT_DOWN = 0 , LAYER_SELECTOR_SORT_UP }

  Defines how to sort the layer selector list box.

Functions

- G_BEGIN_DECLS G_DECLARE_FINAL_TYPE (LayerSelector, layer_selector, LAYER, SELECTOR, GObject)
- LayerSelector * layer_selector_new (GtkListBox *list_box)

  layer_selector_new
- void layer_selector_generate_layer_widgets (LayerSelector *selector, GList *libs)

  Generate layer widgets in in the LayerSelector instance.
- void layer_selector_set_load_mapping_button (LayerSelector *selector, GtkWidget *button, GtkWindow *main_window)

  Supply button for loading the layer mapping.
- void layer_selector_set_save_mapping_button (LayerSelector *selector, GtkWidget *button, GtkWindow *main_window)

  Supply button for saving the layer mapping.
- LayerSettings * layer_selector_export_rendered_layer_info (LayerSelector *selector)

  Get a list of all layers that shall be exported when rendering the cells.
- void layer_selector_force_sort (LayerSelector *selector, enum layer_selector_sort_algo sort_function)

  Force the layer selector list to be sorted according to `sort_function`.
- void layer_selector_select_all_layers (LayerSelector *layer_selector, gboolean select)

  Set 'export' value of all layers in the LayerSelector to the supplied select value.
- void layer_selector_auto_color_layers (LayerSelector *layer_selector, ColorPalette *palette, double global_alpha)

  Apply colors from palette to all layers. Aditionally set alpha.
- void layer_selector_auto_name_layers (LayerSelector *layer_selector, gboolean overwrite)

  Auto name all layers in the layer selector.
- gboolean layer_selector_contains_elements (LayerSelector *layer_selector)

  Check if the given layer selector contains layer elements.
- size_t layer_selector_num_of_named_elements (LayerSelector *layer_selector)

  Get number of layer elements that are named.

13.54.1 Detailed Description

Implementation of the Layer selection list.

Author

Mario Hüttel `mario.huettel@gmx.net`

Definition in file layer-selector.h.

13.55 layer-selector.h

Go to the documentation of this file.

```
00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018  Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
```

```
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter.  If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00031 #ifndef __LAYER_SELECTOR_H__
00032 #define __LAYER_SELECTOR_H__
00033
00034 #include <gtk/gtk.h>
00035 #include <glib.h>
00036 #include <gds-render/layer/color-palette.h>
00037 #include <gds-render/layer/layer-settings.h>
00038
00039 G_BEGIN_DECLS
00040
00041 G_DECLARE_FINAL_TYPE(LayerSelector, layer_selector, LAYER, SELECTOR, GObject);
00042
00043 #define TYPE_LAYER_SELECTOR (layer_selector_get_type())
00044
00048 enum layer_selector_sort_algo {LAYER_SELECTOR_SORT_DOWN = 0, LAYER_SELECTOR_SORT_UP};
00049
00055 LayerSelector *layer_selector_new(GtkListBox *list_box);
00056
00063 void layer_selector_generate_layer_widgets(LayerSelector *selector, GList *libs);
00064
00071 void layer_selector_set_load_mapping_button(LayerSelector *selector, GtkWidget *button, GtkWindow
       *main_window);
00072
00079 void layer_selector_set_save_mapping_button(LayerSelector *selector, GtkWidget *button, GtkWindow
       *main_window);
00080
00086 LayerSettings *layer_selector_export_rendered_layer_info(LayerSelector *selector);
00087
00093 void layer_selector_force_sort(LayerSelector *selector, enum layer_selector_sort_algo sort_function);
00094
00100 void layer_selector_select_all_layers(LayerSelector *layer_selector, gboolean select);
00101
00108 void layer_selector_auto_color_layers(LayerSelector *layer_selector, ColorPalette *palette, double
       global_alpha);
00109
00119 void layer_selector_auto_name_layers(LayerSelector *layer_selector, gboolean overwrite);
00120
00131 gboolean layer_selector_contains_elements(LayerSelector *layer_selector);
00132
00138 size_t layer_selector_num_of_named_elements(LayerSelector *layer_selector);
00139
00140 G_END_DECLS
00141
00142 #endif /* __LAYER_SELECTOR_H__ */
00143
```

13.56 layer-settings.h File Reference

LayerSettings class header file.

`#include <gtk/gtk.h>`
Include dependency graph for layer-settings.h: This graph shows which files directly or indirectly include this file:

Data Structures

- struct layer_info

  Layer information.

Macros

- #define GDS_RENDER_TYPE_LAYER_SETTINGS (layer_settings_get_type())
- #define CSV_LINE_MAX_LEN (1024)

  Maximum length of a layer mapping CSV line.

Functions

- LayerSettings * layer_settings_new ()
  New LayerSettings object.
- int layer_settings_append_layer_info (LayerSettings *settings, struct layer_info *info)
  layer_settings_append_layer_info
- void layer_settings_clear (LayerSettings *settings)
  Clear all layers in this settings object.
- int layer_settings_remove_layer (LayerSettings *settings, int layer)
  Remove a specific layer number from the layer settings.
- GList * layer_settings_get_layer_info_list (LayerSettings *settings)
  Get a GList with layer_info structs.
- int layer_settings_to_csv (LayerSettings *settings, const char *path)
  Write layer settings to a CSV file.
- int layer_settings_load_from_csv (LayerSettings *settings, const char *path)
  Load new layer Settings from CSV.

13.56.1 Detailed Description

LayerSettings class header file.

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file layer-settings.h.

13.56.2 Macro Definition Documentation

13.56.2.1 CSV_LINE_MAX_LEN

```
#define CSV_LINE_MAX_LEN (1024)
```

Maximum length of a layer mapping CSV line.

Definition at line 56 of file layer-settings.h.

13.56.2.2 GDS_RENDER_TYPE_LAYER_SETTINGS

```
#define GDS_RENDER_TYPE_LAYER_SETTINGS (layer_settings_get_type())
```

Definition at line 51 of file layer-settings.h.

13.56.3 Function Documentation

13.56.3.1 layer_settings_append_layer_info()

```
int layer_settings_append_layer_info (
            LayerSettings * settings,
            struct layer_info * info )
```

layer_settings_append_layer_info

Parameters

settings	LayerSettings object.
info	Info to append

Returns

Error code. 0 if successful

Note

`info` is copied internally. You can free this struct afterwards.

Definition at line 111 of file layer-settings.c.

Here is the call graph for this function: Here is the caller graph for this function:

13.56.3.2 layer_settings_clear()

```
void layer_settings_clear (
            LayerSettings * settings )
```

Clear all layers in this settings object.

Parameters

settings	LayerSettings object

Definition at line 128 of file layer-settings.c.

Here is the call graph for this function: Here is the caller graph for this function:

13.56.3.3 layer_settings_get_layer_info_list()

```
GList * layer_settings_get_layer_info_list (
            LayerSettings * settings )
```

Get a GList with layer_info structs.

This function returns a GList with all layer_info structs in rendering order (bottom to top) that shall be rendered.

Parameters

settings	LayerSettings object

Returns

GList with struct layer_info elements.

Definition at line 166 of file layer-settings.c.

Here is the caller graph for this function:

13.56.3.4 layer_settings_load_from_csv()

```
int layer_settings_load_from_csv (
            LayerSettings * settings,
            const char * path )
```

Load new layer Settings from CSV.

This function loads the layer information from a CSV file. All data inside the `settings` is cleared beforehand.

Parameters

settings	Settings to write to.
path	CSV file path

Returns

0 if successful

Definition at line 310 of file layer-settings.c.

Here is the call graph for this function: Here is the caller graph for this function:

13.56.3.5 layer_settings_new()

```
LayerSettings * layer_settings_new ( )
```

New LayerSettings object.

Returns

New object

Definition at line 106 of file layer-settings.c.

Here is the caller graph for this function:

13.56.3.6 layer_settings_remove_layer()

```
int layer_settings_remove_layer (
            LayerSettings * settings,
            int layer )
```

Remove a specific layer number from the layer settings.

Parameters

settings	LayerSettings object
layer	Layer number

Returns

Error code. 0 if successful

Definition at line 137 of file layer-settings.c.

Here is the call graph for this function:

13.56.3.7 layer_settings_to_csv()

```
int layer_settings_to_csv (
            LayerSettings * settings,
            const char * path )
```

Write layer settings to a CSV file.

This function writes the layer settings to a CSV file according to the layer mapping specification (Layer Mapping File Specification)

Parameters

settings	LayerSettings object
path	Output path for CSV file.

Returns

0 if successful

Definition at line 196 of file layer-settings.c.

Here is the call graph for this function: Here is the caller graph for this function:

13.57 layer-settings.h

Go to the documentation of this file.

```
00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2019  Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00014  * GNU General Public License for more details.
00015  *
```

```
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter.  If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00026 #ifndef _LAYER_INFO_H_
00027 #define _LAYER_INFO_H_
00028
00029 #include <gtk/gtk.h>
00030
00031 G_BEGIN_DECLS
00032
00040 struct layer_info
00041 {
00042         int layer;
00043         char *name;
00044         int stacked_position;
00045         GdkRGBA color;
00046         int render;
00047 };
00048
00049 G_DECLARE_FINAL_TYPE(LayerSettings, layer_settings, GDS_RENDER, LAYER_SETTINGS, GObject)
00050
00051 #define GDS_RENDER_TYPE_LAYER_SETTINGS (layer_settings_get_type())
00052
00056 #define CSV_LINE_MAX_LEN (1024)
00057
00062 LayerSettings *layer_settings_new();
00063
00071 int layer_settings_append_layer_info(LayerSettings *settings, struct layer_info *info);
00072
00077 void layer_settings_clear(LayerSettings *settings);
00078
00085 int layer_settings_remove_layer(LayerSettings *settings, int layer);
00086
00096 GList *layer_settings_get_layer_info_list(LayerSettings *settings);
00097
00107 int layer_settings_to_csv(LayerSettings *settings, const char *path);
00108
00119 int layer_settings_load_from_csv(LayerSettings *settings, const char *path);
00120
00121 G_END_DECLS
00122
00123 #endif // _LAYER_INFO_H_
```

13.58 cairo-renderer.h File Reference

Header File for Cairo output renderer.

```
#include <gds-render/gds-utils/gds-types.h>
#include <gds-render/output-renderers/gds-output-renderer.h>
#include <glib-object.h>
```

Include dependency graph for cairo-renderer.h: This graph shows which files directly or indirectly include this file:

Macros

- #define GDS_RENDER_TYPE_CAIRO_RENDERER (cairo_renderer_get_type())
- #define MAX_LAYERS (5000)

  Maximum layer count the output renderer can process. Typically GDS only specifies up to 255 layers.

Functions

- CairoRenderer * cairo_renderer_new_svg ()

  Create new CairoRenderer for SVG output.
- CairoRenderer * cairo_renderer_new_pdf ()

  Create new CairoRenderer for PDF output.

13.58.1 Detailed Description

Header File for Cairo output renderer.

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file cairo-renderer.h.

13.59 cairo-renderer.h

Go to the documentation of this file.

```
00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018  Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter.  If not, see <http://www.gnu.org/licenses/>.
00018  */
00024 #ifndef _CAIRO_OUTPUT_H_
00025 #define _CAIRO_OUTPUT_H_
00026
00027 #include <gds-render/gds-utils/gds-types.h>
00028 #include <gds-render/output-renderers/gds-output-renderer.h>
00029 #include <glib-object.h>
00030
00031 G_BEGIN_DECLS
00032
00037 G_DECLARE_FINAL_TYPE(CairoRenderer, cairo_renderer, GDS_RENDER, CAIRO_RENDERER, GdsOutputRenderer)
00038
00039 #define GDS_RENDER_TYPE_CAIRO_RENDERER (cairo_renderer_get_type())
00040
00041 #define MAX_LAYERS (5000)
00047 CairoRenderer *cairo_renderer_new_svg();
00048
00049
00054 CairoRenderer *cairo_renderer_new_pdf();
00055
00058 G_END_DECLS
00059
00060 #endif /* _CAIRO_OUTPUT_H_ */
```

13.60 external-renderer-interfaces.h File Reference

This graph shows which files directly or indirectly include this file:

Macros

- #define xstr(a) str(a)
- #define str(a) #a
- #define EXPORT_FUNC __attribute__((visibility("default")))

  This define is used to export a function from a shared object.
- #define EXTERNAL_LIBRARY_RENDER_FUNCTION exported_render_cell_to_file

  Function name expected to be found in external library for rendering.
- #define EXTERNAL_LIBRARY_INIT_FUNCTION exported_init

  Function name expected to be found in external library for initialization.
- #define EXTERNAL_LIBRARY_FORK_REQUEST exported_fork_request

  Global integer specified by an external renderer to signal, that the init and render functions shall be executed in a subprocess.
- #define EXPORTED_FUNC_DECL(FUNC) EXPORT_FUNC FUNC

  Define for declaring the exported functions.

13.60.1 Macro Definition Documentation

13.60.1.1 str

```
#define str(
              a ) #a
```

Definition at line 26 of file external-renderer-interfaces.h.

13.60.1.2 xstr

```
#define xstr(
              a ) str(a)
```

Definition at line 25 of file external-renderer-interfaces.h.

13.61 external-renderer-interfaces.h

Go to the documentation of this file.

```
00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018  Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00014  * GNU General Public License for more details.
00015  *
```

```
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter.  If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00020 #ifndef __EXTERNAL_RENDERER_INTERFACES_H__
00021 #define __EXTERNAL_RENDERER_INTERFACES_H__
00022
00023 #ifndef xstr
00024
00025 #define xstr(a) str(a)
00026 #define str(a) #a
00027
00028 #endif /* xstr */
00029
00038 #define EXPORT_FUNC __attribute__((visibility("default")))
00039
00048 #define EXTERNAL_LIBRARY_RENDER_FUNCTION exported_render_cell_to_file
00049
00057 #define EXTERNAL_LIBRARY_INIT_FUNCTION exported_init
00058
00065 #define EXTERNAL_LIBRARY_FORK_REQUEST exported_fork_request
00066
00072 #define EXPORTED_FUNC_DECL(FUNC) EXPORT_FUNC FUNC
00073
00076 #endif /* __EXTERNAL_RENDERER_INTERFACES_H__ */
```

13.62 external-renderer.h File Reference

Render according to command line parameters.

```
#include <gds-render/output-renderers/gds-output-renderer.h>
#include <gds-render/gds-utils/gds-types.h>
#include <gds-render/output-renderers/external-renderer-interfaces.h>
```

Include dependency graph for external-renderer.h: This graph shows which files directly or indirectly include this file:

Macros

- #define GDS_RENDER_TYPE_EXTERNAL_RENDERER (external_renderer_get_type())

Functions

- ExternalRenderer * external_renderer_new ()

  Create new ExternalRenderer object.
- ExternalRenderer * external_renderer_new_with_so_and_param (const char *so_path, const char *param↩ _string)

  Create new ExternalRenderer object with specified shared object path.

13.62.1 Detailed Description

Render according to command line parameters.

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file external-renderer.h.

13.63 external-renderer.h

Go to the documentation of this file.

```
00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018  Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter.  If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00031 #ifndef _EXTERNAL_RENDERER_H_
00032 #define _EXTERNAL_RENDERER_H_
00033
00034 #include <gds-render/output-renderers/gds-output-renderer.h>
00035 #include <gds-render/gds-utils/gds-types.h>
00036 #include <gds-render/output-renderers/external-renderer-interfaces.h>
00037
00038 G_BEGIN_DECLS
00039
00040 #define GDS_RENDER_TYPE_EXTERNAL_RENDERER (external_renderer_get_type())
00041
00042 G_DECLARE_FINAL_TYPE(ExternalRenderer, external_renderer, GDS_RENDER, EXTERNAL_RENDERER,
       GdsOutputRenderer)
00043
00044
00048 ExternalRenderer *external_renderer_new();
00049
00056 ExternalRenderer *external_renderer_new_with_so_and_param(const char *so_path, const char
       *param_string);
00057
00058 G_END_DECLS
00059
00060 #endif /* _EXTERNAL_RENDERER_H_ */
00061
```

13.64 gds-output-renderer.h File Reference

Header for output renderer base class.

```
#include <gds-render/gds-utils/gds-types.h>
#include <glib-object.h>
#include <glib.h>
#include <gds-render/layer/layer-settings.h>
```

Include dependency graph for gds-output-renderer.h: This graph shows which files directly or indirectly include this file:

Data Structures

- struct _GdsOutputRendererClass

  Base output renderer class structure.

Macros

- #define GDS_RENDER_TYPE_OUTPUT_RENDERER (gds_output_renderer_get_type())

Enumerations

- enum { GDS_OUTPUT_RENDERER_GEN_ERR = -100 , GDS_OUTPUT_RENDERER_PARAM_ERR = -200 }

Functions

- G_DECLARE_DERIVABLE_TYPE (GdsOutputRenderer, gds_output_renderer, GDS_RENDER, OUTPUT↩ _RENDERER, GObject)
- GdsOutputRenderer ∗ gds_output_renderer_new ()

  Create a new GdsOutputRenderer GObject.
- GdsOutputRenderer ∗ gds_output_renderer_new_with_props (const char ∗output_file, LayerSettings ∗layer_settings)

  Create a new GdsOutputRenderer GObject with its properties.
- int gds_output_renderer_render_output (GdsOutputRenderer ∗renderer, struct gds_cell ∗cell, double scale)

  gds_output_renderer_render_output
- void gds_output_renderer_set_output_file (GdsOutputRenderer ∗renderer, const gchar ∗file_name)

  Convenience function for setting the "output-file" property.
- const char ∗ gds_output_renderer_get_output_file (GdsOutputRenderer ∗renderer)

  Convenience function for getting the "output-file" property.
- LayerSettings ∗ gds_output_renderer_get_and_ref_layer_settings (GdsOutputRenderer ∗renderer)

  Get layer settings.
- void gds_output_renderer_set_layer_settings (GdsOutputRenderer ∗renderer, LayerSettings ∗settings)

  Set layer settings.
- int gds_output_renderer_render_output_async (GdsOutputRenderer ∗renderer, struct gds_cell ∗cell, double scale)

  Render output asynchronously.
- void gds_output_renderer_update_async_progress (GdsOutputRenderer ∗renderer, const char ∗status)

  This function emits the 'progress-changed' in the thread/context that triggered an asynchronous rendering.

13.64.1 Detailed Description

Header for output renderer base class.

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file gds-output-renderer.h.

13.65 gds-output-renderer.h

Go to the documentation of this file.

```
00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018  Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter.  If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00031 #ifndef _GDS_OUTPUT_RENDERER_H_
00032 #define _GDS_OUTPUT_RENDERER_H_
00033
00034 #include <gds-render/gds-utils/gds-types.h>
00035 #include <glib-object.h>
00036 #include <glib.h>
00037 #include <gds-render/layer/layer-settings.h>
00038
00039 G_BEGIN_DECLS
00040
00041 #define GDS_RENDER_TYPE_OUTPUT_RENDERER (gds_output_renderer_get_type())
00042
00043 G_DECLARE_DERIVABLE_TYPE(GdsOutputRenderer, gds_output_renderer, GDS_RENDER, OUTPUT_RENDERER,
       GObject);
00044
00049 struct _GdsOutputRendererClass {
00050         GObjectClass parent_class;
00051
00055         int (*render_output)(GdsOutputRenderer *renderer,
00056                              struct gds_cell *cell,
00057                              double scale);
00058         gpointer padding[4];
00059 };
00060
00061 enum {
00062         GDS_OUTPUT_RENDERER_GEN_ERR = -100,
00063         GDS_OUTPUT_RENDERER_PARAM_ERR = -200
00064 };
00065
00070 GdsOutputRenderer *gds_output_renderer_new();
00071
00078 GdsOutputRenderer *gds_output_renderer_new_with_props(const char *output_file, LayerSettings
       *layer_settings);
00079
00087 int gds_output_renderer_render_output(GdsOutputRenderer *renderer,
00088                                       struct gds_cell *cell,
00089                                       double scale);
00090
00096 void gds_output_renderer_set_output_file(GdsOutputRenderer *renderer, const gchar *file_name);
00097
00103 const char *gds_output_renderer_get_output_file(GdsOutputRenderer *renderer);
00104
00116 LayerSettings *gds_output_renderer_get_and_ref_layer_settings(GdsOutputRenderer *renderer);
00117
00130 void gds_output_renderer_set_layer_settings(GdsOutputRenderer *renderer, LayerSettings *settings);
00131
00145 int gds_output_renderer_render_output_async(GdsOutputRenderer *renderer, struct gds_cell *cell, double
       scale);
00146
00155 void gds_output_renderer_update_async_progress(GdsOutputRenderer *renderer, const char *status);
00156
00157 G_END_DECLS
00158
00159 #endif /* _GDS_OUTPUT_RENDERER_H_ */
00160
```

13.66 latex-renderer.h File Reference

LaTeX output renderer.

```
#include <gds-render/output-renderers/gds-output-renderer.h>
#include <gds-render/gds-utils/gds-types.h>
```

Include dependency graph for latex-renderer.h: This graph shows which files directly or indirectly include this file:

Macros

- #define GDS_RENDER_TYPE_LATEX_RENDERER (latex_renderer_get_type())
- #define LATEX_LINE_BUFFER_KB (10)

  Buffer for LaTeX Code line in KiB.

Functions

- LatexRenderer * latex_renderer_new ()

  Create new LatexRenderer object.
- LatexRenderer * latex_renderer_new_with_options (gboolean pdf_layers, gboolean standalone)

  Create new LatexRenderer object.

13.66.1 Detailed Description

LaTeX output renderer.

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file latex-renderer.h.

13.67 latex-renderer.h

Go to the documentation of this file.

```
00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018  Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter.  If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00031 #ifndef _LATEX_OUTPUT_H_
00032 #define _LATEX_OUTPUT_H_
00033
00034 #include <gds-render/output-renderers/gds-output-renderer.h>
00035 #include <gds-render/gds-utils/gds-types.h>
00036
00037 G_BEGIN_DECLS
00038
00039 G_DECLARE_FINAL_TYPE(LatexRenderer, latex_renderer, GDS_RENDER, LATEX_RENDERER, GdsOutputRenderer)
00040
```

```
00041 #define GDS_RENDER_TYPE_LATEX_RENDERER (latex_renderer_get_type())
00042
00046 #define LATEX_LINE_BUFFER_KB (10)
00047
00052 LatexRenderer *latex_renderer_new();
00053
00068 LatexRenderer *latex_renderer_new_with_options(gboolean pdf_layers, gboolean standalone);
00069
00070 G_END_DECLS
00071
00072 #endif /* _LATEX_OUTPUT_H_ */
00073
```

13.68 version.h File Reference

This graph shows which files directly or indirectly include this file:

Variables

- const char * _app_version_string

  This string holds the Git Based Version Number of the app.
- const char * _app_git_commit

  This string holds the git commit hash of the current HEAD revision.

13.69 version.h

Go to the documentation of this file.

```
00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018  Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter.  If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00026 extern const char *_app_version_string;
00027
00029 extern const char *_app_git_commit;
```

13.70 activity-bar.h File Reference

Header file for activity bar widget.

`#include <gtk/gtk.h>`
Include dependency graph for activity-bar.h: This graph shows which files directly or indirectly include this file:

Macros

- #define TYPE_ACTIVITY_BAR (activity_bar_get_type())

Functions

- ActivityBar * activity_bar_new ()

  Create new Object ActivityBar.
- void activity_bar_set_ready (ActivityBar *bar)

  Deletes all applied tasks and sets bar to "Ready".
- void activity_bar_set_busy (ActivityBar *bar, const char *text)

  Enable spinner and set `text`. If text is NULL, 'Working...' is displayed.

13.70.1 Detailed Description

Header file for activity bar widget.

Author

Mario Hüttel `mario.huettel@gmx.net`

Definition in file activity-bar.h.

13.71 activity-bar.h

Go to the documentation of this file.

```
00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2019  Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter.  If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00032 #ifndef __LAYER_ELEMENT_H__
00033 #define __LAYER_ELEMENT_H__
00034
00035 #include <gtk/gtk.h>
00036
00037 G_BEGIN_DECLS
00038
00039 /* Creates Class structure etc */
00040 G_DECLARE_FINAL_TYPE(ActivityBar, activity_bar, ACTIVITY, BAR, GtkBox)
00041
00042 #define TYPE_ACTIVITY_BAR (activity_bar_get_type())
00043
00048 ActivityBar *activity_bar_new();
00049
00054 void activity_bar_set_ready(ActivityBar *bar);
00055
00062 void activity_bar_set_busy(ActivityBar *bar, const char *text);
00063
00064 G_END_DECLS
00065
00066 #endif /* __LAYER_ELEMENT_H__ */
00067
```

13.72 conv-settings-dialog.h File Reference

Header file for the Conversion Settings Dialog.

`#include <gtk/gtk.h>`
Include dependency graph for conv-settings-dialog.h: This graph shows which files directly or indirectly include this file:

Data Structures

- struct render_settings

  This struct holds the renderer configuration.

Macros

- #define RENDERER_TYPE_SETTINGS_DIALOG (renderer_settings_dialog_get_type())

Enumerations

- enum output_renderer { RENDERER_LATEX_TIKZ , RENDERER_CAIROGRAPHICS_PDF , RENDERER_CAIROGRAPHICS_
  }

  return type of the RedererSettingsDialog

Functions

- RendererSettingsDialog ∗ renderer_settings_dialog_new (GtkWindow ∗parent)

  Create a new RedererSettingsDialog GObject.
- G_END_DECLS void renderer_settings_dialog_set_settings (RendererSettingsDialog ∗dialog, struct render_settings ∗settings)

  Apply settings to dialog.
- void renderer_settings_dialog_get_settings (RendererSettingsDialog ∗dialog, struct render_settings ∗settings)

  Get the settings configured in the dialog.
- void renderer_settings_dialog_set_cell_width (RendererSettingsDialog ∗dialog, unsigned int width)

  renderer_settings_dialog_set_cell_width Set width for rendered cell
- void renderer_settings_dialog_set_cell_height (RendererSettingsDialog ∗dialog, unsigned int height)

  renderer_settings_dialog_set_cell_height Set height for rendered cell
- void renderer_settings_dialog_set_database_unit_scale (RendererSettingsDialog ∗dialog, double unit_in_↩ meters)

  renderer_settings_dialog_set_database_unit_scale Set database scale

13.72.1 Detailed Description

Header file for the Conversion Settings Dialog.

Author

Mario.Huettel@gmx.net mario.huettel@gmx.net

Definition in file conv-settings-dialog.h.

13.73 conv-settings-dialog.h

Go to the documentation of this file.

```
00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018  Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter.  If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00032 #ifndef __CONV_SETTINGS_DIALOG_H__
00033 #define __CONV_SETTINGS_DIALOG_H__
00034
00035 #include <gtk/gtk.h>
00036
00037 G_BEGIN_DECLS
00038
00040 enum output_renderer {RENDERER_LATEX_TIKZ, RENDERER_CAIROGRAPHICS_PDF, RENDERER_CAIROGRAPHICS_SVG};
00041
00042 G_DECLARE_FINAL_TYPE(RendererSettingsDialog, renderer_settings_dialog, RENDERER, SETTINGS_DIALOG,
      GtkDialog)
00043
00044
00049 RendererSettingsDialog *renderer_settings_dialog_new(GtkWindow *parent);
00050
00051 #define RENDERER_TYPE_SETTINGS_DIALOG (renderer_settings_dialog_get_type())
00052
00056 struct render_settings {
00057         double scale;
00058         enum output_renderer renderer;
00059         gboolean tex_pdf_layers;
00060         gboolean tex_standalone;
00061 };
00062
00063 G_END_DECLS
00064
00070 void renderer_settings_dialog_set_settings(RendererSettingsDialog *dialog, struct render_settings
      *settings);
00071
00077 void renderer_settings_dialog_get_settings(RendererSettingsDialog *dialog, struct render_settings
      *settings);
00078
00084 void renderer_settings_dialog_set_cell_width(RendererSettingsDialog *dialog, unsigned int width);
00085
00091 void renderer_settings_dialog_set_cell_height(RendererSettingsDialog *dialog, unsigned int height);
00092
00098 void renderer_settings_dialog_set_database_unit_scale(RendererSettingsDialog *dialog, double
      unit_in_meters);
00099
00100 #endif /* __CONV_SETTINGS_DIALOG_H__ */
00101
```

13.74 layer-element.h File Reference

Implementation of the layer element used for configuring layer colors etc.

`#include <gtk/gtk.h>`
Include dependency graph for layer-element.h: This graph shows which files directly or indirectly include this file:

Data Structures

- struct _LayerElementPriv
- struct _LayerElement
- struct layer_element_dnd_data

  This structure holds the necessary data to set up a LayerElement for Drag'n'Drop.

Macros

- #define TYPE_LAYER_ELEMENT (layer_element_get_type())

Typedefs

- typedef struct _LayerElementPriv LayerElementPriv

Functions

- GtkWidget * layer_element_new (void)
  Create new layer element object.
- const char * layer_element_get_name (LayerElement *elem)
  get name of the layer
- void layer_element_set_name (LayerElement *elem, const char *name)
  layer_element_set_name
- void layer_element_set_layer (LayerElement *elem, int layer)
  Set layer number for this layer.
- int layer_element_get_layer (LayerElement *elem)
  Get layer number.
- void layer_element_set_export (LayerElement *elem, gboolean export)
  Set export flag for this layer.
- gboolean layer_element_get_export (LayerElement *elem)
  Get export flag of layer.
- void layer_element_get_color (LayerElement *elem, GdkRGBA *rgba)
  Get color of layer.
- void layer_element_set_color (LayerElement *elem, GdkRGBA *rgba)
  Set color of layer.
- void layer_element_set_dnd_callbacks (LayerElement *elem, struct layer_element_dnd_data *data)
  Setup drag and drop of `elem` for use in the LayerSelector.

13.74.1 Detailed Description

Implementation of the layer element used for configuring layer colors etc.

Author

Mario Hüttel `mario.huettel@gmx.net`

Definition in file layer-element.h.

13.75 layer-element.h

Go to the documentation of this file.

```
00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018  Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter.  If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00032 #ifndef __LAYER_ELEMENT_H__
00033 #define __LAYER_ELEMENT_H__
00034
00035 #include <gtk/gtk.h>
00036
00037 G_BEGIN_DECLS
00038
00039 /* Creates Class structure etc */
00040 G_DECLARE_FINAL_TYPE(LayerElement, layer_element, LAYER, ELEMENT, GtkListBoxRow)
00041
00042 #define TYPE_LAYER_ELEMENT (layer_element_get_type())
00043
00044 typedef struct _LayerElementPriv {
00045         GtkEntry *name;
00046         GtkLabel *layer;
00047         int layer_num;
00048         GtkEventBox *event_handle;
00049         GtkColorButton *color;
00050         GtkCheckButton *export;
00051 } LayerElementPriv;
00052
00053 struct _LayerElement {
00054         /* Inheritance */
00055         GtkListBoxRow parent;
00056         /* Custom Elements */
00057         LayerElementPriv priv;
00058 };
00059
00063 struct layer_element_dnd_data {
00065         GtkTargetEntry *entries;
00067         int entry_count;
00069         void (*drag_begin)(GtkWidget *, GdkDragContext *, gpointer);
00071         void (*drag_data_get)(GtkWidget *, GdkDragContext *, GtkSelectionData *, guint, guint,
      gpointer);
00073         void (*drag_end)(GtkWidget *, GdkDragContext *, gpointer);
00074 };
00075
00080 GtkWidget *layer_element_new(void);
00081
00087 const char *layer_element_get_name(LayerElement *elem);
00088
00094 void layer_element_set_name(LayerElement *elem, const char* name);
00095
00101 void layer_element_set_layer(LayerElement *elem, int layer);
00102
00108 int layer_element_get_layer(LayerElement *elem);
00109
00115 void layer_element_set_export(LayerElement *elem, gboolean export);
00116
00122 gboolean layer_element_get_export(LayerElement *elem);
00123
00129 void layer_element_get_color(LayerElement *elem, GdkRGBA *rgba);
00130
00136 void layer_element_set_color(LayerElement *elem, GdkRGBA *rgba);
00137
00143 void layer_element_set_dnd_callbacks(LayerElement *elem, struct layer_element_dnd_data *data);
00144
00145 G_END_DECLS
00146
00147 #endif /* __LAYER_ELEMENT_H__ */
00148
```

13.76 color-palette.c File Reference

Class representing a color palette.

`#include <gds-render/layer/color-palette.h>`
Include dependency graph for color-palette.c:

Data Structures

- struct _ColorPalette

Functions

- static int count_non_empty_lines_in_array (const char *data, size_t length)
  Return the number of non empty lines in array.
- static int color_palette_fill_with_resource (ColorPalette *palette, char *resource_name)
  color_palette_fill_with_resource
- ColorPalette * color_palette_new_from_resource (char *resource_name)
  Create a new object with from a resource containing the html hex color scheme.
- GdkRGBA * color_palette_get_color (ColorPalette *palette, GdkRGBA *color, unsigned int index)
  Get the n-th color in the palette identified by the index.
- unsigned int color_palette_get_color_count (ColorPalette *palette)
  Return amount of stored colors in `palette`.
- static void color_palette_dispose (GObject *gobj)
- static void color_palette_class_init (ColorPaletteClass *klass)
- static void color_palette_init (ColorPalette *self)

13.76.1 Detailed Description

Class representing a color palette.

Author

Mario Hüttel `mario.huettel@gmx.net`

Definition in file color-palette.c.

13.76.2 Function Documentation

13.76.2.1 color_palette_class_init()

```
static void color_palette_class_init (
            ColorPaletteClass * klass )  [static]
```

Definition at line 248 of file color-palette.c.

Here is the call graph for this function:

13.76.2.2 color_palette_dispose()

```
static void color_palette_dispose (
            GObject * gobj )  [static]
```

Definition at line 233 of file color-palette.c.

Here is the caller graph for this function:

13.76.2.3 color_palette_fill_with_resource()

```
static int color_palette_fill_with_resource (
            ColorPalette * palette,
            char * resource_name )  [static]
```

color_palette_fill_with_resource

Parameters

palette	
resource_name	

Returns

0 if successful

Definition at line 84 of file color-palette.c.

Here is the call graph for this function: Here is the caller graph for this function:

13.76.2.4 color_palette_get_color()

```
GdkRGBA * color_palette_get_color (
            ColorPalette * palette,
            GdkRGBA * color,
            unsigned int index )
```

Get the n-th color in the palette identified by the index.

This function fills the nth color into the supplied `color`. `color` is returned.

If `color` is NULL, a new GdkRGBA is created and returned. This element must be freed afterwards.

Parameters

palette	Color palette
color	GdkRGBA struct to fill data in. May be NULL.
index	Index of color. Starts at 0

Returns

GdkRGBA color. If `color` is NULL, the returned color must be freed afterwards

Definition at line 199 of file color-palette.c.

Here is the caller graph for this function:

13.76.2.5 color_palette_get_color_count()

```
unsigned int color_palette_get_color_count (
            ColorPalette * palette )
```

Return amount of stored colors in `palette`.

Parameters

palette	Color palette

Returns

Count of colors

Definition at line 223 of file color-palette.c.

Here is the caller graph for this function:

13.76.2.6 color_palette_init()

```
static void color_palette_init (
            ColorPalette * self )  [static]
```

Definition at line 256 of file color-palette.c.

13.76.2.7 color_palette_new_from_resource()

```
ColorPalette * color_palette_new_from_resource (
            char * resource_name )
```

Create a new object with from a resource containing the html hex color scheme.

Parameters

resource_name	Name of the resource

Returns

New object

Definition at line 188 of file color-palette.c.

Here is the call graph for this function: Here is the caller graph for this function:

13.76.2.8 count_non_empty_lines_in_array()

```
static int count_non_empty_lines_in_array (
            const char * data,
            size_t length )  [static]
```

Return the number of non empty lines in array.

This function returns the number of non empty lines in an array. The scanning is either terminated by the given length or if a \0 terminator is found.

Parameters

in	*data*	Array to count lines in
in	*length*	Length of `data`

Returns

< 0: Error, >=0: Lines

Definition at line 55 of file color-palette.c.

Here is the caller graph for this function:

13.77 color-palette.c

Go to the documentation of this file.

```
00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2019  Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter.  If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00026 #include <gds-render/layer/color-palette.h>
00027
00028 struct _ColorPalette {
00029                 /* Inheritance */
00030                 GObject parent;
00031
```

```
00032                 /* Custom fields */
00034                 GdkRGBA *color_array;
00036                 unsigned int color_array_length;
00037
00038                 /* Dummy bytes to ensure ABI compatibility in future versions */
00039                 gpointer dummy[4];
00040 };
00041
00042 G_DEFINE_TYPE(ColorPalette, color_palette, G_TYPE_OBJECT)
00043
00044
00055 static int count_non_empty_lines_in_array(const char *data, size_t length)
00056 {
00057         unsigned int idx;
00058         int non_empty_lines = 0;
00059         char last_char = '\n';
00060
00061         if (!data)
00062                 return -1;
00063
00064         /* Count each '\n' as a new line if it is not directly preceded by another '\n' */
00065         for (idx = 0; idx < length && data[idx]; idx++) {
00066                 if (data[idx] == '\n' && last_char != '\n')
00067                         non_empty_lines++;
00068                 last_char = data[idx];
00069         }
00070
00071         /* Count the last line in case the data does not end with a '\n' */
00072         if (data[idx-1] != '\n')
00073                 non_empty_lines++;
00074
00075         return non_empty_lines;
00076 }
00077
00084 static int color_palette_fill_with_resource(ColorPalette *palette, char *resource_name)
00085 {
00086         GBytes *data;
00087         char line[10];
00088         int line_idx;
00089         unsigned int color_idx;
00090         int idx;
00091         const char *char_array;
00092         gsize byte_count;
00093         int lines;
00094         GRegex *regex;
00095         GMatchInfo *mi;
00096         char *match;
00097
00098         if (!palette || !resource_name)
00099                 return -1;
00100
00101         data = g_resources_lookup_data(resource_name, 0, NULL);
00102
00103         if (!data)
00104                 return -2;
00105
00106         char_array = (const char *)g_bytes_get_data(data, &byte_count);
00107
00108         if (!char_array || !byte_count)
00109                 goto ret_unref_data;
00110
00111         /* Get maximum lenght of color palette, assuming all entries are valid */
00112         lines = count_non_empty_lines_in_array(char_array, byte_count);
00113
00114         if (lines <= 0)
00115                 goto ret_unref_data;
00116
00117         palette->color_array = (GdkRGBA *)malloc(sizeof(GdkRGBA) * (unsigned int)lines);
00118
00119         /* Setup regex for hexadecimal RGB colors like 'A0CB3F' */
00120         regex =
    g_regex_new("^(?<red>[0-9A-Fa-f][0-9A-Fa-f])(?<green>[0-9A-Fa-f][0-9A-Fa-f])(?<blue>[0-9A-Fa-f][0-9A-Fa-f])$",
00121                                 0, 0, NULL);
00122
00123         /* Reset line */
00124         line_idx = 0;
00125         line[0] = '\0';
00126
00127         /* Set color index */
00128         color_idx = 0;
00129
00130         /* interate over lines and match */
00131         for (idx = 0 ; (unsigned int)idx < byte_count; idx++) {
00132                 /* Fillup line. */
00133                 line[line_idx] = char_array[idx];
00134
00135                 /* If end of line/string is reached, process */
```

```
00136                 if (line[line_idx] == '\n' || line[line_idx] == '\0') {
00137                         line[line_idx] = '\0';
00138 
00139                         /* Match the line */
00140                         g_regex_match(regex, line, 0, &mi);
00141                         if (g_match_info_matches(mi) && color_idx < (unsigned int)lines) {
00142                                 match = g_match_info_fetch_named(mi, "red");
00143                                 palette->color_array[color_idx].red =
00144                                                 (double)g_ascii_strtoll(match, NULL, 16) / 255.0;
00145                                 g_free(match);
00146                                 match = g_match_info_fetch_named(mi, "green");
00147                                 palette->color_array[color_idx].green =
00148                                                 (double)g_ascii_strtoll(match, NULL, 16) / 255.0;
00149                                 g_free(match);
00150                                 match = g_match_info_fetch_named(mi, "blue");
00151                                 palette->color_array[color_idx].blue =
00152                                                 (double)g_ascii_strtoll(match, NULL, 16) / 255.0;
00153                                 g_free(match);
00154 
00155                                 /* Only RGB supported so far. Fix alpha channel to 1.0 */
00156                                 palette->color_array[color_idx].alpha = 1.0;
00157 
00158                                 color_idx++;
00159                         }
00160 
00161                         g_match_info_free(mi);
00162 
00163                         /* End of string */
00164                         if (char_array[idx] == '\0')
00165                                 break;
00166 
00167                         line_idx = 0;
00168                         continue;
00169                 }
00170 
00171                 /* increment line index. If end is reached write all bytes  to the line end.
00172                  * Line is longer than required for parsing. This ensures, that everything works as
      expected
00173                  */
00174                 line_idx += ((unsigned int)line_idx < sizeof(line)-1 ? 1 : 0);
00175         }
00176 
00177         /* Data read; Shrink array in case of invalid lines */
00178         palette->color_array = realloc(palette->color_array, (size_t)color_idx * sizeof(GdkRGBA));
00179         palette->color_array_length = color_idx;
00180 
00181         g_regex_unref(regex);
00182 ret_unref_data:
00183         g_bytes_unref(data);
00184 
00185         return 0;
00186 }
00187 
00188 ColorPalette *color_palette_new_from_resource(char *resource_name)
00189 {
00190         ColorPalette *palette;
00191 
00192         palette = GDS_RENDER_COLOR_PALETTE(g_object_new(TYPE_GDS_RENDER_COLOR_PALETTE, NULL));
00193         if (palette)
00194                 (void)color_palette_fill_with_resource(palette, resource_name);
00195 
00196         return palette;
00197 }
00198 
00199 GdkRGBA *color_palette_get_color(ColorPalette *palette, GdkRGBA *color, unsigned int index)
00200 {
00201         GdkRGBA *c = NULL;
00202 
00203         if (!palette)
00204                 goto ret_c;
00205 
00206         if (index >= palette->color_array_length)
00207                 goto ret_c;
00208 
00209         if (color)
00210                 c = color;
00211         else
00212                 c = (GdkRGBA *)malloc(sizeof(GdkRGBA));
00213 
00214         /* Copy color */
00215         c->red = palette->color_array[index].red;
00216         c->green = palette->color_array[index].green;
00217         c->blue = palette->color_array[index].blue;
00218         c->alpha = palette->color_array[index].alpha;
00219 ret_c:
00220         return c;
00221 }
```

```
00222
00223 unsigned int color_palette_get_color_count(ColorPalette *palette)
00224 {
00225         unsigned int return_val = 0;
00226
00227         if (palette)
00228                 return_val = palette->color_array_length;
00229
00230         return return_val;
00231 }
00232
00233 static void color_palette_dispose(GObject *gobj)
00234 {
00235         ColorPalette *palette;
00236
00237         palette = GDS_RENDER_COLOR_PALETTE(gobj);
00238         if (palette->color_array) {
00239                 palette->color_array_length = 0;
00240                 free(palette->color_array);
00241                 palette->color_array = NULL;
00242         }
00243
00244         /* Chain up to parent class */
00245         G_OBJECT_CLASS(color_palette_parent_class)->dispose(gobj);
00246 }
00247
00248 static void color_palette_class_init(ColorPaletteClass *klass)
00249 {
00250         GObjectClass *gclass;
00251
00252         gclass = G_OBJECT_CLASS(klass);
00253         gclass->dispose = color_palette_dispose;
00254 }
00255
00256 static void color_palette_init(ColorPalette *self)
00257 {
00258         self->color_array = NULL;
00259         self->color_array_length = 0;
00260 }
```

13.78 layer-selector.c File Reference

Implementation of the layer selector.

```
#include <glib.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <gds-render/layer/layer-selector.h>
#include <gds-render/gds-utils/gds-parser.h>
#include <gds-render/widgets/layer-element.h>
```

Include dependency graph for layer-selector.c:

Data Structures

- struct _LayerSelector

Functions

- static void sel_layer_element_drag_begin (GtkWidget *widget, GdkDragContext *context, gpointer data)
- static void sel_layer_element_drag_end (GtkWidget *widget, GdkDragContext *context, gpointer data)
- static void sel_layer_element_drag_data_get (GtkWidget *widget, GdkDragContext *context, GtkSelection↩ Data *selection_data, guint info, guint time, gpointer data)
- static GtkListBoxRow * layer_selector_get_last_row (GtkListBox *list)
- static GtkListBoxRow * layer_selector_get_row_before (GtkListBox *list, GtkListBoxRow *row)

- static GtkListBoxRow * layer_selector_get_row_after (GtkListBox *list, GtkListBoxRow *row)
- static void layer_selector_drag_data_received (GtkWidget *widget, GdkDragContext *context, gint x, gint y, GtkSelectionData *selection_data, guint info, guint32 time, gpointer data)
- static gboolean layer_selector_drag_motion (GtkWidget *widget, GdkDragContext *context, int x, int y, guint time)
- static void layer_selector_drag_leave (GtkWidget *widget, GdkDragContext *context, guint time)
- static void layer_selector_dispose (GObject *self)
- static void layer_selector_class_init (LayerSelectorClass *klass)
- static void layer_selector_setup_dnd (LayerSelector *self)
- static void layer_selector_init (LayerSelector *self)
- LayerSelector * layer_selector_new (GtkListBox *list_box)

  layer_selector_new
- LayerSettings * layer_selector_export_rendered_layer_info (LayerSelector *selector)

  Get a list of all layers that shall be exported when rendering the cells.
- static void layer_selector_clear_widgets (LayerSelector *self)
- static gboolean layer_selector_check_if_layer_widget_exists (LayerSelector *self, int layer)

  Check if a specific layer element with the given layer number is present in the layer selector.
- static void sel_layer_element_setup_dnd_callbacks (LayerSelector *self, LayerElement *element)

  Setup the necessary drag and drop callbacks of layer elements.
- static void layer_selector_analyze_cell_layers (LayerSelector *self, struct gds_cell *cell)

  Analyze `cell` layers and append detected layers to layer selector `self`.
- static gint layer_selector_sort_func (GtkListBoxRow *row1, GtkListBoxRow *row2, gpointer unused)

  sort_func Sort callback for list box
- void layer_selector_generate_layer_widgets (LayerSelector *selector, GList *libs)

  Generate layer widgets in in the LayerSelector instance.
- static LayerElement * layer_selector_find_layer_element_in_list (GList *el_list, int layer)

  Find LayerElement in list with specified layer number.
- static void layer_selector_load_layer_mapping_from_file (LayerSelector *self, const gchar *file_name)

  Load the layer mapping from a CSV formatted file.
- static void layer_selector_load_mapping_clicked (GtkWidget *button, gpointer user_data)

  Callback for Load Mapping Button.
- static void layer_selector_save_layer_mapping_data (LayerSelector *self, const gchar *file_name)

  Save layer mapping of selector `self` to a file.
- static void layer_selector_save_mapping_clicked (GtkWidget *button, gpointer user_data)

  Callback for Save Layer Mapping Button.
- void layer_selector_set_load_mapping_button (LayerSelector *selector, GtkWidget *button, GtkWindow *main_window)

  Supply button for loading the layer mapping.
- void layer_selector_set_save_mapping_button (LayerSelector *selector, GtkWidget *button, GtkWindow *main_window)

  Supply button for saving the layer mapping.
- void layer_selector_force_sort (LayerSelector *selector, enum layer_selector_sort_algo sort_function)

  Force the layer selector list to be sorted according to `sort_function`.
- void layer_selector_select_all_layers (LayerSelector *layer_selector, gboolean select)

  Set 'export' value of all layers in the LayerSelector to the supplied select value.
- void layer_selector_auto_color_layers (LayerSelector *layer_selector, ColorPalette *palette, double global_alpha)

  Apply colors from palette to all layers. Aditionally set alpha.
- void layer_selector_auto_name_layers (LayerSelector *layer_selector, gboolean overwrite)

  Auto name all layers in the layer selector.
- gboolean layer_selector_contains_elements (LayerSelector *layer_selector)

  Check if the given layer selector contains layer elements.
- size_t layer_selector_num_of_named_elements (LayerSelector *layer_selector)

  Get number of layer elements that are named.

Variables

- static const char * dnd_additional_css

13.78.1 Detailed Description

Implementation of the layer selector.

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file layer-selector.c.

13.79 layer-selector.c

Go to the documentation of this file.

```
00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018  Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter.  If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00031 #include <glib.h>
00032 #include <string.h>
00033 #include <stdio.h>
00034 #include <stdlib.h>
00035
00036 #include <gds-render/layer/layer-selector.h>
00037 #include <gds-render/gds-utils/gds-parser.h>
00038 #include <gds-render/widgets/layer-element.h>
00039
00040 struct _LayerSelector {
00041         /* Parent */
00042         GObject parent;
00043         /* Own fields */
00044         GtkWidget *associated_load_button;
00045         GtkWidget *associated_save_button;
00046         GtkWindow *load_parent_window;
00047         GtkWindow *save_parent_window;
00048         GtkListBox *list_box;
00049
00050         GtkTargetEntry dnd_target;
00051
00052         gpointer dummy[4];
00053 };
00054
00055 G_DEFINE_TYPE(LayerSelector, layer_selector, G_TYPE_OBJECT)
00056
00057 /*
00058  * Drag and drop code
00059  * Original code from https://blog.gtk.org/2017/06/01/drag-and-drop-in-lists-revisited/
00060  */
00061
00062 static void sel_layer_element_drag_begin(GtkWidget *widget, GdkDragContext *context, gpointer data)
00063 {
00064         GtkWidget *row;
00065         GtkAllocation alloc;
```

```
00066         cairo_surface_t *surface;
00067         cairo_t *cr;
00068         int x, y;
00069         (void)data;
00070 
00071         row = gtk_widget_get_ancestor(widget, GTK_TYPE_LIST_BOX_ROW);
00072         gtk_widget_get_allocation(row, &alloc);
00073         surface = cairo_image_surface_create(CAIRO_FORMAT_ARGB32, alloc.width, alloc.height);
00074         cr = cairo_create(surface);
00075 
00076         gtk_style_context_add_class(gtk_widget_get_style_context(row), "drag-icon");
00077         gtk_widget_draw(row, cr);
00078         gtk_style_context_remove_class(gtk_widget_get_style_context(row), "drag-icon");
00079 
00080         gtk_widget_translate_coordinates(widget, row, 0, 0, &x, &y);
00081         cairo_surface_set_device_offset(surface, -x, -y);
00082         gtk_drag_set_icon_surface(context, surface);
00083 
00084         cairo_destroy(cr);
00085         cairo_surface_destroy(surface);
00086 
00087         g_object_set_data(G_OBJECT(gtk_widget_get_parent(row)), "drag-row", row);
00088         gtk_style_context_add_class(gtk_widget_get_style_context(row), "drag-row");
00089 }
00090 
00091 static void sel_layer_element_drag_end(GtkWidget *widget, GdkDragContext *context, gpointer data)
00092 {
00093         GtkWidget *row;
00094         (void)context;
00095         (void)data;
00096 
00097         row = gtk_widget_get_ancestor(widget, GTK_TYPE_LIST_BOX_ROW);
00098         g_object_set_data(G_OBJECT(gtk_widget_get_parent(row)), "drag-row", NULL);
00099         gtk_style_context_remove_class(gtk_widget_get_style_context(row), "drag-row");
00100         gtk_style_context_remove_class(gtk_widget_get_style_context(row), "drag-hover");
00101 }
00102 
00103 static void sel_layer_element_drag_data_get(GtkWidget *widget, GdkDragContext *context,
00104                                             GtkSelectionData *selection_data,
00105                                             guint info, guint time, gpointer data)
00106 {
00107         (void)context;
00108         (void)info;
00109         (void)time;
00110         (void)data;
00111         GdkAtom atom;
00112 
00113         atom = gdk_atom_intern_static_string("GTK_LIST_BOX_ROW");
00114 
00115         gtk_selection_data_set(selection_data, atom,
00116                                32, (const guchar *)&widget, sizeof(gpointer));
00117 }
00118 
00119 static GtkListBoxRow *layer_selector_get_last_row(GtkListBox *list)
00120 {
00121         int i;
00122         GtkListBoxRow *row;
00123         GtkListBoxRow *tmp;
00124 
00125         row = NULL;
00126         for (i = 0; ; i++) {
00127                 tmp = gtk_list_box_get_row_at_index(list, i);
00128                 if (tmp == NULL)
00129                         break;
00130                 row = tmp;
00131         }
00132 
00133         return row;
00134 }
00135 
00136 static GtkListBoxRow *layer_selector_get_row_before(GtkListBox *list, GtkListBoxRow *row)
00137 {
00138         int pos;
00139 
00140         pos = gtk_list_box_row_get_index(row);
00141         return gtk_list_box_get_row_at_index(list, pos - 1);
00142 }
00143 
00144 static GtkListBoxRow *layer_selector_get_row_after(GtkListBox *list, GtkListBoxRow *row)
00145 {
00146         int pos;
00147 
00148         pos = gtk_list_box_row_get_index(row);
00149         return gtk_list_box_get_row_at_index(list, pos + 1);
00150 }
00151 
00152 static void layer_selector_drag_data_received(GtkWidget *widget, GdkDragContext *context, gint x, gint
```

```
      y,
00153                                           GtkSelectionData *selection_data, guint info, guint32
      time,
00154                                           gpointer data)
00155 {
00156         GtkWidget *row_before, *row_after;
00157         GtkWidget *row;
00158         GtkWidget *source;
00159         int pos;
00160 
00161         /* Handle unused parameters */
00162         (void)context;
00163         (void)x;
00164         (void)y;
00165         (void)info;
00166         (void)time;
00167         (void)data;
00168 
00169         row_before = GTK_WIDGET(g_object_get_data(G_OBJECT(widget), "row-before"));
00170         row_after = GTK_WIDGET(g_object_get_data(G_OBJECT(widget), "row-after"));
00171 
00172         g_object_set_data(G_OBJECT(widget), "row-before", NULL);
00173         g_object_set_data(G_OBJECT(widget), "row-after", NULL);
00174 
00175         if (row_before)
00176                 gtk_style_context_remove_class(gtk_widget_get_style_context(row_before),
      "drag-hover-bottom");
00177         if (row_after)
00178                 gtk_style_context_remove_class(gtk_widget_get_style_context(row_after),
      "drag-hover-top");
00179 
00180         row = (gpointer) *((gpointer *)gtk_selection_data_get_data(selection_data));
00181         source = gtk_widget_get_ancestor(row, GTK_TYPE_LIST_BOX_ROW);
00182 
00183         if (source == row_after)
00184                 return;
00185 
00186         g_object_ref(source);
00187         gtk_container_remove(GTK_CONTAINER(gtk_widget_get_parent(source)), source);
00188 
00189         if (row_after)
00190                 pos = gtk_list_box_row_get_index(GTK_LIST_BOX_ROW(row_after));
00191         else
00192                 pos = gtk_list_box_row_get_index(GTK_LIST_BOX_ROW(row_before)) + 1;
00193 
00194         gtk_list_box_insert(GTK_LIST_BOX(widget), source, pos);
00195         g_object_unref(source);
00196 }
00197 
00198 static gboolean layer_selector_drag_motion(GtkWidget *widget, GdkDragContext *context, int x, int y,
      guint time)
00199 {
00200         GtkAllocation alloc;
00201         GtkWidget *row;
00202         int hover_row_y;
00203         int hover_row_height;
00204         GtkWidget *drag_row;
00205         GtkWidget *row_before;
00206         GtkWidget *row_after;
00207         (void)context;
00208         (void)x;
00209         (void)y;
00210         (void)time;
00211 
00212         row = GTK_WIDGET(gtk_list_box_get_row_at_y(GTK_LIST_BOX(widget), y));
00213 
00214         drag_row = GTK_WIDGET(g_object_get_data(G_OBJECT(widget), "drag-row"));
00215         row_after = GTK_WIDGET(g_object_get_data(G_OBJECT(widget), "row-after"));
00216         row_before = GTK_WIDGET(g_object_get_data(G_OBJECT(widget), "row-before"));
00217 
00218         gtk_style_context_remove_class(gtk_widget_get_style_context(drag_row), "drag-hover");
00219         if (row_before)
00220                 gtk_style_context_remove_class(gtk_widget_get_style_context(row_before),
      "drag-hover-bottom");
00221         if (row_after)
00222                 gtk_style_context_remove_class(gtk_widget_get_style_context(row_after),
      "drag-hover-top");
00223 
00224         if (row) {
00225                 gtk_widget_get_allocation(row, &alloc);
00226                 hover_row_y = alloc.y;
00227                 hover_row_height = alloc.height;
00228 
00229                 if (y < hover_row_y + hover_row_height/2) {
00230                         row_after = row;
00231                         row_before = GTK_WIDGET(layer_selector_get_row_before(GTK_LIST_BOX(widget),
00232                                                                               GTK_LIST_BOX_ROW(row)));
```

```
00233                 } else {
00234                         row_before = row;
00235                         row_after = GTK_WIDGET(layer_selector_get_row_after(GTK_LIST_BOX(widget),
00236                                                                             GTK_LIST_BOX_ROW(row)));
00237                 }
00238         } else {
00239                 row_before = GTK_WIDGET(layer_selector_get_last_row(GTK_LIST_BOX(widget)));
00240                 row_after = NULL;
00241         }
00242
00243         g_object_set_data(G_OBJECT(widget), "row-before", row_before);
00244         g_object_set_data(G_OBJECT(widget), "row-after", row_after);
00245
00246         if (drag_row == row_before || drag_row == row_after) {
00247                 gtk_style_context_add_class(gtk_widget_get_style_context(drag_row), "drag-hover");
00248                 return FALSE;
00249         }
00250
00251         if (row_before)
00252                 gtk_style_context_add_class(gtk_widget_get_style_context(row_before),
      "drag-hover-bottom");
00253         if (row_after)
00254                 gtk_style_context_add_class(gtk_widget_get_style_context(row_after),
      "drag-hover-top");
00255
00256         return TRUE;
00257 }
00258
00259 static void layer_selector_drag_leave(GtkWidget *widget, GdkDragContext *context, guint time)
00260 {
00261         GtkWidget *drag_row;
00262         GtkWidget *row_before;
00263         GtkWidget *row_after;
00264         (void)context;
00265         (void)time;
00266
00267         drag_row = GTK_WIDGET(g_object_get_data(G_OBJECT(widget), "drag-row"));
00268         row_before = GTK_WIDGET(g_object_get_data(G_OBJECT(widget), "row-before"));
00269         row_after = GTK_WIDGET(g_object_get_data(G_OBJECT(widget), "row-after"));
00270
00271         gtk_style_context_remove_class(gtk_widget_get_style_context(drag_row), "drag-hover");
00272         if (row_before)
00273                 gtk_style_context_remove_class(gtk_widget_get_style_context(row_before),
      "drag-hover-bottom");
00274         if (row_after)
00275                 gtk_style_context_remove_class(gtk_widget_get_style_context(row_after),
      "drag-hover-top");
00276
00277 }
00278
00279 static const char *dnd_additional_css =
00280         ".row:not(:first-child) { "
00281         "  border-top: 1px solid alpha(gray,0.5); "
00282         "  border-bottom: 1px solid transparent; "
00283         "}"
00284         ".row:first-child { "
00285         "  border-top: 1px solid transparent; "
00286         "  border-bottom: 1px solid transparent; "
00287         "}"
00288         ".row:last-child { "
00289         "  border-top: 1px solid alpha(gray,0.5); "
00290         "  border-bottom: 1px solid alpha(gray,0.5); "
00291         "}"
00292         ".row.drag-icon { "
00293         "  background: #282828; "
00294         "  border: 1px solid blue; "
00295         "}"
00296         ".row.drag-row { "
00297         "  color: gray; "
00298         "  background: alpha(gray,0.2); "
00299         "}"
00300         ".row.drag-row.drag-hover { "
00301         "  border-top: 1px solid #4e9a06; "
00302         "  border-bottom: 1px solid #4e9a06; "
00303         "}"
00304         ".row.drag-hover image, "
00305         ".row.drag-hover label { "
00306         "  color: #4e9a06; "
00307         "}"
00308         ".row.drag-hover-top {"
00309         "  border-top: 1px solid #4e9a06; "
00310         "}"
00311         ".row.drag-hover-bottom {"
00312         "  border-bottom: 1px solid #4e9a06; "
00313         "}";
00314
00315 static void layer_selector_dispose(GObject *self)
```

```
00316 {
00317         LayerSelector *sel = LAYER_SELECTOR(self);
00318 
00319         g_clear_object(&sel->list_box);
00320         g_clear_object(&sel->load_parent_window);
00321         g_clear_object(&sel->save_parent_window);
00322         g_clear_object(&sel->associated_load_button);
00323         g_clear_object(&sel->associated_save_button);
00324 
00325         if (sel->dnd_target.target) {
00326                 g_free(sel->dnd_target.target);
00327                 sel->dnd_target.target = NULL;
00328         }
00329 
00330         /* Chain up to parent's dispose function */
00331         G_OBJECT_CLASS(layer_selector_parent_class)->dispose(self);
00332 }
00333 
00334 static void layer_selector_class_init(LayerSelectorClass *klass)
00335 {
00336         GObjectClass *object_class = G_OBJECT_CLASS(klass);
00337         GtkCssProvider *provider;
00338 
00339         /* Implement handles to virtual functions */
00340         object_class->dispose = layer_selector_dispose;
00341 
00342         /* Setup the CSS provider for the drag and drop animations once */
00343         provider = gtk_css_provider_new();
00344         gtk_css_provider_load_from_data(provider, dnd_additional_css, -1, NULL);
00345         gtk_style_context_add_provider_for_screen(gdk_screen_get_default(),
      GTK_STYLE_PROVIDER(provider), 800);
00346 
00347         g_object_unref(provider);
00348 }
00349 
00350 static void layer_selector_setup_dnd(LayerSelector *self)
00351 {
00352         gtk_drag_dest_set(GTK_WIDGET(self->list_box), GTK_DEST_DEFAULT_MOTION | GTK_DEST_DEFAULT_DROP,
00353                           &self->dnd_target, 1, GDK_ACTION_MOVE);
00354         g_signal_connect(self->list_box, "drag-data-received",
      G_CALLBACK(layer_selector_drag_data_received), NULL);
00355         g_signal_connect(self->list_box, "drag-motion", G_CALLBACK(layer_selector_drag_motion), NULL);
00356         g_signal_connect(self->list_box, "drag-leave", G_CALLBACK(layer_selector_drag_leave), NULL);
00357 }
00358 
00359 /* Drag and drop end */
00360 
00361 static void layer_selector_init(LayerSelector *self)
00362 {
00363         self->load_parent_window = NULL;
00364         self->save_parent_window = NULL;
00365         self->associated_load_button = NULL;
00366         self->associated_save_button = NULL;
00367 
00368         self->dnd_target.target = g_strdup_printf("LAYER_SELECTOR_DND_%p", self);
00369         self->dnd_target.info = 0;
00370         self->dnd_target.flags = GTK_TARGET_SAME_APP;
00371 }
00372 
00373 LayerSelector *layer_selector_new(GtkListBox *list_box)
00374 {
00375         LayerSelector *selector;
00376 
00377         if (GTK_IS_LIST_BOX(list_box) == FALSE)
00378                 return NULL;
00379 
00380         selector = LAYER_SELECTOR(g_object_new(TYPE_LAYER_SELECTOR, NULL));
00381         selector->list_box = list_box;
00382         layer_selector_setup_dnd(selector);
00383         g_object_ref(G_OBJECT(list_box));
00384 
00385         return selector;
00386 }
00387 
00388 LayerSettings *layer_selector_export_rendered_layer_info(LayerSelector *selector)
00389 {
00390         LayerSettings *layer_settings;
00391         struct layer_info linfo;
00392         GList *row_list;
00393         GList *iterator;
00394         LayerElement *le;
00395         int i;
00396 
00397         layer_settings = layer_settings_new();
00398         if (!layer_settings)
00399                 return NULL;
00400 
```

```
00401         row_list = gtk_container_get_children(GTK_CONTAINER(selector->list_box));
00402
00403         for (i = 0, iterator = row_list; iterator != NULL; iterator = g_list_next(iterator), i++) {
00404                 le = LAYER_ELEMENT(iterator->data);
00405
00406                 /* Get name from layer element. This must not be freed */
00407                 linfo.name = (char *)layer_element_get_name(le);
00408
00409                 layer_element_get_color(le, &linfo.color);
00410                 linfo.render = (layer_element_get_export(le) ? 1 : 0);
00411                 linfo.stacked_position = i;
00412                 linfo.layer = layer_element_get_layer(le);
00413
00414                 /* This function copies the entire layer info struct including the name string.
00415                  * Therefore, using the same layer_info struct over and over is safe.
00416                  */
00417                 layer_settings_append_layer_info(layer_settings, &linfo);
00418         }
00419
00420         return layer_settings;
00421 }
00422
00423 static void layer_selector_clear_widgets(LayerSelector *self)
00424 {
00425         GList *list;
00426         GList *temp;
00427
00428         list = gtk_container_get_children(GTK_CONTAINER(self->list_box));
00429         for (temp = list; temp != NULL; temp = temp->next)
00430                 gtk_container_remove(GTK_CONTAINER(self->list_box), GTK_WIDGET(temp->data));
00431
00432         /* Widgets are already destroyed when removed from box because they are only referenced inside
      the container */
00433
00434         g_list_free(list);
00435
00436         /* Deactivate buttons */
00437         if (self->associated_load_button)
00438                 gtk_widget_set_sensitive(self->associated_load_button, FALSE);
00439         if (self->associated_save_button)
00440                 gtk_widget_set_sensitive(self->associated_save_button, FALSE);
00441 }
00442
00449 static gboolean layer_selector_check_if_layer_widget_exists(LayerSelector *self, int layer)
00450 {
00451         GList *list;
00452         GList *temp;
00453         LayerElement *widget;
00454         gboolean ret = FALSE;
00455
00456         list = gtk_container_get_children(GTK_CONTAINER(self->list_box));
00457
00458         for (temp = list; temp != NULL; temp = temp->next) {
00459                 widget = LAYER_ELEMENT(temp->data);
00460                 if (layer_element_get_layer(widget) == layer) {
00461                         ret = TRUE;
00462                         break;
00463                 }
00464         }
00465
00466         g_list_free(list);
00467
00468         return ret;
00469 }
00470
00476 static void sel_layer_element_setup_dnd_callbacks(LayerSelector *self, LayerElement *element)
00477 {
00478         struct layer_element_dnd_data dnd_data;
00479
00480         if (!self || !element)
00481                 return;
00482
00483         dnd_data.entries = &self->dnd_target;
00484         dnd_data.entry_count = 1;
00485         dnd_data.drag_end = sel_layer_element_drag_end;
00486         dnd_data.drag_begin = sel_layer_element_drag_begin;
00487         dnd_data.drag_data_get = sel_layer_element_drag_data_get;
00488
00489         layer_element_set_dnd_callbacks(element, &dnd_data);
00490 }
00491
00497 static void layer_selector_analyze_cell_layers(LayerSelector *self, struct gds_cell *cell)
00498 {
00499         GList *graphics;
00500         struct gds_graphics *gfx;
00501         int layer;
00502         GtkWidget *le;
```

```
00503
00504         for (graphics = cell->graphic_objs; graphics != NULL; graphics = graphics->next) {
00505                 gfx = (struct gds_graphics *)graphics->data;
00506                 layer = (int)gfx->layer;
00507                 if (layer_selector_check_if_layer_widget_exists(self, layer) == FALSE) {
00508                         le = layer_element_new();
00509                         sel_layer_element_setup_dnd_callbacks(self, LAYER_ELEMENT(le));
00510                         layer_element_set_layer(LAYER_ELEMENT(le), layer);
00511                         gtk_list_box_insert(self->list_box, le, -1);
00512                         gtk_widget_show(le);
00513                 }
00514         }
00515 }
00516
00525 static gint layer_selector_sort_func(GtkListBoxRow *row1, GtkListBoxRow *row2, gpointer unused)
00526 {
00527         LayerElement *le1, *le2;
00528         gint ret;
00529         static const enum layer_selector_sort_algo default_sort = LAYER_SELECTOR_SORT_DOWN;
00530         const enum layer_selector_sort_algo *algo = (const enum layer_selector_sort_algo *)unused;
00531
00532         /* Assume downward sorting */
00533         /* TODO: This is nasty. Find a better way */
00534         if (!algo)
00535                 algo = &default_sort;
00536
00537         le1 = LAYER_ELEMENT(row1);
00538         le2 = LAYER_ELEMENT(row2);
00539
00540         /* Determine sort fow downward sort */
00541         ret = layer_element_get_layer(le1) - layer_element_get_layer(le2);
00542
00543         /* Change order if upward sort is requested */
00544         ret *= (*algo == LAYER_SELECTOR_SORT_DOWN ? 1 : -1);
00545
00546         return ret;
00547 }
00548
00549 void layer_selector_generate_layer_widgets(LayerSelector *selector, GList *libs)
00550 {
00551         GList *cell_list = NULL;
00552         struct gds_library *lib;
00553
00554         layer_selector_clear_widgets(selector);
00555
00556         for (; libs != NULL; libs = libs->next) {
00557                 lib = (struct gds_library *)libs->data;
00558                 for (cell_list = lib->cells; cell_list != NULL; cell_list = cell_list->next)
00559                         layer_selector_analyze_cell_layers(selector, (struct gds_cell
       *)cell_list->data);
00560         } /* For libs */
00561
00562         /* Sort the layers */
00563         layer_selector_force_sort(selector, LAYER_SELECTOR_SORT_DOWN);
00564
00565         /* Activate Buttons */
00566         if (selector->associated_load_button)
00567                 gtk_widget_set_sensitive(selector->associated_load_button, TRUE);
00568         if (selector->associated_save_button)
00569                 gtk_widget_set_sensitive(selector->associated_save_button, TRUE);
00570 }
00571
00578 static LayerElement *layer_selector_find_layer_element_in_list(GList *el_list, int layer)
00579 {
00580         LayerElement *ret = NULL;
00581
00582         for (; el_list != NULL; el_list = el_list->next) {
00583                 if (layer_element_get_layer(LAYER_ELEMENT(el_list->data)) == layer) {
00584                         ret = LAYER_ELEMENT(el_list->data);
00585                         break;
00586                 }
00587         }
00588         return ret;
00589 }
00590
00602 static void layer_selector_load_layer_mapping_from_file(LayerSelector *self, const gchar *file_name)
00603 {
00604         GFile *file;
00605         GFileInputStream *stream;
00606         GDataInputStream *dstream;
00607         LayerElement *le;
00608         GList *rows;
00609         GList *temp;
00610         GList *layer_infos;
00611         int status;
00612         LayerSettings *layer_settings;
00613         struct layer_info *linfo;
```

```
00614
00615         file = g_file_new_for_path(file_name);
00616         stream = g_file_read(file, NULL, NULL);
00617
00618         if (!stream)
00619                 goto destroy_file;
00620
00621         dstream = g_data_input_stream_new(G_INPUT_STREAM(stream));
00622
00623         rows = gtk_container_get_children(GTK_CONTAINER(self->list_box));
00624
00625         /* Reference and remove all rows from box */
00626         for (temp = rows; temp != NULL; temp = temp->next) {
00627                 le = LAYER_ELEMENT(temp->data);
00628                 /* Referencing protects the widget from being deleted when removed */
00629                 g_object_ref(G_OBJECT(le));
00630                 gtk_container_remove(GTK_CONTAINER(self->list_box), GTK_WIDGET(le));
00631         }
00632
00633         /* Load Layer settings. No need to check pointer, will be checked by load csv func. */
00634         layer_settings = layer_settings_new();
00635
00636         status = layer_settings_load_from_csv(layer_settings, file_name);
00637         if (status)
00638                 goto abort_layer_settings;
00639
00640         layer_infos = layer_settings_get_layer_info_list(layer_settings);
00641         if (!layer_infos)
00642                 goto abort_layer_settings;
00643
00644         /* Loop over all layer infos read from the CSV file */
00645         for (; layer_infos; layer_infos = g_list_next(layer_infos)) {
00646                 linfo = (struct layer_info *)layer_infos->data;
00647                 le = layer_selector_find_layer_element_in_list(rows, linfo->layer);
00648                 if (!le)
00649                         continue;
00650
00651                 layer_element_set_name(le, linfo->name);
00652                 layer_element_set_export(le, (linfo->render ? TRUE : FALSE));
00653                 layer_element_set_color(le, &linfo->color);
00654                 gtk_container_add(GTK_CONTAINER(self->list_box), GTK_WIDGET(le));
00655                 rows = g_list_remove(rows, le);
00656         }
00657
00658 abort_layer_settings:
00659         /* Destroy layer settings. Not needed for adding remaining elements */
00660         g_object_unref(layer_settings);
00661
00662         /* Add remaining elements */
00663         for (temp = rows; temp != NULL; temp = temp->next) {
00664                 le = LAYER_ELEMENT(temp->data);
00665                 /* Referencing protets the widget from being deleted when removed */
00666                 gtk_list_box_insert(self->list_box, GTK_WIDGET(le), -1);
00667                 g_object_unref(G_OBJECT(le));
00668         }
00669
00670         /* Delete list */
00671         g_list_free(rows);
00672
00673         /* read line */
00674         g_object_unref(dstream);
00675         g_object_unref(stream);
00676 destroy_file:
00677         g_object_unref(file);
00678 }
00679
00685 static void layer_selector_load_mapping_clicked(GtkWidget *button, gpointer user_data)
00686 {
00687         LayerSelector *sel;
00688         GtkWidget *dialog;
00689         gint res;
00690         gchar *file_name;
00691         (void)button;
00692
00693         sel = LAYER_SELECTOR(user_data);
00694
00695         dialog = gtk_file_chooser_dialog_new("Load Mapping File", GTK_WINDOW(sel->load_parent_window),
00696                                              GTK_FILE_CHOOSER_ACTION_OPEN,
00697                                              "Cancel", GTK_RESPONSE_CANCEL, "Load Mapping",
      GTK_RESPONSE_ACCEPT, NULL);
00698         res = gtk_dialog_run(GTK_DIALOG(dialog));
00699         if (res == GTK_RESPONSE_ACCEPT) {
00700                 file_name = gtk_file_chooser_get_filename(GTK_FILE_CHOOSER(dialog));
00701                 layer_selector_load_layer_mapping_from_file(sel, file_name);
00702                 g_free(file_name);
00703         }
00704         gtk_widget_destroy(dialog);
```

```
00705 }
00706
00707
00708
00714 static void layer_selector_save_layer_mapping_data(LayerSelector *self, const gchar *file_name)
00715 {
00716         LayerSettings *layer_settings;
00717
00718         g_return_if_fail(LAYER_IS_SELECTOR(self));
00719         g_return_if_fail(file_name);
00720
00721         /* Get layer settings. No need to check return value. to_csv func is safe */
00722         layer_settings = layer_selector_export_rendered_layer_info(self);
00723         (void)layer_settings_to_csv(layer_settings, file_name);
00724 }
00725
00731 static void layer_selector_save_mapping_clicked(GtkWidget *button, gpointer user_data)
00732 {
00733         GtkWidget *dialog;
00734         gint res;
00735         gchar *file_name;
00736         LayerSelector *sel;
00737         (void)button;
00738
00739         sel = LAYER_SELECTOR(user_data);
00740
00741         dialog = gtk_file_chooser_dialog_new("Save Mapping File", GTK_WINDOW(sel->save_parent_window),
00742                                              GTK_FILE_CHOOSER_ACTION_SAVE,
00743                                              "Cancel", GTK_RESPONSE_CANCEL, "Save Mapping",
      GTK_RESPONSE_ACCEPT, NULL);
00744         gtk_file_chooser_set_do_overwrite_confirmation(GTK_FILE_CHOOSER(dialog), TRUE);
00745
00746         res = gtk_dialog_run(GTK_DIALOG(dialog));
00747         if (res == GTK_RESPONSE_ACCEPT) {
00748                 file_name = gtk_file_chooser_get_filename(GTK_FILE_CHOOSER(dialog));
00749                 layer_selector_save_layer_mapping_data(sel, file_name);
00750                 g_free(file_name);
00751         }
00752         gtk_widget_destroy(dialog);
00753 }
00754
00755 void layer_selector_set_load_mapping_button(LayerSelector *selector, GtkWidget *button, GtkWindow
      *main_window)
00756 {
00757         g_clear_object(&selector->load_parent_window);
00758         g_clear_object(&selector->associated_load_button);
00759
00760         g_object_ref(G_OBJECT(button));
00761         g_object_ref(G_OBJECT(main_window));
00762         selector->associated_load_button = button;
00763         selector->load_parent_window = main_window;
00764         g_signal_connect(button, "clicked", G_CALLBACK(layer_selector_load_mapping_clicked),
      selector);
00765 }
00766
00767 void layer_selector_set_save_mapping_button(LayerSelector *selector, GtkWidget *button,  GtkWindow
      *main_window)
00768 {
00769         g_clear_object(&selector->save_parent_window);
00770         g_clear_object(&selector->associated_save_button);
00771
00772         g_object_ref(G_OBJECT(button));
00773         g_object_ref(G_OBJECT(main_window));
00774         selector->associated_save_button = button;
00775         selector->save_parent_window = main_window;
00776         g_signal_connect(button, "clicked", G_CALLBACK(layer_selector_save_mapping_clicked),
      selector);
00777 }
00778
00779 void layer_selector_force_sort(LayerSelector *selector, enum layer_selector_sort_algo sort_function)
00780 {
00781         GtkListBox *box;
00782
00783         if (!selector)
00784                 return;
00785
00786         box = selector->list_box;
00787         if (!box)
00788                 return;
00789
00790         /* Set sorting function, sort, and disable sorting function */
00791         gtk_list_box_set_sort_func(box, layer_selector_sort_func, (gpointer)&sort_function, NULL);
00792         gtk_list_box_invalidate_sort(box);
00793         gtk_list_box_set_sort_func(box, NULL, NULL, NULL);
00794 }
00795
00796 void layer_selector_select_all_layers(LayerSelector *layer_selector, gboolean select)
```

```
00797 {
00798         GList *le_list;
00799         GList *iter;
00800         LayerElement *le;
00801 
00802         g_return_if_fail(LAYER_IS_SELECTOR(layer_selector));
00803         g_return_if_fail(GTK_IS_LIST_BOX(layer_selector->list_box));
00804 
00805         le_list = gtk_container_get_children(GTK_CONTAINER(layer_selector->list_box));
00806 
00807         for (iter = le_list; iter != NULL; iter = g_list_next(iter)) {
00808                 le = LAYER_ELEMENT(iter->data);
00809                 if (LAYER_IS_ELEMENT(le))
00810                         layer_element_set_export(le, select);
00811         }
00812 
00813         g_list_free(le_list);
00814 }
00815 
00816 void layer_selector_auto_color_layers(LayerSelector *layer_selector, ColorPalette *palette, double
       global_alpha)
00817 {
00818         GList *le_list;
00819         GList *le_list_ptr;
00820         LayerElement *le;
00821         unsigned int color_index = 0;
00822         unsigned int color_count;
00823         GdkRGBA color;
00824 
00825         g_return_if_fail(GDS_RENDER_IS_COLOR_PALETTE(palette));
00826         g_return_if_fail(LAYER_IS_SELECTOR(layer_selector));
00827         g_return_if_fail(global_alpha > 0);
00828         g_return_if_fail(GTK_IS_LIST_BOX(layer_selector->list_box));
00829 
00830         le_list = gtk_container_get_children(GTK_CONTAINER(layer_selector->list_box));
00831 
00832         /* iterate over layer elements and fill colors */
00833         color_index = 0;
00834         color_count = color_palette_get_color_count(palette);
00835         if (color_count == 0)
00836                 goto ret_free_le_list;
00837 
00838         for (le_list_ptr = le_list; le_list_ptr != NULL; le_list_ptr = le_list_ptr->next) {
00839                 le = LAYER_ELEMENT(le_list_ptr->data);
00840                 if (le) {
00841                         color_palette_get_color(palette, &color, color_index++);
00842                         color.alpha *= global_alpha;
00843                         layer_element_set_color(le, &color);
00844 
00845                         if (color_index >= color_count)
00846                                 color_index = 0;
00847                 }
00848         }
00849 
00850 ret_free_le_list:
00851         g_list_free(le_list);
00852 }
00853 
00854 void layer_selector_auto_name_layers(LayerSelector *layer_selector, gboolean overwrite)
00855 {
00856         GList *le_list;
00857         GList *le_list_ptr;
00858         LayerElement *le;
00859         const char *old_layer_name;
00860         GString *new_layer_name;
00861 
00862         g_return_if_fail(LAYER_IS_SELECTOR(layer_selector));
00863 
00864         new_layer_name = g_string_new_len(NULL, 10);
00865         le_list = gtk_container_get_children(GTK_CONTAINER(layer_selector->list_box));
00866 
00867         for (le_list_ptr = le_list; le_list_ptr != NULL; le_list_ptr = g_list_next(le_list_ptr)) {
00868                 le = LAYER_ELEMENT(le_list_ptr->data);
00869                 if (!le)
00870                         continue;
00871                 old_layer_name = layer_element_get_name(le);
00872 
00873                 /* Check if layer name is empty or may be overwritten */
00874                 if (!old_layer_name || *old_layer_name == '\0' || overwrite) {
00875                         g_string_printf(new_layer_name, "Layer %d", layer_element_get_layer(le));
00876                         layer_element_set_name(le, new_layer_name->str);
00877                 }
00878         }
00879 
00880         g_string_free(new_layer_name, TRUE);
00881         g_list_free(le_list);
00882 }
```

```
00883
00884 gboolean layer_selector_contains_elements(LayerSelector *layer_selector)
00885 {
00886         GList *layer_element_list;
00887
00888         /* Check objects */
00889         g_return_val_if_fail(LAYER_IS_SELECTOR(layer_selector), FALSE);
00890         g_return_val_if_fail(GTK_IS_LIST_BOX(layer_selector->list_box), FALSE);
00891
00892         /* Get a list of the child elements inside the list boy associated with this selector */
00893         layer_element_list = gtk_container_get_children(GTK_CONTAINER(layer_selector->list_box));
00894
00895         /* Return TRUE if there is an element in the list, else return FALSE */
00896         return (layer_element_list ? TRUE : FALSE);
00897 }
00898
00899 size_t layer_selector_num_of_named_elements(LayerSelector *layer_selector)
00900 {
00901         GList *le_list;
00902         GList *le_list_ptr;
00903         LayerElement *le;
00904         const char *layer_name;
00905         size_t count = 0U;
00906
00907         g_return_val_if_fail(LAYER_IS_SELECTOR(layer_selector), 0U);
00908
00909         le_list = gtk_container_get_children(GTK_CONTAINER(layer_selector->list_box));
00910
00911         for (le_list_ptr = le_list; le_list_ptr != NULL; le_list_ptr = g_list_next(le_list_ptr)) {
00912                 le = LAYER_ELEMENT(le_list_ptr->data);
00913                 if (!le)
00914                         continue;
00915                 layer_name = layer_element_get_name(le);
00916
00917                 if (layer_name && *layer_name) {
00918                         /* Layer name is not empty. Count it */
00919                         count++;
00920                 }
00921         }
00922
00923         return count;
00924 }
00925
```

13.80 layer-settings.c File Reference

Implementation of the LayerSettings class.

```
#include <gds-render/layer/layer-settings.h>
#include <stdlib.h>
```

Include dependency graph for layer-settings.c:

Data Structures

- struct _LayerSettings

Functions

- static void layer_settings_init (LayerSettings *self)
- static void layer_info_delete_with_name (struct layer_info *const info)
- static void layer_settings_dispose (GObject *obj)
- static void layer_settings_class_init (LayerSettingsClass *klass)
- static struct layer_info * layer_info_copy (const struct layer_info *const info)

  Copy layer_info struct.
- LayerSettings * layer_settings_new ()

  New LayerSettings object.

- int layer_settings_append_layer_info (LayerSettings *settings, struct layer_info *info)
  layer_settings_append_layer_info
- void layer_settings_clear (LayerSettings *settings)
  Clear all layers in this settings object.
- int layer_settings_remove_layer (LayerSettings *settings, int layer)
  Remove a specific layer number from the layer settings.
- GList * layer_settings_get_layer_info_list (LayerSettings *settings)
  Get a GList with layer_info structs.
- static void layer_settings_gen_csv_line (GString *string, struct layer_info *linfo)
  Generate a layer mapping CSV line for a given layer_info struct.
- int layer_settings_to_csv (LayerSettings *settings, const char *path)
  Write layer settings to a CSV file.
- static int layer_settings_load_csv_line_from_stream (GDataInputStream *stream, struct layer_info *linfo)
  Load a line from `stream` and parse try to parse it as layer information.
- int layer_settings_load_from_csv (LayerSettings *settings, const char *path)
  Load new layer Settings from CSV.

13.80.1 Detailed Description

Implementation of the LayerSettings class.

Author

Mario Hüttel `mario.huettel@gmx.net`

Definition in file layer-settings.c.

13.80.2 Function Documentation

13.80.2.1 layer_info_copy()

```
static struct layer_info * layer_info_copy (
            const struct layer_info *const info )  [static]
```

Copy layer_info struct.

This function copies a layer info struct.

Note

Be aware, that it does not only copy the pointer to the layer name, but instead duplicates the string.

Parameters

info	Info to copy

Returns

new layer_info struct

Definition at line 86 of file layer-settings.c.

Here is the caller graph for this function:

13.80.2.2 layer_info_delete_with_name()

```
static void layer_info_delete_with_name (
            struct layer_info *const info )  [static]
```

Definition at line 42 of file layer-settings.c.

Here is the caller graph for this function:

13.80.2.3 layer_settings_append_layer_info()

```
int layer_settings_append_layer_info (
            LayerSettings * settings,
            struct layer_info * info )
```

layer_settings_append_layer_info

Parameters

settings	LayerSettings object.
info	Info to append

Returns

Error code. 0 if successful

Note

`info` is copied internally. You can free this struct afterwards.

Definition at line 111 of file layer-settings.c.

Here is the call graph for this function: Here is the caller graph for this function:

13.80.2.4 layer_settings_class_init()

```
static void layer_settings_class_init (
            LayerSettingsClass * klass )  [static]
```

Definition at line 66 of file layer-settings.c.

Here is the call graph for this function:

13.80.2.5 layer_settings_clear()

```
void layer_settings_clear (
            LayerSettings * settings )
```

Clear all layers in this settings object.

Parameters

settings	LayerSettings object

Definition at line 128 of file layer-settings.c.

Here is the call graph for this function: Here is the caller graph for this function:

13.80.2.6 layer_settings_dispose()

```
static void layer_settings_dispose (
            GObject * obj )  [static]
```

Definition at line 52 of file layer-settings.c.

Here is the call graph for this function: Here is the caller graph for this function:

13.80.2.7 layer_settings_gen_csv_line()

```
static void layer_settings_gen_csv_line (
            GString * string,
            struct layer_info * linfo )  [static]
```

Generate a layer mapping CSV line for a given layer_info struct.

Parameters

string	Buffer to write to
linfo	Layer information

Definition at line 177 of file layer-settings.c.

Here is the caller graph for this function:

13.80.2.8 layer_settings_get_layer_info_list()

```
GList * layer_settings_get_layer_info_list (
            LayerSettings * settings )
```

Get a GList with layer_info structs.

This function returns a GList with all layer_info structs in rendering order (bottom to top) that shall be rendered.

Parameters

settings	LayerSettings object

Returns

GList with struct layer_info elements.

Definition at line 166 of file layer-settings.c.

Here is the caller graph for this function:

13.80.2.9 layer_settings_init()

```
static void layer_settings_init (
            LayerSettings * self )  [static]
```

Definition at line 37 of file layer-settings.c.

13.80.2.10 layer_settings_load_csv_line_from_stream()

```
static int layer_settings_load_csv_line_from_stream (
            GDataInputStream * stream,
            struct layer_info * linfo )  [static]
```

Load a line from `stream` and parse try to parse it as layer information.

Parameters

stream	Input data stream
linfo	Layer info struct to fill

Returns

1 if malformatted line, 0 if parsing was successful and parameters are valid, -1 if file end

Definition at line 247 of file layer-settings.c.

Here is the caller graph for this function:

13.80.2.11 layer_settings_load_from_csv()

```
int layer_settings_load_from_csv (
            LayerSettings * settings,
            const char * path )
```

Load new layer Settings from CSV.

This function loads the layer information from a CSV file. All data inside the `settings` is cleared beforehand.

Parameters

settings	Settings to write to.
path	CSV file path

Returns

0 if successful

Definition at line 310 of file layer-settings.c.

Here is the call graph for this function: Here is the caller graph for this function:

13.80.2.12 layer_settings_new()

```
LayerSettings * layer_settings_new ( )
```

New LayerSettings object.

Returns

New object

Definition at line 106 of file layer-settings.c.

Here is the caller graph for this function:

13.80.2.13 layer_settings_remove_layer()

```
int layer_settings_remove_layer (
            LayerSettings * settings,
            int layer )
```

Remove a specific layer number from the layer settings.

Parameters

settings	LayerSettings object
layer	Layer number

Returns

Error code. 0 if successful

Definition at line 137 of file layer-settings.c.

Here is the call graph for this function:

13.80.2.14 layer_settings_to_csv()

```
int layer_settings_to_csv (
            LayerSettings * settings,
            const char * path )
```

Write layer settings to a CSV file.

This function writes the layer settings to a CSV file according to the layer mapping specification (Layer Mapping File Specification)

Parameters

settings	LayerSettings object
path	Output path for CSV file.

Returns

0 if successful

Definition at line 196 of file layer-settings.c.

Here is the call graph for this function: Here is the caller graph for this function:

13.81 layer-settings.c

Go to the documentation of this file.

```
00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2019  Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter.  If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00026 #include <gds-render/layer/layer-settings.h>
00027 #include <stdlib.h>
00028
00029 struct _LayerSettings {
00030         GObject parent;
00031         GList *layer_infos;
00032         gpointer padding[12];
00033 };
00034
00035 G_DEFINE_TYPE(LayerSettings, layer_settings, G_TYPE_OBJECT)
00036
00037 static void layer_settings_init(LayerSettings *self)
00038 {
00039         self->layer_infos = NULL;
00040 }
00041
00042 static void layer_info_delete_with_name(struct layer_info *const info)
00043 {
00044         if (!info)
00045                 return;
00046
```

```
00047         if (info->name)
00048                 free(info->name);
00049         free(info);
00050 }
00051
00052 static void layer_settings_dispose(GObject *obj)
00053 {
00054         LayerSettings *self;
00055
00056         self = GDS_RENDER_LAYER_SETTINGS(obj);
00057
00058         if (self->layer_infos) {
00059                 g_list_free_full(self->layer_infos, (GDestroyNotify)layer_info_delete_with_name);
00060                 self->layer_infos = NULL;
00061         }
00062
00063         G_OBJECT_CLASS(layer_settings_parent_class)->dispose(obj);
00064 }
00065
00066 static void layer_settings_class_init(LayerSettingsClass *klass)
00067 {
00068         GObjectClass *oclass;
00069
00070         oclass = G_OBJECT_CLASS(klass);
00071         oclass->dispose = layer_settings_dispose;
00072
00073         return;
00074 }
00075
00086 static struct layer_info *layer_info_copy(const struct layer_info * const info)
00087 {
00088         struct layer_info *copy;
00089
00090         if (!info)
00091                 return 0;
00092
00093         copy = (struct layer_info *)malloc(sizeof(struct layer_info));
00094         if (!copy)
00095                 return 0;
00096
00097         /* Copy data */
00098         memcpy(copy, info, sizeof(struct layer_info));
00099         /* Duplicate string */
00100         if (info->name)
00101                 copy->name = strdup(info->name);
00102
00103         return copy;
00104 }
00105
00106 LayerSettings *layer_settings_new()
00107 {
00108         return g_object_new(GDS_RENDER_TYPE_LAYER_SETTINGS, NULL);
00109 }
00110
00111 int layer_settings_append_layer_info(LayerSettings *settings, struct layer_info *info)
00112 {
00113         struct layer_info *info_copy;
00114
00115         g_return_val_if_fail(GDS_RENDER_IS_LAYER_SETTINGS(settings), -1);
00116         if (!info)
00117                 return -2;
00118
00119         /* Copy layer info */
00120         info_copy = layer_info_copy(info);
00121
00122         /* Append to list */
00123         settings->layer_infos = g_list_append(settings->layer_infos, info_copy);
00124
00125         return (settings->layer_infos ? 0 : -3);
00126 }
00127
00128 void layer_settings_clear(LayerSettings *settings)
00129 {
00130         g_return_if_fail(GDS_RENDER_IS_LAYER_SETTINGS(settings));
00131
00132         /* Clear list and delete layer_info structs including the name field */
00133         g_list_free_full(settings->layer_infos, (GDestroyNotify)layer_info_delete_with_name);
00134         settings->layer_infos = NULL;
00135 }
00136
00137 int layer_settings_remove_layer(LayerSettings *settings, int layer)
00138 {
00139         GList *list_iter;
00140         GList *found = NULL;
00141         struct layer_info *inf;
00142
00143         g_return_val_if_fail(GDS_RENDER_IS_LAYER_SETTINGS(settings), -1);
```

```
00144
00145         /* Find in list */
00146         for (list_iter = settings->layer_infos; list_iter; list_iter = list_iter->next) {
00147                 inf = (struct layer_info *)list_iter->data;
00148
00149                 if (!inf)
00150                         continue;
00151                 if (inf->layer == layer)
00152                         found = list_iter;
00153         }
00154
00155         if (found)  {
00156                 /* Free the layer_info struct */
00157                 layer_info_delete_with_name((struct layer_info *)found->data);
00158                 /* Delete the list element */
00159                 settings->layer_infos = g_list_delete_link(settings->layer_infos, found);
00160                 return 0;
00161         }
00162
00163         return -2;
00164 }
00165
00166 GList *layer_settings_get_layer_info_list(LayerSettings *settings)
00167 {
00168         g_return_val_if_fail(GDS_RENDER_IS_LAYER_SETTINGS(settings), NULL);
00169         return settings->layer_infos;
00170 }
00171
00177 static void layer_settings_gen_csv_line(GString *string, struct layer_info *linfo)
00178 {
00179         int i;
00180
00181         g_string_printf(string, "%d:%lf:%lf:%lf:%lf:%d:%s\n",
00182                         linfo->layer, linfo->color.red, linfo->color.green,
00183                         linfo->color.blue, linfo->color.alpha, (linfo->render ? 1 : 0), linfo->name);
00184         /* Fix broken locale settings */
00185         for (i = 0; string->str[i]; i++) {
00186                 if (string->str[i] == ',')
00187                         string->str[i] = '.';
00188         }
00189
00190         for (i = 0; string->str[i]; i++) {
00191                 if (string->str[i] == ':')
00192                         string->str[i] = ',';
00193         }
00194 }
00195
00196 int layer_settings_to_csv(LayerSettings *settings, const char *path)
00197 {
00198         GFile *file;
00199         GOutputStream *w_fstream;
00200         GString *string;
00201         GList *info_iter;
00202         struct layer_info *linfo;
00203         int ret = 0;
00204
00205         file = g_file_new_for_path(path);
00206         w_fstream = G_OUTPUT_STREAM(g_file_replace(file, NULL, FALSE, G_FILE_CREATE_NONE, NULL,
      NULL));
00207         if (!w_fstream) {
00208                 ret = -1;
00209                 goto ret_unref_file;
00210         }
00211
00212         /* Allocate new working buffer string. A size bigger than 200 is unexpected, but possible
00213          * 200 is a tradeoff between memory usage and preventing the necessity of realloc'ing the
      string
00214          */
00215         string = g_string_new_len(NULL, 200);
00216         if (!string) {
00217                 ret = -2;
00218                 goto ret_close_file;
00219         }
00220
00221         /* Loop over layers and write CSV lines */
00222         for (info_iter = settings->layer_infos; info_iter; info_iter = info_iter->next) {
00223                 linfo = (struct layer_info *)info_iter->data;
00224
00225                 layer_settings_gen_csv_line(string, linfo);
00226                 g_output_stream_write(w_fstream, string->str, string->len * sizeof(gchar), NULL,
      NULL);
00227         }
00228
00229         /* Delete string */
00230         g_string_free(string, TRUE);
00231 ret_close_file:
00232         g_output_stream_flush(w_fstream, NULL, NULL);
```

```
00233         g_output_stream_close(w_fstream, NULL, NULL);
00234         g_object_unref(w_fstream);
00235 ret_unref_file:
00236         g_object_unref(file);
00237
00238         return ret;
00239 }
00240
00247 static int layer_settings_load_csv_line_from_stream(GDataInputStream *stream, struct layer_info
       *linfo)
00248 {
00249         int ret;
00250         gsize len;
00251         gchar *line;
00252         GRegex *regex;
00253         GMatchInfo *mi;
00254         char *match;
00255
00256         if (!linfo) {
00257                 ret = 1;
00258                 goto ret_direct;
00259         }
00260
00261         regex =
       g_regex_new("^(?<layer>[0-9]+),(?<r>[0-9\\.]+),(?<g>[0-9\\.]+),(?<b>[0-9\\.]+),(?<a>[0-9\\.]+),(?<export>[01]),(?<name>
       0, 0, NULL);
00262
00263         line = g_data_input_stream_read_line(stream, &len, NULL, NULL);
00264         if (!line) {
00265                 ret = -1;
00266                 goto destroy_regex;
00267         }
00268
00269         /* Match line in CSV */
00270         g_regex_match(regex, line, 0, &mi);
00271         if (g_match_info_matches(mi)) {
00272                 /* Line is valid */
00273                 match = g_match_info_fetch_named(mi, "layer");
00274                 linfo->layer = (int)g_ascii_strtoll(match, NULL, 10);
00275                 g_free(match);
00276                 match = g_match_info_fetch_named(mi, "r");
00277                 linfo->color.red = g_ascii_strtod(match, NULL);
00278                 g_free(match);
00279                 match = g_match_info_fetch_named(mi, "g");
00280                 linfo->color.green = g_ascii_strtod(match, NULL);
00281                 g_free(match);
00282                 match = g_match_info_fetch_named(mi, "b");
00283                 linfo->color.blue = g_ascii_strtod(match, NULL);
00284                 g_free(match);
00285                 match = g_match_info_fetch_named(mi, "a");
00286                 linfo->color.alpha = g_ascii_strtod(match, NULL);
00287                 g_free(match);
00288                 match = g_match_info_fetch_named(mi, "export");
00289                 linfo->render = ((!strcmp(match, "1")) ? 1 : 0);
00290                 g_free(match);
00291                 match = g_match_info_fetch_named(mi, "name");
00292                 linfo->name = match;
00293
00294                 ret = 0;
00295         } else {
00296                 /* Line is malformatted */
00297                 printf("Could not recognize line in CSV as valid entry: %s\n", line);
00298                 ret = 1;
00299         }
00300
00301         g_match_info_free(mi);
00302         g_free(line);
00303 destroy_regex:
00304         g_regex_unref(regex);
00305 ret_direct:
00306         return ret;
00307
00308 }
00309
00310 int layer_settings_load_from_csv(LayerSettings *settings, const char *path)
00311 {
00312         GFile *file;
00313         int ret = 0;
00314         GInputStream *in_stream;
00315         GDataInputStream *data_stream;
00316         int parser_ret;
00317         int stacked_pos;
00318         struct layer_info linfo;
00319
00320         file = g_file_new_for_path(path);
00321         in_stream = G_INPUT_STREAM(g_file_read(file, NULL, NULL));
00322
```

```
00323         g_return_val_if_fail(GDS_RENDER_IS_LAYER_SETTINGS(settings), -2);
00324
00325         if (!in_stream) {
00326                 ret = -1;
00327                 goto ret_destroy_file;
00328         }
00329         /* Delete old settings */
00330         layer_settings_clear(settings);
00331
00332         data_stream = g_data_input_stream_new(in_stream);
00333
00334         stacked_pos = 0;
00335         while ((parser_ret = layer_settings_load_csv_line_from_stream(data_stream, &linfo)) >= 0) {
00336                 /* Line broken */
00337                 if (parser_ret == 1)
00338                         continue;
00339
00340                 linfo.stacked_position = stacked_pos++;
00341
00342                 layer_settings_append_layer_info(settings, &linfo);
00343                 /* Clear name to prevent memory leak */
00344                 if (linfo.name)
00345                         g_free(linfo.name);
00346         }
00347
00348         g_object_unref(data_stream);
00349         g_object_unref(in_stream);
00350 ret_destroy_file:
00351         g_object_unref(file);
00352
00353         return ret;
00354 }
```

13.82 gpl-2.0.md File Reference

13.83 main.c File Reference

main.c

```
#include <stdio.h>
#include <gtk/gtk.h>
#include <glib.h>
#include <glib/gi18n.h>
#include <locale.h>
#include <gds-render/gds-render-gui.h>
#include <gds-render/command-line.h>
#include <gds-render/output-renderers/external-renderer.h>
#include <gds-render/version.h>
```

Include dependency graph for main.c:

Data Structures

- struct application_data

  Structure containing The GtkApplication and a list containing the GdsRenderGui objects.

Functions

- static void app_quit (GSimpleAction *action, GVariant *parameter, gpointer user_data)

  Callback for the menu entry 'Quit'.
- static void app_about (GSimpleAction *action, GVariant *parameter, gpointer user_data)

  Callback for the 'About' menu entry.

- static void gui_window_closed_callback (GdsRenderGui *gui, gpointer user_data)

  Called when a GUI main window is closed.
- static void gapp_activate (GApplication *app, gpointer user_data)

  Activation of the GUI.
- static int start_gui (int argc, char **argv)

  Start the graphical interface.
- static void print_version (void)

  Print the application version string to stdout.
- int main (int argc, char **argv)

  The "entry point" of the application.

Variables

- static const GActionEntry app_actions [ ]

  Contains the application menu entries.

13.83.1 Detailed Description

main.c

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file main.c.

13.83.2 Function Documentation

13.83.2.1 app_about()

```
static void app_about (
            GSimpleAction * action,
            GVariant * parameter,
            gpointer user_data )  [static]
```

Callback for the 'About' menu entry.

This function shows the about dialog.

Parameters

action	GSimpleAction, unused
parameter	Unused.
user_data	Unused

Definition at line 85 of file main.c.

13.83.2.2 app_quit()

```
static void app_quit (
             GSimpleAction * action,
             GVariant * parameter,
             gpointer user_data )  [static]
```

Callback for the menu entry 'Quit'.

Destroys all GUIs contained in the application_data structure provided by `user_data`.

The complete suspension of all main windows leads to the termination of the GApplication.

Parameters

action	unused
parameter	unused
user_data	application_data structure

Definition at line 58 of file main.c.

13.83.2.3 gapp_activate()

```
static void gapp_activate (
             GApplication * app,
             gpointer user_data )  [static]
```

Activation of the GUI.

Parameters

app	The GApplication reference
user_data	Used to store the individual GUI instances.

Definition at line 157 of file main.c.

Here is the call graph for this function: Here is the caller graph for this function:

13.83.2.4 gui_window_closed_callback()

```
static void gui_window_closed_callback (
             GdsRenderGui * gui,
             gpointer user_data )  [static]
```

Called when a GUI main window is closed.

The GdsRenderGui object associated with the closed main window is removed from the list of open GUIs (`user_data`) and dereferenced.

Parameters

gui	The GUI instance the closed main window belongs to
user_data	List of GUIs

Definition at line 143 of file main.c.

Here is the caller graph for this function:

13.83.2.5 main()

```
int main (
            int argc,
            char ** argv )
```

The "entry point" of the application.

Parameters

argc	Number of command line parameters
argv	Command line parameters

Returns

Execution status of the application

Definition at line 254 of file main.c.

Here is the call graph for this function:

13.83.2.6 print_version()

```
static void print_version (
            void  )  [static]
```

Print the application version string to stdout.

Definition at line 242 of file main.c.

Here is the caller graph for this function:

13.83.2.7 start_gui()

```
static int start_gui (
            int argc,
            char ** argv )  [static]
```

Start the graphical interface.

This function starts the GUI. If there's already a running instance of this program, a second window will be created in that instance and the second one is terminated.

Parameters

argc	
argv	

Returns

Definition at line 185 of file main.c.

Here is the call graph for this function: Here is the caller graph for this function:

13.83.3 Variable Documentation

13.83.3.1 app_actions

```
const GActionEntry app_actions[]  [static]
```

Initial value:

```
= {
        { "quit", app_quit, NULL, NULL, NULL, {0} },
        { "about", app_about, NULL, NULL, NULL, {0} },
}
```

Contains the application menu entries.

Definition at line 129 of file main.c.

13.84 main.c

Go to the documentation of this file.

```
00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018  Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter.  If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00026 #include <stdio.h>
00027 #include <gtk/gtk.h>
00028 #include <glib.h>
00029 #include <glib/gi18n.h>
00030 #include <locale.h>
00031
00032 #include <gds-render/gds-render-gui.h>
```

```
00033 #include <gds-render/command-line.h>
00034 #include <gds-render/output-renderers/external-renderer.h>
00035 #include <gds-render/version.h>
00036
00040 struct application_data {
00041         GtkApplication *app;
00042         GList *gui_list;
00043 };
00044
00058 static void app_quit(GSimpleAction *action, GVariant *parameter, gpointer user_data)
00059 {
00060         struct application_data * const appdata = (struct application_data *)user_data;
00061         (void)action;
00062         (void)parameter;
00063         GList *list_iter;
00064         GdsRenderGui *gui;
00065
00066         /* Dispose all GUIs */
00067         for (list_iter = appdata->gui_list; list_iter != NULL; list_iter = g_list_next(list_iter)) {
00068                 gui = RENDERER_GUI(list_iter->data);
00069                 g_object_unref(gui);
00070         }
00071
00072         g_list_free(appdata->gui_list);
00073         appdata->gui_list = NULL;
00074 }
00075
00085 static void app_about(GSimpleAction *action, GVariant *parameter, gpointer user_data)
00086 {
00087         GtkBuilder *builder;
00088         GtkDialog *dialog;
00089         GdkPixbuf *logo_buf;
00090         GError *error = NULL;
00091         (void)user_data;
00092         (void)action;
00093         (void)parameter;
00094         GString *comment_text;
00095
00096         comment_text = g_string_new(_("gds-render is a free tool for rendering GDS2 layout files into
      vector graphics."));
00097         g_string_append_printf(comment_text, _("\n\nFull git commit: %s"), _app_git_commit);
00098
00099         builder = gtk_builder_new_from_resource("/gui/about.glade");
00100         dialog = GTK_DIALOG(gtk_builder_get_object(builder, "about-dialog"));
00101         gtk_window_set_transient_for(GTK_WINDOW(dialog), NULL);
00102         gtk_about_dialog_set_version(GTK_ABOUT_DIALOG(dialog), _app_version_string);
00103         gtk_about_dialog_set_comments(GTK_ABOUT_DIALOG(dialog), comment_text->str);
00104
00105         g_string_free(comment_text, TRUE);
00106
00107         /* Load icon from resource */
00108         logo_buf = gdk_pixbuf_new_from_resource_at_scale("/images/logo.svg", 100, 100, TRUE, &error);
00109         if (logo_buf) {
00110                 /* Set logo */
00111                 gtk_about_dialog_set_logo(GTK_ABOUT_DIALOG(dialog), logo_buf);
00112
00113                 /* Pixbuf is now owned by about dialog. Unref */
00114                 g_object_unref(logo_buf);
00115         } else if (error) {
00116                 fprintf(stderr, _("Logo could not be displayed: %s\n"), error->message);
00117                 g_error_free(error);
00118         }
00119
00120         gtk_dialog_run(dialog);
00121
00122         gtk_widget_destroy(GTK_WIDGET(dialog));
00123         g_object_unref(builder);
00124 }
00125
00129 static const GActionEntry app_actions[] = {
00130         { "quit", app_quit, NULL, NULL, NULL, {0} },
00131         { "about", app_about, NULL, NULL, NULL, {0} },
00132 };
00133
00143 static void gui_window_closed_callback(GdsRenderGui *gui, gpointer user_data)
00144 {
00145         GList **gui_list = (GList **)user_data;
00146
00147         /* Dispose of Gui element */
00148         *gui_list = g_list_remove(*gui_list, gui);
00149         g_object_unref(gui);
00150 }
00151
00157 static void gapp_activate(GApplication *app, gpointer user_data)
00158 {
00159         GtkWindow *main_window;
00160         GdsRenderGui *gui;
```

```
00161         struct application_data * const appdata = (struct application_data *)user_data;
00162
00163         gui = gds_render_gui_new();
00164         appdata->gui_list = g_list_append(appdata->gui_list, gui);
00165
00166         g_signal_connect(gui, "window-closed", G_CALLBACK(gui_window_closed_callback),
      &appdata->gui_list);
00167
00168         main_window = gds_render_gui_get_main_window(gui);
00169
00170         gtk_application_add_window(GTK_APPLICATION(app), main_window);
00171         gtk_widget_show(GTK_WIDGET(main_window));
00172 }
00173
00185 static int start_gui(int argc, char **argv)
00186 {
00187         GtkApplication *gapp;
00188         GString *application_domain;
00189         int app_status;
00190         static struct application_data appdata = {
00191                 .gui_list = NULL
00192         };
00193         GMenu *menu;
00194         GMenu *m_quit;
00195         GMenu *m_about;
00196
00197         /*
00198          * Generate version dependent application id
00199          * This allows running the application in different versions at the same time.
00200          */
00201         application_domain = g_string_new(NULL);
00202         g_string_printf(application_domain, "de.shimatta.gds_render_%s", _app_git_commit);
00203
00204         gapp = gtk_application_new(application_domain->str, G_APPLICATION_FLAGS_NONE);
00205         g_string_free(application_domain, TRUE);
00206
00207         g_application_register(G_APPLICATION(gapp), NULL, NULL);
00208         g_signal_connect(gapp, "activate", G_CALLBACK(gapp_activate), &appdata);
00209
00210         if (g_application_get_is_remote(G_APPLICATION(gapp)) == TRUE) {
00211                 g_application_activate(G_APPLICATION(gapp));
00212                 printf(_("There is already an open instance. Will open second window in that
      instance.\n"));
00213                 return 0;
00214         }
00215
00216         menu = g_menu_new();
00217         m_quit = g_menu_new();
00218         m_about = g_menu_new();
00219         g_menu_append(m_quit, _("Quit"), "app.quit");
00220         g_menu_append(m_about, _("About"), "app.about");
00221         g_menu_append_section(menu, NULL, G_MENU_MODEL(m_about));
00222         g_menu_append_section(menu, NULL, G_MENU_MODEL(m_quit));
00223         g_action_map_add_action_entries(G_ACTION_MAP(gapp), app_actions,
00224                                         G_N_ELEMENTS(app_actions), &appdata);
00225         gtk_application_set_app_menu(GTK_APPLICATION(gapp), G_MENU_MODEL(menu));
00226
00227         g_object_unref(m_quit);
00228         g_object_unref(m_about);
00229         g_object_unref(menu);
00230
00231         app_status = g_application_run(G_APPLICATION(gapp), argc, argv);
00232         g_object_unref(gapp);
00233
00234         g_list_free(appdata.gui_list);
00235
00236         return app_status;
00237 }
00238
00242 static void print_version(void)
00243 {
00244         printf(_("This is gds-render, version: %s\n\nFor a list of supported commands execute with
      --help option.\n"),
00245                _app_version_string);
00246 }
00247
00254 int main(int argc, char **argv)
00255 {
00256         int i;
00257         GError *error = NULL;
00258         GOptionContext *context;
00259         gchar *gds_name;
00260         gchar **output_paths = NULL;
00261         gchar *mappingname = NULL;
00262         gchar *cellname = NULL;
00263         gchar **renderer_args = NULL;
00264         gboolean version = FALSE, pdf_standalone = FALSE, pdf_layers = FALSE;
```

```
00265         int scale = 1000;
00266         int app_status = 0;
00267         struct external_renderer_params so_render_params;
00268 
00269         so_render_params.so_path = NULL;
00270         so_render_params.cli_params = NULL;
00271 
00272         bindtextdomain(GETTEXT_PACKAGE, LOCALEDATADIR "/locale");
00273         bind_textdomain_codeset(GETTEXT_PACKAGE, "UTF-8");
00274         textdomain(GETTEXT_PACKAGE);
00275 
00276         GOptionEntry entries[] = {
00277                 {"version", 'v', 0, G_OPTION_ARG_NONE, &version, _("Print version"), NULL},
00278                 {"renderer", 'r', 0, G_OPTION_ARG_STRING_ARRAY, &renderer_args,
00279                         _("Renderer to use. Can be used multiple times."), "pdf|svg|tikz|ext"},
00280                 {"scale", 's', 0, G_OPTION_ARG_INT, &scale, _("Divide output coordinates by <SCALE>"),
      "<SCALE>" },
00281                 {"output-file", 'o', 0, G_OPTION_ARG_FILENAME_ARRAY, &output_paths,
00282                         _("Output file path. Can be used multiple times."), "PATH" },
00283                 {"mapping", 'm', 0, G_OPTION_ARG_FILENAME, &mappingname, _("Path for Layer Mapping
      File"), "PATH" },
00284                 {"cell", 'c', 0, G_OPTION_ARG_STRING, &cellname, _("Cell to render"), "NAME" },
00285                 {"tex-standalone", 'a', 0, G_OPTION_ARG_NONE, &pdf_standalone, _("Create standalone
      TeX"), NULL },
00286                 {"tex-layers", 'l', 0, G_OPTION_ARG_NONE, &pdf_layers, _("Create PDF Layers (OCG)"),
      NULL },
00287                 {"custom-render-lib", 'P', 0, G_OPTION_ARG_FILENAME, &so_render_params.so_path,
00288                         _("Path to a custom shared object, that implements the necessary rendering
      functions"), "PATH"},
00289                 {"render-lib-params", 'W', 0, G_OPTION_ARG_STRING, &so_render_params.cli_params,
00290                         _("Argument string passed to render lib"), NULL},
00291                 {NULL, 0, 0, 0, NULL, NULL, NULL}
00292         };
00293 
00294         context = g_option_context_new(_(" FILE - Convert GDS file <FILE> to graphic"));
00295         g_option_context_add_main_entries(context, entries, NULL);
00296         g_option_context_add_group(context, gtk_get_option_group(TRUE));
00297 
00298         if (!g_option_context_parse(context, &argc, &argv, &error)) {
00299                 g_print(_("Option parsing failed: %s\n"), error->message);
00300                 exit(1);
00301         }
00302 
00303         g_option_context_free(context);
00304 
00305         if (version) {
00306                 print_version();
00307                 goto ret_status;
00308         }
00309 
00310         if (argc >= 2) {
00311                 if (scale < 1) {
00312                         printf(_("Scale < 1 not allowed. Setting to 1\n"));
00313                         scale = 1;
00314                 }
00315 
00316                 /* Get gds name */
00317                 gds_name = argv[1];
00318 
00319                 /* Print out additional arguments as ignored */
00320                 for (i = 2; i < argc; i++)
00321                         printf(_("Ignored argument: %s"), argv[i]);
00322 
00323                 app_status =
00324                         command_line_convert_gds(gds_name, cellname, renderer_args, output_paths,
      mappingname,
00325                                                  &so_render_params, pdf_standalone, pdf_layers,
      scale);
00326 
00327         } else {
00328                 app_status = start_gui(argc, argv);
00329         }
00330 
00331 ret_status:
00332         /* If necessary, free command line parameters.
00333          * This is only really necessary for automated mem-leak testing.
00334          * Omitting these frees would be perfectly fine.
00335          */
00336         if (output_paths)
00337                 g_strfreev(output_paths);
00338         if (renderer_args)
00339                 g_strfreev(renderer_args);
00340         if (mappingname)
00341                 g_free(mappingname);
00342         if (cellname)
00343                 free(cellname);
00344         if (so_render_params.so_path)
```

```
00345                 free(so_render_params.so_path);
00346         if (so_render_params.cli_params)
00347                 g_free(so_render_params.cli_params);
00348
00349         return app_status;
00350 }
```

13.85 cairo-renderer.c File Reference

Output renderer for Cairo PDF export.

```
#include <math.h>
#include <stdlib.h>
#include <cairo.h>
#include <cairo-pdf.h>
#include <cairo-svg.h>
#include <glib/gi18n.h>
#include <gds-render/output-renderers/cairo-renderer.h>
#include <sys/wait.h>
#include <unistd.h>
```

Include dependency graph for cairo-renderer.c:

Data Structures

- struct _CairoRenderer
- struct cairo_layer

  The cairo_layer struct Each rendered layer is represented by this struct.

Functions

- static void revert_inherited_transform (struct cairo_layer *layers)

  Revert the last transformation on all layers.
- static void apply_inherited_transform_to_all_layers (struct cairo_layer *layers, const struct gds_point *origin, double magnification, gboolean flipping, double rotation, double scale)

  Applies transformation to all layers.
- static void render_cell (struct gds_cell *cell, struct cairo_layer *layers, double scale)

  render_cell Render a cell with its sub-cells
- static int read_line_from_fd (int fd, char *buff, size_t buff_size)

  Read a line from a file descriptor.
- static int cairo_renderer_render_cell_to_vector_file (GdsOutputRenderer *renderer, struct gds_cell *cell, GList *layer_infos, const char *pdf_file, const char *svg_file, double scale)

  Render `cell` to a PDF file specified by `pdf_file`.
- static void cairo_renderer_init (CairoRenderer *self)
- static int cairo_renderer_render_output (GdsOutputRenderer *renderer, struct gds_cell *cell, double scale)
- static void cairo_renderer_class_init (CairoRendererClass *klass)
- CairoRenderer * cairo_renderer_new_pdf ()

  Create new CairoRenderer for PDF output.
- CairoRenderer * cairo_renderer_new_svg ()

  Create new CairoRenderer for SVG output.

13.85.1 Detailed Description

Output renderer for Cairo PDF export.

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file cairo-renderer.c.

13.86 cairo-renderer.c

Go to the documentation of this file.

```
00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018  Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter.  If not, see <http://www.gnu.org/licenses/>.
00018  */
00029 #include <math.h>
00030 #include <stdlib.h>
00031 #include <cairo.h>
00032 #include <cairo-pdf.h>
00033 #include <cairo-svg.h>
00034 #include <glib/gi18n.h>
00035
00036 #include <gds-render/output-renderers/cairo-renderer.h>
00037 #include <sys/wait.h>
00038 #include <unistd.h>
00039
00040 struct _CairoRenderer {
00041         GdsOutputRenderer parent;
00042         gboolean svg;
00043 };
00044
00045 G_DEFINE_TYPE(CairoRenderer, cairo_renderer, GDS_RENDER_TYPE_OUTPUT_RENDERER)
00046
00047
00051 struct cairo_layer {
00052         cairo_t *cr;
00053         cairo_surface_t *rec;
00054         struct layer_info *linfo;
00055 };
00056
00061 static void revert_inherited_transform(struct cairo_layer *layers)
00062 {
00063         int i;
00064
00065         for (i = 0; i < MAX_LAYERS; i++) {
00066                 if (layers[i].cr == NULL)
00067                         continue;
00068                 cairo_restore(layers[i].cr);
00069         }
00070 }
00071
00081 static void apply_inherited_transform_to_all_layers(struct cairo_layer *layers,
00082                                                      const struct gds_point *origin,
00083                                                      double magnification,
00084                                                      gboolean flipping,
00085                                                      double rotation,
00086                                                      double scale)
00087 {
00088         int i;
```

```
00089         cairo_t *temp_layer_cr;
00090
00091         for (i = 0; i < MAX_LAYERS; i++) {
00092                 temp_layer_cr = layers[i].cr;
00093                 if (temp_layer_cr == NULL)
00094                         continue;
00095
00096                 /* Save the state and apply transformation */
00097                 cairo_save(temp_layer_cr);
00098                 cairo_translate(temp_layer_cr, (double)origin->x/scale, (double)origin->y/scale);
00099                 cairo_rotate(temp_layer_cr, M_PI*rotation/180.0);
00100                 cairo_scale(temp_layer_cr, magnification,
00101                             (flipping == TRUE ? -magnification : magnification));
00102         }
00103 }
00104
00111 static void render_cell(struct gds_cell *cell, struct cairo_layer *layers, double scale)
00112 {
00113         GList *instance_list;
00114         struct gds_cell *temp_cell;
00115         struct gds_cell_instance *cell_instance;
00116         GList *gfx_list;
00117         struct gds_graphics *gfx;
00118         GList *vertex_list;
00119         struct gds_point *vertex;
00120         cairo_t *cr;
00121
00122         /* Render child cells */
00123         for (instance_list = cell->child_cells; instance_list != NULL; instance_list =
      instance_list->next) {
00124                 cell_instance = (struct gds_cell_instance *)instance_list->data;
00125                 temp_cell = cell_instance->cell_ref;
00126                 if (temp_cell != NULL) {
00127                         apply_inherited_transform_to_all_layers(layers,
00128                                                                 &cell_instance->origin,
00129                                                                 cell_instance->magnification,
00130                                                                 cell_instance->flipped,
00131                                                                 cell_instance->angle,
00132                                                                 scale);
00133                         render_cell(temp_cell, layers, scale);
00134                         revert_inherited_transform(layers);
00135                 }
00136         }
00137
00138         /* Render graphics */
00139         for (gfx_list = cell->graphic_objs; gfx_list != NULL; gfx_list = gfx_list->next) {
00140                 gfx = (struct gds_graphics *)gfx_list->data;
00141
00142                 /* Get layer renderer */
00143                 if (gfx->layer >= MAX_LAYERS)
00144                         continue;
00145
00146                 cr = layers[gfx->layer].cr;
00147                 if (cr == NULL)
00148                         continue;
00149
00150                 /* Apply settings */
00151                 cairo_set_line_width(cr, (gfx->width_absolute ? gfx->width_absolute/scale : 1));
00152
00153                 switch (gfx->path_render_type) {
00154                 case PATH_FLUSH:
00155                         cairo_set_line_cap(cr, CAIRO_LINE_CAP_BUTT);
00156                         break;
00157                 case PATH_ROUNDED:
00158                         cairo_set_line_cap(cr, CAIRO_LINE_CAP_ROUND);
00159                         break;
00160                 case PATH_SQUARED:
00161                         cairo_set_line_cap(cr, CAIRO_LINE_CAP_SQUARE);
00162                         break;
00163                 }
00164
00165                 /* Add vertices */
00166                 for (vertex_list = gfx->vertices; vertex_list != NULL; vertex_list =
      vertex_list->next) {
00167                         vertex = (struct gds_point *)vertex_list->data;
00168
00169                         /* If first point -> move to, else line to */
00170                         if (vertex_list->prev == NULL)
00171                                 cairo_move_to(cr, vertex->x/scale, vertex->y/scale);
00172                         else
00173                                 cairo_line_to(cr, vertex->x/scale, vertex->y/scale);
00174                 }
00175
00176                 /* Create graphics object */
00177                 switch (gfx->gfx_type) {
00178                 case GRAPHIC_PATH:
00179                         cairo_stroke(cr);
```

```
00180                   break;
00181           case GRAPHIC_BOX:
00182                   /* Expected fallthrough */
00183           case GRAPHIC_POLYGON:
00184                   cairo_set_line_width(cr, 0.1/scale);
00185                   cairo_close_path(cr);
00186                   cairo_stroke_preserve(cr); // Prevent graphic glitches
00187                   cairo_fill(cr);
00188                   break;
00189           }
00190   } /* for gfx list */
00191 }
00192
00203 static int read_line_from_fd(int fd, char *buff, size_t buff_size)
00204 {
00205   ssize_t cnt;
00206   char c;
00207   unsigned int buff_cnt = 0;
00208
00209   while ((cnt = read(fd, &c, 1)) == 1) {
00210           if (buff_cnt < (buff_size-1)) {
00211                   buff[buff_cnt++] = c;
00212                   if (c == '\n')
00213                           break;
00214           } else {
00215                   break;
00216           }
00217   }
00218
00219   buff[buff_cnt] = 0;
00220   return (int)buff_cnt;
00221 }
00222
00233 static int cairo_renderer_render_cell_to_vector_file(GdsOutputRenderer *renderer,
00234                                                      struct gds_cell *cell,
00235                                                      GList *layer_infos,
00236                                                      const char *pdf_file,
00237                                                      const char *svg_file,
00238                                                      double scale)
00239 {
00240   cairo_surface_t *pdf_surface = NULL, *svg_surface = NULL;
00241   cairo_t *pdf_cr = NULL, *svg_cr = NULL;
00242   struct layer_info *linfo;
00243   struct cairo_layer *layers;
00244   struct cairo_layer *lay;
00245   GList *info_list;
00246   int i;
00247   double rec_x0, rec_y0, rec_width, rec_height;
00248   double xmin = INT32_MAX, xmax = INT32_MIN, ymin = INT32_MAX, ymax = INT32_MIN;
00249   pid_t process_id;
00250   int comm_pipe[2];
00251   char receive_message[200];
00252
00253   if (pdf_file == NULL && svg_file == NULL) {
00254           /* No output specified */
00255           return -1;
00256   }
00257
00258   /* Generate communication pipe for status updates */
00259   if (pipe(comm_pipe) == -1)
00260           return -2;
00261
00262   /* Fork to a new child process. This ensures the memory leaks (see issue #16) in Cairo don't
00263    * brick everything.
00264    *
00265    * And by the way: This now bricks all Windows compatibility. Deal with it.
00266    */
00267   process_id = fork();
00268   //process_id = -1;
00269   if (process_id < 0) {
00270           /* This should not happen */
00271           fprintf(stderr, _("Fatal error: Cairo Renderer: Could not spawn child process!"));
00272           exit(-2);
00273   } else if (process_id > 0) {
00274           /* Woohoo... Successfully dumped the shitty code to an unknowing victim */
00275           goto ret_parent;
00276   }
00277
00278   /* We are now in a separate process just for rendering the output image.
00279  * You may print a log message to the activity bar of the gui by writing a line
00280  * teminated with '\n' to comm_pipe[1]. This will be handled by the parent process.
00281  * Directly calling the update function
00282  *     gds_output_renderer_update_async_progress()
00283  * does not have any effect because this is a separate process.
00284  */
00285
00286   /*
```

```
00287      * Close stdin and (stdout and stderr may live on)
00288      */
00289         close(0);
00290         close(comm_pipe[0]);
00291
00292         layers = (struct cairo_layer *)calloc(MAX_LAYERS, sizeof(struct cairo_layer));
00293
00294         /* Clear layers */
00295         for (i = 0; i < MAX_LAYERS; i++) {
00296                 layers[i].cr = NULL;
00297                 layers[i].rec = NULL;
00298         }
00299
00300         /* Create recording surface for each layer */
00301         for (info_list = layer_infos; info_list != NULL; info_list = g_list_next(info_list)) {
00302                 linfo = (struct layer_info *)info_list->data;
00303                 if (linfo->layer < MAX_LAYERS) {
00304                         /* Layer shall not be rendered */
00305                         if (!linfo->render)
00306                                 continue;
00307
00308                         lay = &(layers[(unsigned int)linfo->layer]);
00309                         lay->linfo = linfo;
00310                         lay->rec = cairo_recording_surface_create(CAIRO_CONTENT_COLOR_ALPHA,
00311                                                                   NULL);
00312                         lay->cr = cairo_create(layers[(unsigned int)linfo->layer].rec);
00313                         cairo_scale(lay->cr, 1, -1); // Fix coordinate system
00314                         cairo_set_source_rgb(lay->cr, linfo->color.red, linfo->color.green,
      linfo->color.blue);
00315                 } else {
00316                         printf("Layer number (%d) too high!\n", linfo->layer);
00317                         goto ret_clear_layers;
00318                 }
00319         }
00320
00321         dprintf(comm_pipe[1], "Rendering layers\n");
00322         render_cell(cell, layers, scale);
00323
00324         /* get size of image and top left coordinate */
00325         for (info_list = layer_infos; info_list != NULL; info_list = g_list_next(info_list)) {
00326                 linfo = (struct layer_info *)info_list->data;
00327
00328                 if (linfo->layer >= MAX_LAYERS) {
00329                         printf(_("Layer number too high / outside of spec.\n"));
00330                         continue;
00331                 }
00332
00333                 if (!linfo->render)
00334                         continue;
00335
00336                 /* Print size */
00337                 cairo_recording_surface_ink_extents(layers[linfo->layer].rec, &rec_x0, &rec_y0,
00338                                 &rec_width, &rec_height);
00339                 dprintf(comm_pipe[1], _("Size of layer %d%s%s%s: <%lf x %lf> @ (%lf | %lf)\n"),
00340                         linfo->layer,
00341                         (linfo->name && linfo->name[0] ? " (" : ""),
00342                         (linfo->name && linfo->name[0] ? linfo->name : ""),
00343                         (linfo->name && linfo->name[0] ? ")" : ""),
00344                         rec_width, rec_height, rec_x0, rec_y0);
00345
00346                 /* update bounding box */
00347                 xmin = MIN(xmin, rec_x0);
00348                 xmax = MAX(xmax, rec_x0);
00349                 ymin = MIN(ymin, rec_y0);
00350                 ymax = MAX(ymax, rec_y0);
00351                 xmin = MIN(xmin, rec_x0+rec_width);
00352                 xmax = MAX(xmax, rec_x0+rec_width);
00353                 ymin = MIN(ymin, rec_y0+rec_height);
00354                 ymax = MAX(ymax, rec_y0+rec_height);
00355
00356         }
00357
00358         /* printf("Cell bounding box: (%lf | %lf) -- (%lf | %lf)\n", xmin, ymin, xmax, ymax); */
00359
00360         if (pdf_file) {
00361                 pdf_surface = cairo_pdf_surface_create(pdf_file, xmax-xmin, ymax-ymin);
00362                 pdf_cr = cairo_create(pdf_surface);
00363         }
00364
00365         if (svg_file) {
00366                 svg_surface = cairo_svg_surface_create(svg_file, xmax-xmin, ymax-ymin);
00367                 svg_cr = cairo_create(svg_surface);
00368         }
00369
00370         /* Write layers to PDF */
00371         for (info_list = layer_infos; info_list != NULL; info_list = g_list_next(info_list)) {
00372                 linfo = (struct layer_info *)info_list->data;
```

```
00373
00374                 if (linfo->layer >= MAX_LAYERS) {
00375                         printf(_("Layer outside of spec.\n"));
00376                         continue;
00377                 }
00378
00379                 if (!linfo->render)
00380                         continue;
00381
00382                 dprintf(comm_pipe[1], _("Exporting layer %d to file\n"), linfo->layer);
00383
00384                 if (pdf_file && pdf_cr) {
00385                         cairo_set_source_surface(pdf_cr, layers[linfo->layer].rec, -xmin, -ymin);
00386                         cairo_paint_with_alpha(pdf_cr, linfo->color.alpha);
00387                 }
00388
00389                 if (svg_file && svg_cr) {
00390                         cairo_set_source_surface(svg_cr, layers[linfo->layer].rec, -xmin, -ymin);
00391                         cairo_paint_with_alpha(svg_cr, linfo->color.alpha);
00392                 }
00393         }
00394
00395         if (pdf_file) {
00396                 cairo_show_page(pdf_cr);
00397                 cairo_destroy(pdf_cr);
00398                 cairo_surface_destroy(pdf_surface);
00399         }
00400
00401         if (svg_file) {
00402                 cairo_show_page(svg_cr);
00403                 cairo_destroy(svg_cr);
00404                 cairo_surface_destroy(svg_surface);
00405         }
00406
00407 ret_clear_layers:
00408         for (i = 0; i < MAX_LAYERS; i++) {
00409                 lay = &layers[i];
00410                 if (lay->cr) {
00411                         cairo_destroy(lay->cr);
00412                         cairo_surface_destroy(lay->rec);
00413                 }
00414         }
00415         free(layers);
00416
00417         printf(_("Cairo export finished. It might still be buggy!\n"));
00418
00419         /* Suspend child process */
00420         exit(0);
00421
00422 ret_parent:
00423         close(comm_pipe[1]);
00424
00425         while (read_line_from_fd(comm_pipe[0], receive_message, sizeof(receive_message)) > 0) {
00426                 /* Strip \n from string and replace with ' ' */
00427                 for (i = 0; receive_message[i] != '\0'; i++) {
00428                         if (receive_message[i] == '\n')
00429                                 receive_message[i] = ' ';
00430                 }
00431
00432                 /* Update asyc progress*/
00433                 gds_output_renderer_update_async_progress(renderer, receive_message);
00434         }
00435
00436         waitpid(process_id, NULL, 0);
00437
00438         close(comm_pipe[0]);
00439         return 0;
00440 }
00441
00442 static void cairo_renderer_init(CairoRenderer *self)
00443 {
00444         /* PDF default */
00445         self->svg = FALSE;
00446 }
00447
00448 static int cairo_renderer_render_output(GdsOutputRenderer *renderer,
00449                                         struct gds_cell *cell,
00450                                         double scale)
00451 {
00452         CairoRenderer *c_renderer = GDS_RENDER_CAIRO_RENDERER(renderer);
00453         const char *pdf_file = NULL;
00454         const char *svg_file = NULL;
00455         LayerSettings *settings;
00456         GList *layer_infos = NULL;
00457         const char *output_file;
00458         int ret;
00459
```

```
00460         if (!c_renderer)
00461                 return -2000;
00462
00463         output_file = gds_output_renderer_get_output_file(renderer);
00464         settings = gds_output_renderer_get_and_ref_layer_settings(renderer);
00465
00466         /* Set layer info list. In case of failure it remains NULL */
00467         if (settings)
00468                 layer_infos = layer_settings_get_layer_info_list(settings);
00469
00470         if (c_renderer->svg == TRUE)
00471                 svg_file = output_file;
00472         else
00473                 pdf_file = output_file;
00474
00475         gds_output_renderer_update_async_progress(renderer, _("Rendering Cairo Output..."));
00476         ret = cairo_renderer_render_cell_to_vector_file(renderer, cell, layer_infos, pdf_file,
      svg_file, scale);
00477
00478         if (settings)
00479                 g_object_unref(settings);
00480
00481         return ret;
00482 }
00483
00484 static void cairo_renderer_class_init(CairoRendererClass *klass)
00485 {
00486         GdsOutputRendererClass *renderer_class = GDS_RENDER_OUTPUT_RENDERER_CLASS(klass);
00487
00488         renderer_class->render_output = cairo_renderer_render_output;
00489 }
00490
00491 CairoRenderer *cairo_renderer_new_pdf()
00492 {
00493         CairoRenderer *renderer;
00494
00495         renderer = GDS_RENDER_CAIRO_RENDERER(g_object_new(GDS_RENDER_TYPE_CAIRO_RENDERER, NULL));
00496         renderer->svg = FALSE;
00497
00498         return renderer;
00499 }
00500
00501 CairoRenderer *cairo_renderer_new_svg()
00502 {
00503         CairoRenderer *renderer;
00504
00505         renderer = GDS_RENDER_CAIRO_RENDERER(g_object_new(GDS_RENDER_TYPE_CAIRO_RENDERER, NULL));
00506         renderer->svg = TRUE;
00507
00508         return renderer;
00509 }
00510
```

13.87 external-renderer.c File Reference

This file implements the dynamic library loading for the external rendering feature.

```
#include <dlfcn.h>
#include <stdio.h>
#include <sys/wait.h>
#include <glib/gi18n.h>
#include <gds-render/output-renderers/external-renderer.h>
#include <gds-render/version.h>
```

Include dependency graph for external-renderer.c:

Data Structures

- struct _ExternalRenderer

Macros

- #define FORCE_FORK 0U

  if != 0, then forking is forced regardless of the shared object's settings

Enumerations

- enum { PROP_SO_PATH = 1 , PROP_PARAM_STRING , N_PROPERTIES }

Functions

- static int external_renderer_render_cell (struct gds_cell *toplevel_cell, GList *layer_info_list, const char *output_file, double scale, const char *so_path, const char *params)

  Execute render function in shared object to render the supplied cell.
- static int external_renderer_render_output (GdsOutputRenderer *renderer, struct gds_cell *cell, double scale)
- static void external_renderer_get_property (GObject *obj, guint property_id, GValue *value, GParamSpec *pspec)
- static void external_renderer_set_property (GObject *obj, guint property_id, const GValue *value, GParamSpec *pspec)
- static void external_renderer_dispose (GObject *self_obj)
- static void external_renderer_class_init (ExternalRendererClass *klass)
- static void external_renderer_init (ExternalRenderer *self)
- ExternalRenderer * external_renderer_new ()

  Create new ExternalRenderer object.
- ExternalRenderer * external_renderer_new_with_so_and_param (const char *so_path, const char *param_string)

  Create new ExternalRenderer object with specified shared object path.

Variables

- static GParamSpec * external_renderer_properties [N_PROPERTIES] = {NULL}

13.87.1 Detailed Description

This file implements the dynamic library loading for the external rendering feature.

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file external-renderer.c.

13.88 external-renderer.c

Go to the documentation of this file.

```
00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018  Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter.  If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00031 #include <dlfcn.h>
00032 #include <stdio.h>
00033 #include <sys/wait.h>
00034 #include <glib/gi18n.h>
00035
00036 #include <gds-render/output-renderers/external-renderer.h>
00037 #include <gds-render/version.h>
00038
00039 #define FORCE_FORK 0U
00041 struct _ExternalRenderer {
00042         GdsOutputRenderer parent;
00043         char *shared_object_path;
00044         char *cli_param_string;
00045 };
00046
00047 enum {
00048         PROP_SO_PATH = 1,
00049         PROP_PARAM_STRING,
00050         N_PROPERTIES
00051 };
00052
00053 G_DEFINE_TYPE(ExternalRenderer, external_renderer, GDS_RENDER_TYPE_OUTPUT_RENDERER)
00054
00055
00065 static int external_renderer_render_cell(struct gds_cell *toplevel_cell, GList *layer_info_list,
00066                                          const char *output_file, double scale,  const char *so_path, const
       char *params)
00067 {
00068         int (*so_render_func)(struct gds_cell *, GList *, const char *, double) = NULL;
00069         int (*so_init_func)(const char *, const char *) = NULL;
00070         void *so_handle = NULL;
00071         char *error_msg;
00072         int forking_req;
00073         int ret = 0;
00074         pid_t fork_pid = 0;
00075         int forked_status;
00076
00077         if (!so_path) {
00078                 fprintf(stderr, _("Path to shared object not set!\n"));
00079                 return -3000;
00080         }
00081
00082         /* Check parameter sanity */
00083         if (!output_file || !toplevel_cell || !layer_info_list)
00084                 return -3000;
00085
00086         /* Load shared object */
00087         so_handle = dlopen(so_path, RTLD_LAZY);
00088         if (!so_handle) {
00089                 fprintf(stderr, _("Could not load external library '%s'\nDetailed error is:\n%s\n"),
       so_path, dlerror());
00090                 return -2000;
00091         }
00092
00093         /* Load rendering symbol from library */
00094         so_render_func = (int (*)(struct gds_cell *, GList *, const char *, double))
00095                                  dlsym(so_handle, xstr(EXTERNAL_LIBRARY_RENDER_FUNCTION));
00096         error_msg = dlerror();
00097         if (error_msg != NULL) {
00098                 fprintf(stderr, _("Rendering function not found in library:\n%s\n"), error_msg);
00099                 goto ret_close_so_handle;
00100         }
00101
```

```
00102         /* Load the init function */
00103         so_init_func = (int (*)(const char *, const char *))dlsym(so_handle,
      xstr(EXTERNAL_LIBRARY_INIT_FUNCTION));
00104         error_msg = dlerror();
00105         if (error_msg != NULL) {
00106                 fprintf(stderr, _("Init function not found in library:\n%s\n"), error_msg);
00107                 goto ret_close_so_handle;
00108         }
00109
00110         /* Check if forking is requested */
00111         if (dlsym(so_handle, xstr(EXTERNAL_LIBRARY_FORK_REQUEST)))
00112                 forking_req = 1;
00113         else if (FORCE_FORK)
00114                 forking_req = 1;
00115         else
00116                 forking_req = 0;
00117
00118         /* Execute */
00119
00120         g_message(_("Calling external renderer."));
00121
00122         if (forking_req)
00123                 fork_pid = fork();
00124         if (fork_pid != 0)
00125                 goto end_forked;
00126
00127         ret = so_init_func(params, _app_version_string);
00128         if (!ret)
00129                 ret = so_render_func(toplevel_cell, layer_info_list, output_file, scale);
00130
00131         /* If we are in a separate process, terminate here */
00132         if (forking_req)
00133                 exit(ret);
00134
00135         /* The forked paths end here */
00136 end_forked:
00137         if (forking_req) {
00138                 waitpid(fork_pid, &forked_status, 0);
00139                 ret = WEXITSTATUS(forked_status);
00140         }
00141
00142         g_message(_("External renderer finished."));
00143
00144 ret_close_so_handle:
00145         dlclose(so_handle);
00146         return ret;
00147 }
00148
00149 static int external_renderer_render_output(GdsOutputRenderer *renderer,
00150                                            struct gds_cell *cell,
00151                                            double scale)
00152 {
00153         ExternalRenderer *ext_renderer = GDS_RENDER_EXTERNAL_RENDERER(renderer);
00154         LayerSettings *settings;
00155         GList *layer_infos = NULL;
00156         const char *output_file;
00157         int ret;
00158
00159         output_file = gds_output_renderer_get_output_file(renderer);
00160         settings = gds_output_renderer_get_and_ref_layer_settings(renderer);
00161
00162         /* Set layer info list. In case of failure it remains NULL */
00163         if (settings)
00164                 layer_infos = layer_settings_get_layer_info_list(settings);
00165
00166         ret = external_renderer_render_cell(cell, layer_infos, output_file, scale,
      ext_renderer->shared_object_path,
00167                                             ext_renderer->cli_param_string);
00168         if (settings)
00169                 g_object_unref(settings);
00170
00171         return ret;
00172 }
00173
00174 static void external_renderer_get_property(GObject *obj, guint property_id, GValue *value, GParamSpec
      *pspec)
00175 {
00176         ExternalRenderer *self;
00177
00178         self = GDS_RENDER_EXTERNAL_RENDERER(obj);
00179
00180         switch (property_id) {
00181         case PROP_SO_PATH:
00182                 g_value_set_string(value, self->shared_object_path);
00183                 break;
00184         case PROP_PARAM_STRING:
00185                 g_value_set_string(value, self->cli_param_string);
```

```
00186                 break;
00187         default:
00188                 G_OBJECT_WARN_INVALID_PROPERTY_ID(obj, property_id, pspec);
00189                 break;
00190         }
00191 }
00192
00193 static void external_renderer_set_property(GObject *obj, guint property_id, const GValue *value,
      GParamSpec *pspec)
00194 {
00195         ExternalRenderer *self;
00196
00197         self = GDS_RENDER_EXTERNAL_RENDERER(obj);
00198
00199         switch (property_id) {
00200         case PROP_SO_PATH:
00201                 if (self->shared_object_path)
00202                         g_free(self->shared_object_path);
00203                 self->shared_object_path = g_value_dup_string(value);
00204                 break;
00205         case PROP_PARAM_STRING:
00206                 if (self->cli_param_string)
00207                         g_free(self->cli_param_string);
00208                 self->cli_param_string = g_value_dup_string(value);
00209                 break;
00210         default:
00211                 G_OBJECT_WARN_INVALID_PROPERTY_ID(obj, property_id, pspec);
00212                 break;
00213         }
00214 }
00215
00216 static void external_renderer_dispose(GObject *self_obj)
00217 {
00218         ExternalRenderer *self;
00219
00220         self = GDS_RENDER_EXTERNAL_RENDERER(self_obj);
00221
00222         if (self->shared_object_path) {
00223                 g_free(self->shared_object_path);
00224                 self->shared_object_path = NULL;
00225         }
00226
00227         G_OBJECT_CLASS(external_renderer_parent_class)->dispose(self_obj);
00228 }
00229
00230 static GParamSpec *external_renderer_properties[N_PROPERTIES] = {NULL};
00231
00232 static void external_renderer_class_init(ExternalRendererClass *klass)
00233 {
00234         GdsOutputRendererClass *inherited_parent_class;
00235         GObjectClass *oclass;
00236
00237         inherited_parent_class = GDS_RENDER_OUTPUT_RENDERER_CLASS(klass);
00238         oclass = G_OBJECT_CLASS(klass);
00239
00240         /* Override virtual function */
00241         inherited_parent_class->render_output = external_renderer_render_output;
00242
00243         /* Setup Gobject callbacks */
00244         oclass->set_property = external_renderer_set_property;
00245         oclass->get_property = external_renderer_get_property;
00246         oclass->dispose = external_renderer_dispose;
00247
00248         /* Setup properties */
00249         external_renderer_properties[PROP_SO_PATH] =
00250                         g_param_spec_string(N_("shared-object-path"),
00251                                             N_("Shared object file path"),
00252                                             N_("Path to the shared object to search rendering function
      in."),
00253                                             NULL,
00254                                             G_PARAM_READWRITE);
00255         external_renderer_properties[PROP_PARAM_STRING] =
00256                         g_param_spec_string(N_("param-string"),
00257                                             N_("Shared object renderer parameter string"),
00258                                             N_("Command line arguments passed to the external shared
      object renderer"),
00259                                             NULL,
00260                                             G_PARAM_READWRITE);
00261         g_object_class_install_properties(oclass, N_PROPERTIES, external_renderer_properties);
00262 }
00263
00264 static void external_renderer_init(ExternalRenderer *self)
00265 {
00266         self->shared_object_path = NULL;
00267         self->cli_param_string = NULL;
00268 }
00269
```

```
00270 ExternalRenderer *external_renderer_new()
00271 {
00272         return g_object_new(GDS_RENDER_TYPE_EXTERNAL_RENDERER, NULL);
00273 }
00274
00275 ExternalRenderer *external_renderer_new_with_so_and_param(const char *so_path, const char
       *param_string)
00276 {
00277         return g_object_new(GDS_RENDER_TYPE_EXTERNAL_RENDERER, N_("shared-object-path"), so_path,
00278                             N_("param-string"), param_string, NULL);
00279 }
00280
```

13.89 gds-output-renderer.c File Reference

Base GObject class for output renderers.

```
#include <gds-render/output-renderers/gds-output-renderer.h>
#include <glib/gi18n.h>
```

Include dependency graph for gds-output-renderer.c:

Data Structures

- struct renderer_params
- struct idle_function_params
- struct GdsOutputRendererPrivate

Enumerations

- enum { PROP_OUTPUT_FILE = 1 , PROP_LAYER_SETTINGS , N_PROPERTIES }
- enum gds_output_renderer_signal_ids { ASYNC_FINISHED = 0 , ASYNC_PROGRESS_CHANGED , GDS_OUTPUT_RENDERER_SIGNAL_COUNT }

Functions

- static int gds_output_renderer_render_dummy (GdsOutputRenderer ∗renderer, struct gds_cell ∗cell, double scale)
- static void gds_output_renderer_dispose (GObject ∗self_obj)
- static void gds_output_renderer_get_property (GObject ∗obj, guint property_id, GValue ∗value, GParamSpec ∗pspec)
- static void gds_output_renderer_set_property (GObject ∗obj, guint property_id, const GValue ∗value, GParamSpec ∗pspec)
- static void gds_output_renderer_class_init (GdsOutputRendererClass ∗klass)
- void gds_output_renderer_init (GdsOutputRenderer ∗self)
- GdsOutputRenderer ∗ gds_output_renderer_new ()

  Create a new GdsOutputRenderer GObject.
- GdsOutputRenderer ∗ gds_output_renderer_new_with_props (const char ∗output_file, LayerSettings ∗layer_settings)

  Create a new GdsOutputRenderer GObject with its properties.
- void gds_output_renderer_set_output_file (GdsOutputRenderer ∗renderer, const gchar ∗file_name)

  Convenience function for setting the "output-file" property.
- const char ∗ gds_output_renderer_get_output_file (GdsOutputRenderer ∗renderer)

  Convenience function for getting the "output-file" property.

- LayerSettings * gds_output_renderer_get_and_ref_layer_settings (GdsOutputRenderer *renderer)

  Get layer settings.
- void gds_output_renderer_set_layer_settings (GdsOutputRenderer *renderer, LayerSettings *settings)

  Set layer settings.
- int gds_output_renderer_render_output (GdsOutputRenderer *renderer, struct gds_cell *cell, double scale)

  gds_output_renderer_render_output
- static void gds_output_renderer_async_wrapper (GTask *task, gpointer source_object, gpointer task_data, GCancellable *cancellable)
- static void gds_output_renderer_async_finished (GObject *src_obj, GAsyncResult *res, gpointer user_data)
- int gds_output_renderer_render_output_async (GdsOutputRenderer *renderer, struct gds_cell *cell, double scale)

  Render output asynchronously.
- static gboolean idle_event_processor_callback (gpointer user_data)
- void gds_output_renderer_update_async_progress (GdsOutputRenderer *renderer, const char *status)

  This function emits the 'progress-changed' in the thread/context that triggered an asynchronous rendering.

Variables

- static guint gds_output_renderer_signals [GDS_OUTPUT_RENDERER_SIGNAL_COUNT]
- static GParamSpec * gds_output_renderer_properties [N_PROPERTIES] = {NULL}

13.89.1 Detailed Description

Base GObject class for output renderers.

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file gds-output-renderer.c.

13.90 gds-output-renderer.c

Go to the documentation of this file.

```
00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2019  Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter.  If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00030 #include <gds-render/output-renderers/gds-output-renderer.h>
00031 #include <glib/gi18n.h>
00032
00033 struct renderer_params {
00034                 struct gds_cell *cell;
```

```
00035                 double scale;
00036 };
00037
00038 struct idle_function_params {
00039         GMutex message_lock;
00040         char *status_message;
00041 };
00042
00043 typedef struct {
00044         gchar *output_file;
00045         LayerSettings *layer_settings;
00046         GMutex settings_lock;
00047         gboolean mutex_init_status;
00048         GTask *task;
00049         GMainContext *main_context;
00050         struct renderer_params async_params;
00051         struct idle_function_params idle_function_parameters;
00052         gpointer padding[11];
00053 } GdsOutputRendererPrivate;
00054
00055 enum {
00056         PROP_OUTPUT_FILE = 1,
00057         PROP_LAYER_SETTINGS,
00058         N_PROPERTIES
00059 };
00060
00061 G_DEFINE_TYPE_WITH_PRIVATE(GdsOutputRenderer, gds_output_renderer, G_TYPE_OBJECT)
00062
00063 enum gds_output_renderer_signal_ids {ASYNC_FINISHED = 0, ASYNC_PROGRESS_CHANGED,
      GDS_OUTPUT_RENDERER_SIGNAL_COUNT};
00064 static guint gds_output_renderer_signals[GDS_OUTPUT_RENDERER_SIGNAL_COUNT];
00065
00066 static int gds_output_renderer_render_dummy(GdsOutputRenderer *renderer,
00067                                             struct gds_cell *cell,
00068                                             double scale)
00069 {
00070         (void)renderer;
00071         (void)cell;
00072         (void)scale;
00073
00074         g_warning(_("Output renderer does not define a render_output function!"));
00075         return 0;
00076 }
00077
00078 static void gds_output_renderer_dispose(GObject *self_obj)
00079 {
00080         GdsOutputRenderer *renderer = GDS_RENDER_OUTPUT_RENDERER(self_obj);
00081         GdsOutputRendererPrivate *priv;
00082
00083         priv = gds_output_renderer_get_instance_private(renderer);
00084
00085         if (priv->mutex_init_status) {
00086                 /* Try locking the mutex, to test if it's free */
00087                 g_mutex_lock(&priv->settings_lock);
00088                 g_mutex_unlock(&priv->settings_lock);
00089                 g_mutex_clear(&priv->settings_lock);
00090
00091                 g_mutex_lock(&priv->idle_function_parameters.message_lock);
00092                 g_mutex_unlock(&priv->idle_function_parameters.message_lock);
00093                 g_mutex_clear(&priv->idle_function_parameters.message_lock);
00094
00095                 priv->mutex_init_status = FALSE;
00096         }
00097
00098         g_clear_object(&priv->task);
00099
00100         if (priv->output_file)
00101                 g_free(priv->output_file);
00102
00103         if (priv->idle_function_parameters.status_message) {
00104                 g_free(priv->idle_function_parameters.status_message);
00105                 priv->idle_function_parameters.status_message = NULL;
00106         }
00107
00108         g_clear_object(&priv->layer_settings);
00109
00110         /* Chain up to parent class */
00111         G_OBJECT_CLASS(gds_output_renderer_parent_class)->dispose(self_obj);
00112 }
00113
00114 static void gds_output_renderer_get_property(GObject *obj, guint property_id, GValue *value,
      GParamSpec *pspec)
00115 {
00116         GdsOutputRenderer *self = GDS_RENDER_OUTPUT_RENDERER(obj);
00117         GdsOutputRendererPrivate *priv;
00118
00119         priv = gds_output_renderer_get_instance_private(self);
```

```
00120
00121         switch (property_id) {
00122         case PROP_OUTPUT_FILE:
00123                 g_value_set_string(value, priv->output_file);
00124                 break;
00125         case PROP_LAYER_SETTINGS:
00126                 g_value_set_object(value, priv->layer_settings);
00127                 break;
00128         default:
00129                 G_OBJECT_WARN_INVALID_PROPERTY_ID(obj, property_id, pspec);
00130                 break;
00131         }
00132 }
00133
00134 static void gds_output_renderer_set_property(GObject *obj, guint property_id, const GValue *value,
      GParamSpec *pspec)
00135 {
00136         GdsOutputRenderer *self = GDS_RENDER_OUTPUT_RENDERER(obj);
00137         GdsOutputRendererPrivate *priv;
00138
00139         priv = gds_output_renderer_get_instance_private(self);
00140
00141         switch (property_id) {
00142         case PROP_OUTPUT_FILE:
00143                 g_mutex_lock(&priv->settings_lock);
00144                 if (priv->output_file)
00145                         g_free(priv->output_file);
00146                 priv->output_file = g_strdup(g_value_get_string(value));
00147                 g_mutex_unlock(&priv->settings_lock);
00148                 break;
00149         case PROP_LAYER_SETTINGS:
00150                 g_mutex_lock(&priv->settings_lock);
00151                 g_clear_object(&priv->layer_settings);
00152                 priv->layer_settings = g_value_get_object(value);
00153                 g_object_ref(priv->layer_settings);
00154                 g_mutex_unlock(&priv->settings_lock);
00155                 break;
00156         default:
00157                 G_OBJECT_WARN_INVALID_PROPERTY_ID(obj, property_id, pspec);
00158                 break;
00159         }
00160 }
00161
00162 static GParamSpec *gds_output_renderer_properties[N_PROPERTIES] = {NULL};
00163
00164 static void gds_output_renderer_class_init(GdsOutputRendererClass *klass)
00165 {
00166         GObjectClass *oclass = G_OBJECT_CLASS(klass);
00167         GType progress_changed_param_types[1] = {G_TYPE_POINTER};
00168
00169         klass->render_output = gds_output_renderer_render_dummy;
00170
00171         oclass->dispose = gds_output_renderer_dispose;
00172         oclass->set_property = gds_output_renderer_set_property;
00173         oclass->get_property = gds_output_renderer_get_property;
00174
00175         /* Setup properties */
00176         gds_output_renderer_properties[PROP_OUTPUT_FILE] =
00177                         g_param_spec_string(N_("output-file"), N_("output file"), N_("Output file for
      renderer"),
00178                                             NULL, G_PARAM_READWRITE);
00179         gds_output_renderer_properties[PROP_LAYER_SETTINGS] =
00180                         g_param_spec_object(N_("layer-settings"), N_("Layer Settings object"),
00181                                             N_("Object containing the layer rendering information"),
00182                                             GDS_RENDER_TYPE_LAYER_SETTINGS, G_PARAM_READWRITE);
00183         g_object_class_install_properties(oclass, N_PROPERTIES, gds_output_renderer_properties);
00184
00185         /* Setup output signals */
00186         gds_output_renderer_signals[ASYNC_FINISHED] =
00187                         g_signal_newv(N_("async-finished"), GDS_RENDER_TYPE_OUTPUT_RENDERER,
00188                                       G_SIGNAL_RUN_LAST | G_SIGNAL_NO_RECURSE,
00189                                       NULL,
00190                                       NULL,
00191                                       NULL,
00192                                       NULL,
00193                                       G_TYPE_NONE,
00194                                       0,
00195                                       NULL);
00196         gds_output_renderer_signals[ASYNC_PROGRESS_CHANGED] =
00197                         g_signal_newv(N_("progress-changed"), GDS_RENDER_TYPE_OUTPUT_RENDERER,
00198                                       G_SIGNAL_RUN_LAST | G_SIGNAL_NO_RECURSE,
00199                                       NULL,
00200                                       NULL,
00201                                       NULL,
00202                                       NULL,
00203                                       G_TYPE_NONE,
00204                                       1,
```

```
00205                                       progress_changed_param_types);
00206 }
00207
00208 void gds_output_renderer_init(GdsOutputRenderer *self)
00209 {
00210         GdsOutputRendererPrivate *priv;
00211
00212         priv = gds_output_renderer_get_instance_private(self);
00213
00214         priv->layer_settings = NULL;
00215         priv->output_file = NULL;
00216         priv->task = NULL;
00217         priv->mutex_init_status = TRUE;
00218         priv->main_context = NULL;
00219         priv->idle_function_parameters.status_message = NULL;
00220         g_mutex_init(&priv->settings_lock);
00221         g_mutex_init(&priv->idle_function_parameters.message_lock);
00222 }
00223
00224 GdsOutputRenderer *gds_output_renderer_new()
00225 {
00226         return GDS_RENDER_OUTPUT_RENDERER(g_object_new(GDS_RENDER_TYPE_OUTPUT_RENDERER, NULL));
00227 }
00228
00229 GdsOutputRenderer *gds_output_renderer_new_with_props(const char *output_file, LayerSettings
      *layer_settings)
00230 {
00231         return GDS_RENDER_OUTPUT_RENDERER(g_object_new(GDS_RENDER_TYPE_OUTPUT_RENDERER,
00232                                                        N_("layer-settings"), layer_settings,
00233                                                        N_("output-file"), output_file,
00234                                                        NULL));
00235 }
00236
00237 void gds_output_renderer_set_output_file(GdsOutputRenderer *renderer, const gchar *file_name)
00238 {
00239         g_return_if_fail(GDS_RENDER_IS_OUTPUT_RENDERER(renderer));
00240
00241         /* Check if the filename is actually filled */
00242         if (!file_name || !file_name[0])
00243                 return;
00244         g_object_set(renderer, N_("output-file"), file_name, NULL);
00245 }
00246
00247 const char *gds_output_renderer_get_output_file(GdsOutputRenderer *renderer)
00248 {
00249         const char *file = NULL;
00250
00251         g_object_get(renderer, N_("output-file"), &file, NULL);
00252         return file;
00253 }
00254
00255 LayerSettings *gds_output_renderer_get_and_ref_layer_settings(GdsOutputRenderer *renderer)
00256 {
00257         LayerSettings *ret = NULL;
00258         GdsOutputRendererPrivate *priv;
00259
00260         priv = gds_output_renderer_get_instance_private(renderer);
00261
00262         /* Acquire settings lock */
00263         g_mutex_lock(&priv->settings_lock);
00264
00265         /* This function seems to already reference the LayerSettings object */
00266         g_object_get(renderer, N_("layer-settings"), &ret, NULL);
00267
00268         /* It is now safe to clear the lock */
00269         g_mutex_unlock(&priv->settings_lock);
00270
00271         return ret;
00272 }
00273
00274 void gds_output_renderer_set_layer_settings(GdsOutputRenderer *renderer, LayerSettings *settings)
00275 {
00276         g_return_if_fail(GDS_RENDER_IS_LAYER_SETTINGS(settings));
00277
00278         g_object_set(renderer, N_("layer-settings"), settings, NULL);
00279 }
00280
00281 int gds_output_renderer_render_output(GdsOutputRenderer *renderer, struct gds_cell *cell, double
      scale)
00282 {
00283         int ret;
00284         GdsOutputRendererClass *klass;
00285         GdsOutputRendererPrivate *priv = gds_output_renderer_get_instance_private(renderer);
00286
00287         if (GDS_RENDER_IS_OUTPUT_RENDERER(renderer) == FALSE) {
00288                 g_error(_("Output Renderer not valid."));
00289                 return GDS_OUTPUT_RENDERER_GEN_ERR;
```

```
00290         }
00291
00292         if (!priv->output_file || !priv->output_file[0]) {
00293                 g_error(_("No/invalid output file set."));
00294                 return GDS_OUTPUT_RENDERER_GEN_ERR;
00295         }
00296
00297         if (!priv->layer_settings) {
00298                 g_error(_("No layer specification supplied."));
00299                 return GDS_OUTPUT_RENDERER_GEN_ERR;
00300         }
00301
00302         if (!cell) {
00303                 g_error(_("Output renderer called without cell to render."));
00304                 return GDS_OUTPUT_RENDERER_PARAM_ERR;
00305         }
00306
00307         klass = GDS_RENDER_OUTPUT_RENDERER_GET_CLASS(renderer);
00308         if (klass->render_output == NULL) {
00309                 g_critical(_("Output Renderer: Rendering function broken. This is a bug."));
00310                 return GDS_OUTPUT_RENDERER_GEN_ERR;
00311         }
00312
00313         ret = klass->render_output(renderer, cell, scale);
00314
00315         return ret;
00316 }
00317
00318 static void gds_output_renderer_async_wrapper(GTask *task,
00319                                               gpointer source_object,
00320                                               gpointer task_data,
00321                                               GCancellable *cancellable)
00322 {
00323         GdsOutputRenderer *renderer;
00324         GdsOutputRendererPrivate *priv;
00325         int ret;
00326         (void)task_data;
00327         (void)cancellable;
00328
00329         renderer = GDS_RENDER_OUTPUT_RENDERER(source_object);
00330         priv = gds_output_renderer_get_instance_private(renderer);
00331         if (!priv) {
00332                 ret = -1000;
00333                 goto ret_from_task;
00334         }
00335         if (!priv->mutex_init_status) {
00336                 ret = -1001;
00337                 goto ret_from_task;
00338         }
00339
00340         ret = gds_output_renderer_render_output(renderer, priv->async_params.cell,
      priv->async_params.scale);
00341
00342 ret_from_task:
00343         g_task_return_int(task, ret);
00344 }
00345
00346 static void gds_output_renderer_async_finished(GObject *src_obj, GAsyncResult *res, gpointer
      user_data)
00347 {
00348         GdsOutputRendererPrivate *priv;
00349         (void)user_data;
00350         (void)res; /* Will hopefully be destroyed later */
00351
00352         priv = gds_output_renderer_get_instance_private(GDS_RENDER_OUTPUT_RENDERER(src_obj));
00353
00354         priv->main_context = NULL;
00355
00356         g_signal_emit(src_obj, gds_output_renderer_signals[ASYNC_FINISHED], 0);
00357         g_clear_object(&priv->task);
00358
00359         /* Clear reference set in gds_output_renderer_render_output_async() */
00360         g_object_unref(src_obj);
00361 }
00362
00363 int gds_output_renderer_render_output_async(GdsOutputRenderer *renderer, struct gds_cell *cell, double
      scale)
00364 {
00365         GdsOutputRendererPrivate *priv;
00366         int ret = -1;
00367
00368         priv = gds_output_renderer_get_instance_private(renderer);
00369         if (priv->task) {
00370                 g_warning(_("Renderer already started asynchronously"));
00371                 return -2000;
00372         }
00373
```

```
00374         priv->task = g_task_new(renderer, NULL, gds_output_renderer_async_finished, NULL);
00375
00376         /* This function is not available on current debian distros. */
00377         /* g_task_set_name(priv->task, "Rendering Thread"); */
00378
00379         g_mutex_lock(&priv->settings_lock);
00380         priv->async_params.cell = cell;
00381         priv->async_params.scale = scale;
00382         priv->main_context = g_main_context_default();
00383         g_mutex_unlock(&priv->settings_lock);
00384
00385         /* Self reference. This could end up being nasty... */
00386         g_object_ref(renderer);
00387
00388         /* Do the magic */
00389         g_task_run_in_thread(priv->task, gds_output_renderer_async_wrapper);
00390
00391         return ret;
00392 }
00393
00394 static gboolean idle_event_processor_callback(gpointer user_data)
00395 {
00396         GdsOutputRenderer *renderer;
00397         GdsOutputRendererPrivate *priv;
00398         char *status_message;
00399
00400         /* If the rendering is finished before the mainloop gets to this point
00401          * the renderer is already disposed. Catch this!
00402          */
00403         if (!GDS_RENDER_IS_OUTPUT_RENDERER(user_data))
00404                 return FALSE;
00405
00406         renderer = GDS_RENDER_OUTPUT_RENDERER(user_data);
00407         priv = gds_output_renderer_get_instance_private(renderer);
00408
00409         if (g_mutex_trylock(&priv->idle_function_parameters.message_lock)) {
00410                 status_message = priv->idle_function_parameters.status_message;
00411                 g_signal_emit(renderer, gds_output_renderer_signals[ASYNC_PROGRESS_CHANGED], 0,
      status_message);
00412                 g_free(priv->idle_function_parameters.status_message);
00413                 priv->idle_function_parameters.status_message = NULL;
00414                 g_mutex_unlock(&priv->idle_function_parameters.message_lock);
00415         } else {
00416                 return TRUE;
00417         }
00418
00419         return FALSE;
00420 }
00421
00422 void gds_output_renderer_update_async_progress(GdsOutputRenderer *renderer, const char *status)
00423 {
00424         GSource *idle_event_processor;
00425         GdsOutputRendererPrivate *priv;
00426         gboolean skip_source = FALSE;
00427
00428         g_return_if_fail(GDS_RENDER_IS_OUTPUT_RENDERER(renderer));
00429         if (!status)
00430                 return;
00431
00432         priv = gds_output_renderer_get_instance_private(renderer);
00433
00434         /* If rendering is not async */
00435         if (!priv->main_context)
00436                 return;
00437
00438         g_mutex_lock(&priv->idle_function_parameters.message_lock);
00439         if (priv->idle_function_parameters.status_message) {
00440                 g_free(priv->idle_function_parameters.status_message);
00441
00442                 /* Skip adding new idle source because there's already an active one */
00443                 skip_source = TRUE;
00444         }
00445         priv->idle_function_parameters.status_message = g_strdup(status);
00446         g_mutex_unlock(&priv->idle_function_parameters.message_lock);
00447
00448         if (!skip_source) {
00449                 idle_event_processor = g_idle_source_new();
00450                 g_source_set_callback(idle_event_processor, idle_event_processor_callback,
      (gpointer)renderer, NULL);
00451                 g_source_attach(idle_event_processor, priv->main_context);
00452         }
00453 }
00454
```

13.91 latex-renderer.c File Reference

LaTeX Output Renderer.

```
#include <math.h>
#include <stdio.h>
#include <gds-render/output-renderers/latex-renderer.h>
#include <gdk/gdk.h>
#include <glib/gi18n.h>
```

Include dependency graph for latex-renderer.c:

Data Structures

- struct _LatexRenderer

  Struct representing the LaTeX-Renderer object.

Macros

- #define WRITEOUT_BUFFER(buff) fwrite((buff)->str, sizeof(char), (buff)->len, tex_file)

  Writes a GString `buffer` to the fixed file tex_file.

Enumerations

- enum { PROP_STANDALONE = 1 , PROP_PDF_LAYERS , N_PROPERTIES }

Functions

- static void write_layer_definitions (FILE *tex_file, GList *layer_infos, GString *buffer)

  Write the layer declarration to TeX file.
- static gboolean write_layer_env (FILE *tex_file, GdkRGBA *color, int layer, GList *linfo, GString *buffer)

  Write layer Envirmonment.
- static void generate_graphics (FILE *tex_file, GList *graphics, GList *linfo, GString *buffer, double scale)

  Writes a graphics object to the specified tex_file.
- static void render_cell (struct gds_cell *cell, GList *layer_infos, FILE *tex_file, GString *buffer, double scale, GdsOutputRenderer *renderer)

  Render cell to file.
- static int latex_render_cell_to_code (struct gds_cell *cell, GList *layer_infos, FILE *tex_file, double scale, gboolean create_pdf_layers, gboolean standalone_document, GdsOutputRenderer *renderer)
- static int latex_renderer_render_output (GdsOutputRenderer *renderer, struct gds_cell *cell, double scale)
- static void latex_renderer_init (LatexRenderer *self)
- static void latex_renderer_get_property (GObject *obj, guint property_id, GValue *value, GParamSpec *pspec)
- static void latex_renderer_set_property (GObject *obj, guint property_id, const GValue *value, GParamSpec *pspec)
- static void latex_renderer_class_init (LatexRendererClass *klass)
- LatexRenderer * latex_renderer_new ()

  Create new LatexRenderer object.
- LatexRenderer * latex_renderer_new_with_options (gboolean pdf_layers, gboolean standalone)

  Create new LatexRenderer object.

Variables

- static GParamSpec * latex_renderer_properties [N_PROPERTIES] = {NULL}

13.91.1 Detailed Description

LaTeX Output Renderer.

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file latex-renderer.c.

13.92 latex-renderer.c

Go to the documentation of this file.

```
00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018  Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter.  If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00026 #include <math.h>
00027 #include <stdio.h>
00028 #include <gds-render/output-renderers/latex-renderer.h>
00029 #include <gdk/gdk.h>
00030 #include <glib/gi18n.h>
00031
00042 struct _LatexRenderer {
00043         GdsOutputRenderer parent;
00044         gboolean tex_standalone;
00045         gboolean pdf_layers;
00046 };
00047
00048 G_DEFINE_TYPE(LatexRenderer, latex_renderer, GDS_RENDER_TYPE_OUTPUT_RENDERER)
00049
00050 enum {
00051         PROP_STANDALONE = 1,
00052         PROP_PDF_LAYERS,
00053         N_PROPERTIES
00054 };
00055
00060 #define WRITEOUT_BUFFER(buff) fwrite((buff)->str, sizeof(char), (buff)->len, tex_file)
00061
00074 static void write_layer_definitions(FILE *tex_file, GList *layer_infos, GString *buffer)
00075 {
00076         GList *list;
00077         struct layer_info *lifo;
00078
00079         for (list = layer_infos; list != NULL; list = list->next) {
00080                 lifo = (struct layer_info *)list->data;
00081
00082                 if (!lifo->render)
00083                         continue;
00084
00085                 g_string_printf(buffer,
       "\\pgfdeclarelayer{l%d}\n\\definecolor{c%d}{rgb}{%lf,%lf,%lf}\n",
```

```
00086                                 lifo->layer, lifo->layer,
00087                                 lifo->color.red, lifo->color.green, lifo->color.blue);
00088                 WRITEOUT_BUFFER(buffer);
00089         }
00090
00091         g_string_printf(buffer, "\\pgfsetlayers{");
00092         WRITEOUT_BUFFER(buffer);
00093
00094         for (list = layer_infos; list != NULL; list = list->next) {
00095                 lifo = (struct layer_info *)list->data;
00096
00097                 if (!lifo->render)
00098                         continue;
00099
00100                 g_string_printf(buffer, "l%d,", lifo->layer);
00101                 WRITEOUT_BUFFER(buffer);
00102         }
00103         g_string_printf(buffer, "main}\n");
00104         WRITEOUT_BUFFER(buffer);
00105 }
00106
00133 static gboolean write_layer_env(FILE *tex_file, GdkRGBA *color, int layer, GList *linfo, GString
      *buffer)
00134 {
00135         GList *temp;
00136         struct layer_info *inf;
00137
00138         for (temp = linfo; temp != NULL; temp = temp->next) {
00139                 inf = (struct layer_info *)temp->data;
00140                 if (inf->layer == layer && inf->render) {
00141                         color->alpha = inf->color.alpha;
00142                         color->red = inf->color.red;
00143                         color->green = inf->color.green;
00144                         color->blue = inf->color.blue;
00145                         g_string_printf(buffer,
00146
      "\\begin{pgfonlayer}{l%d}\n\\ifcreatepdflayers\n\\begin{scope}[ocg={ref=%d,
      status=visible,name={%s}}]\n\\fi\n",
00147                                         layer, layer, inf->name);
00148                         WRITEOUT_BUFFER(buffer);
00149                         return TRUE;
00150                 }
00151         }
00152         return FALSE;
00153 }
00154
00166 static void generate_graphics(FILE *tex_file, GList *graphics, GList *linfo, GString *buffer, double
      scale)
00167 {
00168         GList *temp;
00169         GList *temp_vertex;
00170         struct gds_graphics *gfx;
00171         struct gds_point *pt;
00172         GdkRGBA color;
00173         static const char * const line_caps[] = {"butt", "round", "rect"};
00174
00175         for (temp = graphics; temp != NULL; temp = temp->next) {
00176                 gfx = (struct gds_graphics *)temp->data;
00177                 if (write_layer_env(tex_file, &color, (int)gfx->layer, linfo, buffer) == TRUE) {
00178
00179                         /* Layer is defined => create graphics */
00180                         if (gfx->gfx_type == GRAPHIC_POLYGON || gfx->gfx_type == GRAPHIC_BOX) {
00181                                 g_string_printf(buffer,
00182                                                 "\\draw[line width=0.00001 pt, draw={c%d}, fill={c%d},
      fill opacity={%lf}] ",
00183                                                 gfx->layer, gfx->layer, color.alpha);
00184                                 WRITEOUT_BUFFER(buffer);
00185                                 /* Append vertices */
00186                                 for (temp_vertex = gfx->vertices;
00187                                      temp_vertex != NULL;
00188                                      temp_vertex = temp_vertex->next) {
00189                                         pt = (struct gds_point *)temp_vertex->data;
00190                                         g_string_printf(buffer, "(%lf pt, %lf pt) -- ",
00191                                                         ((double)pt->x)/scale,
00192                                                         ((double)pt->y)/scale);
00193                                         WRITEOUT_BUFFER(buffer);
00194                                 }
00195                                 g_string_printf(buffer, "cycle;\n");
00196                                 WRITEOUT_BUFFER(buffer);
00197                         } else if (gfx->gfx_type == GRAPHIC_PATH) {
00198
00199                                 if (g_list_length(gfx->vertices) < 2) {
00200                                         printf("Cannot write path with less than 2 points\n");
00201                                         break;
00202                                 }
00203
00204                                 if (gfx->path_render_type < 0 || gfx->path_render_type > 2) {
```

```
00205                                           printf("Path type unrecognized. Setting to 'flushed'\n");
00206                                           gfx->path_render_type = PATH_FLUSH;
00207                                   }
00208
00209                                   g_string_printf(buffer, "\\draw[line width=%lf pt, draw={c%d},
      opacity={%lf}, cap=%s] ",
00210                                                   gfx->width_absolute/scale, gfx->layer, color.alpha,
00211                                                   line_caps[gfx->path_render_type]);
00212                                   WRITEOUT_BUFFER(buffer);
00213
00214                                   /* Append vertices */
00215                                   for (temp_vertex = gfx->vertices;
00216                                        temp_vertex != NULL;
00217                                        temp_vertex = temp_vertex->next) {
00218                                           pt = (struct gds_point *)temp_vertex->data;
00219                                           g_string_printf(buffer, "(%lf pt, %lf pt)%s",
00220                                                           ((double)pt->x)/scale,
00221                                                           ((double)pt->y)/scale,
00222                                                           (temp_vertex->next ? " -- " : ""));
00223                                           WRITEOUT_BUFFER(buffer);
00224                                   }
00225                                   g_string_printf(buffer, ";\n");
00226                                   WRITEOUT_BUFFER(buffer);
00227                           }
00228
00229                           g_string_printf(buffer,
      "\\ifcreatepdflayers\n\\end{scope}\n\\fi\n\\end{pgfonlayer}\n");
00230                           WRITEOUT_BUFFER(buffer);
00231                   }
00232
00233           } /* For graphics */
00234 }
00235
00245 static void render_cell(struct gds_cell *cell, GList *layer_infos, FILE *tex_file, GString *buffer,
      double scale,
00246                           GdsOutputRenderer *renderer)
00247 {
00248           GString *status;
00249           GList *list_child;
00250           struct gds_cell_instance *inst;
00251
00252           status = g_string_new(NULL);
00253           g_string_printf(status, _("Generating cell %s"), cell->name);
00254           gds_output_renderer_update_async_progress(renderer, status->str);
00255           g_string_free(status, TRUE);
00256
00257           /* Draw polygons of current cell */
00258           generate_graphics(tex_file, cell->graphic_objs, layer_infos, buffer, scale);
00259
00260           /* Draw polygons of childs */
00261           for (list_child = cell->child_cells; list_child != NULL; list_child = list_child->next) {
00262                   inst = (struct gds_cell_instance *)list_child->data;
00263
00264                   /* Abort if cell has no reference */
00265                   if (!inst->cell_ref)
00266                           continue;
00267
00268                   /* generate translation scope */
00269                   g_string_printf(buffer, "\\begin{scope}[shift={(%lf pt,%lf pt)}]\n",
00270                                   ((double)inst->origin.x) / scale, ((double)inst->origin.y) / scale);
00271                   WRITEOUT_BUFFER(buffer);
00272
00273                   g_string_printf(buffer, "\\begin{scope}[rotate=%lf]\n", inst->angle);
00274                   WRITEOUT_BUFFER(buffer);
00275
00276                   g_string_printf(buffer, "\\begin{scope}[yscale=%lf, xscale=%lf]\n",
00277                                   (inst->flipped ? -1*inst->magnification : inst->magnification),
00278                                   inst->magnification);
00279                   WRITEOUT_BUFFER(buffer);
00280
00281                   render_cell(inst->cell_ref, layer_infos, tex_file, buffer, scale, renderer);
00282
00283                   g_string_printf(buffer, "\\end{scope}\n");
00284                   WRITEOUT_BUFFER(buffer);
00285
00286                   g_string_printf(buffer, "\\end{scope}\n");
00287                   WRITEOUT_BUFFER(buffer);
00288
00289                   g_string_printf(buffer, "\\end{scope}\n");
00290                   WRITEOUT_BUFFER(buffer);
00291           }
00292
00293 }
00294
00295 static int latex_render_cell_to_code(struct gds_cell *cell, GList *layer_infos, FILE *tex_file, double
      scale,
00296                                      gboolean create_pdf_layers, gboolean standalone_document,
```

```
      GdsOutputRenderer *renderer)
00297 {
00298         GString *working_line;
00299
00300
00301         if (!tex_file || !layer_infos || !cell)
00302                 return -1;
00303
00304         /* 10 kB Line working buffer should be enough */
00305         working_line = g_string_new_len(NULL, LATEX_LINE_BUFFER_KB*1024);
00306
00307         /* standalone foo */
00308         g_string_printf(working_line, "\\newif\\iftestmode\n\\testmode%s\n",
00309                         (standalone_document ? "true" : "false"));
00310         WRITEOUT_BUFFER(working_line);
00311         g_string_printf(working_line, "\\newif\\ifcreatepdflayers\n\\createpdflayers%s\n",
00312                         (create_pdf_layers ? "true" : "false"));
00313         WRITEOUT_BUFFER(working_line);
00314         g_string_printf(working_line, "\\iftestmode\n");
00315         WRITEOUT_BUFFER(working_line);
00316         g_string_printf(working_line,
00317
      "\\documentclass[tikz]{standalone}\n\\usepackage{xcolor}\n\\usetikzlibrary{ocgx}\n\\begin{document}\n");
00318         WRITEOUT_BUFFER(working_line);
00319         g_string_printf(working_line, "\\fi\n");
00320         WRITEOUT_BUFFER(working_line);
00321
00322         /* Write layer definitions */
00323         write_layer_definitions(tex_file, layer_infos, working_line);
00324
00325         /* Open tikz Pictute */
00326         g_string_printf(working_line, "\\begin{tikzpicture}\n");
00327         WRITEOUT_BUFFER(working_line);
00328
00329         /* Generate graphics output */
00330         render_cell(cell, layer_infos, tex_file, working_line, scale, renderer);
00331
00332
00333         g_string_printf(working_line, "\\end{tikzpicture}\n");
00334         WRITEOUT_BUFFER(working_line);
00335
00336         g_string_printf(working_line, "\\iftestmode\n");
00337         WRITEOUT_BUFFER(working_line);
00338         g_string_printf(working_line, "\\end{document}\n");
00339         WRITEOUT_BUFFER(working_line);
00340         g_string_printf(working_line, "\\fi\n");
00341         WRITEOUT_BUFFER(working_line);
00342
00343         fflush(tex_file);
00344         g_string_free(working_line, TRUE);
00345
00346         return 0;
00347 }
00348
00349 static int latex_renderer_render_output(GdsOutputRenderer *renderer,
00350                                         struct gds_cell *cell,
00351                                         double scale)
00352 {
00353         LatexRenderer *l_renderer = GDS_RENDER_LATEX_RENDERER(renderer);
00354         FILE *tex_file;
00355         int ret = -2;
00356         LayerSettings *settings;
00357         GList *layer_infos = NULL;
00358         const char *output_file;
00359
00360         output_file = gds_output_renderer_get_output_file(renderer);
00361         settings = gds_output_renderer_get_and_ref_layer_settings(renderer);
00362
00363         /* Set layer info list. In case of failure it remains NULL */
00364         if (settings)
00365                 layer_infos = layer_settings_get_layer_info_list(settings);
00366
00367         tex_file = fopen(output_file, "w");
00368         if (tex_file) {
00369                 ret = latex_render_cell_to_code(cell, layer_infos, tex_file, scale,
00370                                                 l_renderer->pdf_layers, l_renderer->tex_standalone,
      renderer);
00371                 fclose(tex_file);
00372         } else {
00373         g_warning(_("Could not open LaTeX output file"));
00374         }
00375
00376         if (settings)
00377                 g_object_unref(settings);
00378
00379         return ret;
00380 }
```

```
00381
00382 static void latex_renderer_init(LatexRenderer *self)
00383 {
00384         self->pdf_layers = FALSE;
00385         self->tex_standalone = FALSE;
00386 }
00387
00388 static void latex_renderer_get_property(GObject *obj, guint property_id, GValue *value, GParamSpec
      *pspec)
00389 {
00390         LatexRenderer *self = GDS_RENDER_LATEX_RENDERER(obj);
00391
00392         switch (property_id) {
00393         case PROP_STANDALONE:
00394                 g_value_set_boolean(value, self->tex_standalone);
00395                 break;
00396         case PROP_PDF_LAYERS:
00397                 g_value_set_boolean(value, self->pdf_layers);
00398                 break;
00399         default:
00400                 G_OBJECT_WARN_INVALID_PROPERTY_ID(obj, property_id, pspec);
00401                 break;
00402         }
00403 }
00404
00405 static void latex_renderer_set_property(GObject *obj, guint property_id, const GValue *value,
      GParamSpec *pspec)
00406 {
00407         LatexRenderer *self = GDS_RENDER_LATEX_RENDERER(obj);
00408
00409         switch (property_id) {
00410         case PROP_STANDALONE:
00411                 self->tex_standalone = g_value_get_boolean(value);
00412                 break;
00413         case PROP_PDF_LAYERS:
00414                 self->pdf_layers = g_value_get_boolean(value);
00415                 break;
00416         default:
00417                 G_OBJECT_WARN_INVALID_PROPERTY_ID(obj, property_id, pspec);
00418                 break;
00419         }
00420 }
00421
00422 static GParamSpec *latex_renderer_properties[N_PROPERTIES] = {NULL};
00423
00424 static void latex_renderer_class_init(LatexRendererClass *klass)
00425 {
00426         GdsOutputRendererClass *render_class = GDS_RENDER_OUTPUT_RENDERER_CLASS(klass);
00427         GObjectClass *oclass = G_OBJECT_CLASS(klass);
00428
00429         /* Overwrite virtual function */
00430         render_class->render_output = latex_renderer_render_output;
00431
00432         /* Property stuff */
00433         oclass->get_property = latex_renderer_get_property;
00434         oclass->set_property = latex_renderer_set_property;
00435
00436         latex_renderer_properties[PROP_STANDALONE] =
00437                         g_param_spec_boolean("standalone",
00438                                              N_("Standalone TeX file"),
00439                                              N_("Generate a standalone LaTeX file."),
00440                                              FALSE,
00441                                              G_PARAM_READWRITE);
00442         latex_renderer_properties[PROP_PDF_LAYERS] =
00443                         g_param_spec_boolean("pdf-layers",
00444                                              N_("PDF OCR layers"),
00445                                              N_("Generate OCR layers"),
00446                                              FALSE,
00447                                              G_PARAM_READWRITE);
00448
00449         g_object_class_install_properties(oclass, N_PROPERTIES, latex_renderer_properties);
00450 }
00451
00452 LatexRenderer *latex_renderer_new()
00453 {
00454         return GDS_RENDER_LATEX_RENDERER(g_object_new(GDS_RENDER_TYPE_LATEX_RENDERER, NULL));
00455 }
00456
00457 LatexRenderer *latex_renderer_new_with_options(gboolean pdf_layers, gboolean standalone)
00458 {
00459         GObject *obj;
00460
00461         obj = g_object_new(GDS_RENDER_TYPE_LATEX_RENDERER, "standalone", standalone, "pdf-layers",
      pdf_layers, NULL);
00462         return GDS_RENDER_LATEX_RENDERER(obj);
00463 }
00464
```

13.93 plugin-main.c File Reference

```
#include <stdio.h>
#include <glib.h>
#include <gds-render/gds-utils/gds-types.h>
#include <gds-render/output-renderers/external-renderer-interfaces.h>
```

Include dependency graph for plugin-main.c:

Functions

- int EXPORTED_FUNC_DECL() EXTERNAL_LIBRARY_RENDER_FUNCTION (struct gds_cell *toplevel, GList *layer_info_list, const char *output_file_name, double scale)
- int EXPORTED_FUNC_DECL() EXTERNAL_LIBRARY_INIT_FUNCTION (const char *params, const char *version)

13.94 plugin-main.c

Go to the documentation of this file.

```
00001 /*
00002  * GDSII-Converter example plugin
00003  * Copyright (C) 2019  Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter.  If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00029 #include <stdio.h>
00030 #include <glib.h>
00031 #include <gds-render/gds-utils/gds-types.h>
00032 #include <gds-render/output-renderers/external-renderer-interfaces.h>
00033
00034 int EXPORTED_FUNC_DECL(EXTERNAL_LIBRARY_RENDER_FUNCTION)(struct gds_cell *toplevel, GList
       *layer_info_list, const char *output_file_name, double scale)
00035 {
00036         if (!toplevel)
00037                 return -1000;
00038
00039         printf("Rendering %s\n", toplevel->name);
00040         return 0;
00041 }
00042
00043 int EXPORTED_FUNC_DECL(EXTERNAL_LIBRARY_INIT_FUNCTION)(const char *params, const char *version)
00044 {
00045         printf("Init with params: %s\ngds-render version: %s\n", params, version);
00046         return 0;
00047 }
00048
```

13.95 README.MD File Reference

13.96 version.c File Reference

Variables

- const char * _app_version_string = "! version not set !"

This string holds the Git Based Version Number of the app.

- const char * _app_git_commit = "! Commit hash not available !"

  This string holds the git commit hash of the current HEAD revision.

13.97 version.c

Go to the documentation of this file.

```
00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018  Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter.  If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00027 #ifdef PROJECT_GIT_VERSION
00028 #define xstr(a) str(a)
00029 #define str(a) #a
00030 const char *_app_version_string = xstr(PROJECT_GIT_VERSION);
00031 #else
00032 const char *_app_version_string = "! version not set !";
00033 #endif
00034
00035 #ifdef PROJECT_GIT_COMMIT
00036 #define xstr(a) str(a)
00037 #define str(a) #a
00038 const char *_app_git_commit = xstr(PROJECT_GIT_COMMIT);
00039 #else
00040 const char *_app_git_commit = "! Commit hash not available !";
00041 #endif
00042
```

13.98 activity-bar.c File Reference

Status bar indicating activity of the program.

```
#include <gds-render/widgets/activity-bar.h>
#include <glib/gi18n.h>
```

Include dependency graph for activity-bar.c:

13.99 activity-bar.c

Go to the documentation of this file.

```
00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2019  Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
```

```
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter.  If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00020 /*
00021  * The drag and drop implementation is adapted from
00022  * https://gitlab.gnome.org/GNOME/gtk/blob/gtk-3-22/tests/testlist3.c
00023  *
00024  * Thanks to the GTK3 people for creating these examples.
00025  */
00026
00039 #include <gds-render/widgets/activity-bar.h>
00040 #include <glib/gi18n.h>
00041
00043 struct _ActivityBar {
00044         GtkBox super;
00045         /* Private stuff */
00046         GtkWidget *spinner;
00047         GtkWidget *label;
00048 };
00049
00050 G_DEFINE_TYPE(ActivityBar, activity_bar, GTK_TYPE_BOX)
00051
00052 static void activity_bar_dispose(GObject *obj)
00053 {
00054         ActivityBar *bar;
00055
00056         bar = ACTIVITY_BAR(obj);
00057
00058         /* Clear references on owned objects */
00059         g_clear_object(&bar->label);
00060         g_clear_object(&bar->spinner);
00061
00062         /* Chain up */
00063         G_OBJECT_CLASS(activity_bar_parent_class)->dispose(obj);
00064 }
00065
00066 static void activity_bar_class_init(ActivityBarClass *klass)
00067 {
00068         GObjectClass *oclass = G_OBJECT_CLASS(klass);
00069
00070         oclass->dispose = activity_bar_dispose;
00071 }
00072
00073 static void activity_bar_init(ActivityBar *self)
00074 {
00075         GtkContainer *box = GTK_CONTAINER(self);
00076
00077         /* Create Widgets */
00078         self->label = gtk_label_new("");
00079         self->spinner = gtk_spinner_new();
00080
00081         /* Add to this widget and show */
00082         gtk_container_add(box, self->spinner);
00083         gtk_container_add(box, self->label);
00084         gtk_widget_show(self->label);
00085         gtk_widget_show(self->spinner);
00086
00087         g_object_ref(self->spinner);
00088         g_object_ref(self->label);
00089 }
00090
00091 ActivityBar *activity_bar_new()
00092 {
00093         ActivityBar *bar;
00094
00095         bar = ACTIVITY_BAR(g_object_new(TYPE_ACTIVITY_BAR, "orientation", GTK_ORIENTATION_HORIZONTAL,
       NULL));
00096         if (bar)
00097                 activity_bar_set_ready(bar);
00098
00099         return bar;
00100 }
00101
00102 void activity_bar_set_ready(ActivityBar *bar)
00103 {
00104         gtk_label_set_text(GTK_LABEL(bar->label), _("Ready"));
00105         gtk_spinner_stop(GTK_SPINNER(bar->spinner));
00106 }
00107
00108 void activity_bar_set_busy(ActivityBar *bar, const char *text)
00109 {
00110         gtk_label_set_text(GTK_LABEL(bar->label), (text ? text : _("Working...")));
00111         gtk_spinner_start(GTK_SPINNER(bar->spinner));
```

```
00112 }
00113
00114
```

13.100 conv-settings-dialog.c File Reference

Implementation of the setting dialog.

```
#include <gds-render/widgets/conv-settings-dialog.h>
#include <glib/gi18n.h>
```

Include dependency graph for conv-settings-dialog.c:

Data Structures

- struct _RendererSettingsDialog

Enumerations

- enum { PROP_CELL_NAME = 1 , PROP_COUNT }

Functions

- static void renderer_settings_dialog_set_property (GObject *object, guint property_id, const GValue *value, GParamSpec *pspec)
- static void renderer_settings_dialog_get_property (GObject *object, guint property_id, GValue *value, GParamSpec *pspec)
- static void renderer_settings_dialog_class_init (RendererSettingsDialogClass *klass)
- static void show_tex_options (RendererSettingsDialog *self)
- static void hide_tex_options (RendererSettingsDialog *self)
- static void latex_render_callback (GtkToggleButton *radio, RendererSettingsDialog *dialog)
- static gboolean shape_drawer_drawing_callback (GtkWidget *widget, cairo_t *cr, gpointer data)
- static double convert_number_to_engineering (double input, const char **out_prefix)
- static void renderer_settings_dialog_update_labels (RendererSettingsDialog *self)
- static void scale_value_changed (GtkRange *range, gpointer user_data)
- static void renderer_settings_dialog_init (RendererSettingsDialog *self)
- RendererSettingsDialog * renderer_settings_dialog_new (GtkWindow *parent)

  Create a new RedererSettingsDialog GObject.
- void renderer_settings_dialog_get_settings (RendererSettingsDialog *dialog, struct render_settings *settings)

  Get the settings configured in the dialog.
- G_END_DECLS void renderer_settings_dialog_set_settings (RendererSettingsDialog *dialog, struct render_settings *settings)

  Apply settings to dialog.
- void renderer_settings_dialog_set_cell_width (RendererSettingsDialog *dialog, unsigned int width)

  renderer_settings_dialog_set_cell_width Set width for rendered cell
- void renderer_settings_dialog_set_cell_height (RendererSettingsDialog *dialog, unsigned int height)

  renderer_settings_dialog_set_cell_height Set height for rendered cell
- void renderer_settings_dialog_set_database_unit_scale (RendererSettingsDialog *dialog, double unit_in_↩ meters)

  renderer_settings_dialog_set_database_unit_scale Set database scale

Variables

- static GParamSpec * properties [PROP_COUNT]

13.100.1 Detailed Description

Implementation of the setting dialog.

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file conv-settings-dialog.c.

13.101 conv-settings-dialog.c

Go to the documentation of this file.

```
00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018  Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter.  If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00032 #include <gds-render/widgets/conv-settings-dialog.h>
00033 #include <glib/gi18n.h>
00034
00035 struct  _RendererSettingsDialog {
00036                 GtkDialog parent;
00037                 /* Private loot */
00038                 GtkWidget *radio_latex;
00039                 GtkWidget *radio_cairo_pdf;
00040                 GtkWidget *radio_cairo_svg;
00041                 GtkWidget *scale;
00042                 GtkWidget *layer_check;
00043                 GtkWidget *standalone_check;
00044                 GtkDrawingArea *shape_drawing;
00045                 GtkLabel *x_label;
00046                 GtkLabel *y_label;
00047
00048                 GtkLabel *x_output_label;
00049                 GtkLabel *y_output_label;
00050
00051                 unsigned int cell_height;
00052                 unsigned int cell_width;
00053                 double unit_in_meters;
00054 };
00055
00056 G_DEFINE_TYPE(RendererSettingsDialog, renderer_settings_dialog, GTK_TYPE_DIALOG)
00057
00058 enum {
00059         PROP_CELL_NAME = 1,
00060         PROP_COUNT
00061 };
00062
00063 static GParamSpec *properties[PROP_COUNT];
00064
00065 static void renderer_settings_dialog_set_property(GObject *object, guint property_id,
00066                                                   const GValue *value, GParamSpec *pspec)
```

```
00067 {
00068         const gchar *title = NULL;
00069 
00070         switch (property_id) {
00071         case PROP_CELL_NAME:
00072                 title = g_value_get_string(value);
00073                 if (title)
00074                         gtk_window_set_title(GTK_WINDOW(object), title);
00075                 break;
00076         default:
00077                 G_OBJECT_WARN_INVALID_PROPERTY_ID(object, property_id, pspec);
00078                 break;
00079         }
00080 }
00081 
00082 static void renderer_settings_dialog_get_property(GObject *object, guint property_id,
00083                                                    GValue *value, GParamSpec *pspec)
00084 {
00085         const gchar *title;
00086 
00087         switch (property_id) {
00088         case PROP_CELL_NAME:
00089                 title = gtk_window_get_title(GTK_WINDOW(object));
00090                 g_value_set_string(value, title);
00091                 break;
00092         default:
00093                 G_OBJECT_WARN_INVALID_PROPERTY_ID(object, property_id, pspec);
00094                 break;
00095         }
00096 }
00097 
00098 static void renderer_settings_dialog_class_init(RendererSettingsDialogClass *klass)
00099 {
00100         GObjectClass *oclass = G_OBJECT_CLASS(klass);
00101 
00102         /* Override virtual functions */
00103         oclass->set_property = renderer_settings_dialog_set_property;
00104         oclass->get_property = renderer_settings_dialog_get_property;
00105 
00106         properties[PROP_CELL_NAME] = g_param_spec_string(N_("cell-name"),
00107                                                          N_("cell-name"),
00108                                                          N_("Cell name to be displayed in header
      bar"),
00109                                                          "",
00110                                                          G_PARAM_READWRITE);
00111         g_object_class_install_properties(oclass, PROP_COUNT, properties);
00112 }
00113 
00114 static void show_tex_options(RendererSettingsDialog *self)
00115 {
00116         gtk_widget_show(self->layer_check);
00117         gtk_widget_show(self->standalone_check);
00118 
00119 }
00120 
00121 static void hide_tex_options(RendererSettingsDialog *self)
00122 {
00123         gtk_widget_hide(self->layer_check);
00124         gtk_widget_hide(self->standalone_check);
00125 }
00126 
00127 static void latex_render_callback(GtkToggleButton *radio, RendererSettingsDialog *dialog)
00128 {
00129         if (gtk_toggle_button_get_active(radio))
00130                 show_tex_options(dialog);
00131         else
00132                 hide_tex_options(dialog);
00133 }
00134 
00135 static gboolean shape_drawer_drawing_callback(GtkWidget *widget, cairo_t *cr, gpointer data)
00136 {
00137         int width;
00138         int height;
00139         GtkStyleContext *style_context;
00140         GdkRGBA foreground_color;
00141         RendererSettingsDialog *dialog = (RendererSettingsDialog *)data;
00142         double usable_width;
00143         double usable_height;
00144         double height_scale;
00145         double width_scale;
00146         double final_scale_value;
00147 
00148         style_context = gtk_widget_get_style_context(widget);
00149         width = gtk_widget_get_allocated_width(widget);
00150         height = gtk_widget_get_allocated_height(widget);
00151 
00152         gtk_render_background(style_context, cr, 0, 0, width, height);
```

```
00153
00154         gtk_style_context_get_color(style_context, gtk_style_context_get_state(style_context),
00155                                     &foreground_color);
00156
00157         gdk_cairo_set_source_rgba(cr, &foreground_color);
00158
00159         cairo_save(cr);
00160
00161         /* Tranform coordiante system */
00162         cairo_scale(cr, 1, -1);
00163         cairo_translate(cr, (double)width/2.0, -(double)height/2.0);
00164
00165         /* Define usable drawing area */
00166         usable_width = (0.95*(double)width) - 15.0;
00167         usable_height = (0.95*(double)height) - 15.0;
00168
00169         width_scale = usable_width/(double)dialog->cell_width;
00170         height_scale = usable_height/(double)dialog->cell_height;
00171
00172         final_scale_value = (width_scale < height_scale ? width_scale : height_scale);
00173
00174         cairo_rectangle(cr,
00175                         -(double)dialog->cell_width * final_scale_value / 2.0,
00176                         -(double)dialog->cell_height * final_scale_value / 2.0,
00177                         (double)dialog->cell_width * final_scale_value,
00178                         (double)dialog->cell_height * final_scale_value);
00179         cairo_stroke(cr);
00180         cairo_restore(cr);
00181
00182         return FALSE;
00183 }
00184
00185 static double convert_number_to_engineering(double input, const char **out_prefix)
00186 {
00187         const char *selected_prefix = NULL;
00188         double return_val = 0.0;
00189         int idx;
00190         static const char * const prefixes[] = {"y", "z", "a", "f", "p", "n", "u", "m", "c", "d", /* <
      1 */
00191                                                 "", /* 1 */
00192                                                 "h", "k", "M", "G", "T", "P", "E", "Z", "Y"}; /* > 1
      */
00193         static const double scale[] = {1E-24, 1E-21, 1E-18, 1E-15, 1E-12, 1E-9, 1E-6, 1E-3, 1E-2,
      1E-1,
00194                                        1,
00195                                        1E2, 1E3, 1E6, 1E9, 1E12, 1E15, 1E18, 1E21, 1E24};
00196         const int prefix_count = (int)(sizeof(prefixes)/sizeof(char *));
00197
00198         /* If pointer is invalid, return NaN */
00199         if (!out_prefix)
00200                 return (0.0 / 0.0);
00201
00202         /* Start with the 2nd smallest prefix */
00203         for (idx = 1; idx < prefix_count; idx++) {
00204                 if (input < scale[idx]) {
00205                         /* This prefix is bigger than the number. Take the previous one */
00206                         selected_prefix = prefixes[idx-1];
00207                         return_val = input / scale[idx-1];
00208                         break;
00209                 }
00210         }
00211
00212         /* Check if prefix was set by loop. Else take the largest in the list */
00213         if (selected_prefix == NULL) {
00214                 selected_prefix = prefixes[prefix_count-1];
00215                 return_val = input / scale[prefix_count-1];
00216         }
00217
00218         if (out_prefix)
00219                 *out_prefix = selected_prefix;
00220
00221         return return_val;
00222 }
00223
00224 static void renderer_settings_dialog_update_labels(RendererSettingsDialog *self)
00225 {
00226         char default_buff[100];
00227         double scale;
00228         double width_meters;
00229         double height_meters;
00230         double width_engineering;
00231         const char *width_prefix;
00232         double height_engineering;
00233         const char *height_prefix;
00234
00235         if (!self)
00236                 return;
```

```
00237
00238         width_meters = (double)self->cell_width * self->unit_in_meters;
00239         height_meters = (double)self->cell_height * self->unit_in_meters;
00240
00241         width_engineering = convert_number_to_engineering(width_meters, &width_prefix);
00242         height_engineering = convert_number_to_engineering(height_meters, &height_prefix);
00243
00244         snprintf(default_buff, sizeof(default_buff), _("Width: %.3lf %sm"), width_engineering,
      width_prefix);
00245         gtk_label_set_text(self->x_label, default_buff);
00246         snprintf(default_buff, sizeof(default_buff), _("Height: %.3lf %sm"), height_engineering,
      height_prefix);
00247         gtk_label_set_text(self->y_label, default_buff);
00248
00249         scale = gtk_range_get_value(GTK_RANGE(self->scale));
00250
00251         /* Set the pixel sizes */
00252         snprintf(default_buff, sizeof(default_buff), _("Output Width: %u px"),
00253                  (unsigned int)((double)self->cell_width / scale));
00254         gtk_label_set_text(self->x_output_label, default_buff);
00255         snprintf(default_buff, sizeof(default_buff), _("Output Height: %u px"),
00256                  (unsigned int)((double)self->cell_height / scale));
00257         gtk_label_set_text(self->y_output_label, default_buff);
00258 }
00259
00260 static void scale_value_changed(GtkRange *range, gpointer user_data)
00261 {
00262         (void)range;
00263         RendererSettingsDialog *dialog;
00264
00265         dialog = RENDERER_SETTINGS_DIALOG(user_data);
00266         renderer_settings_dialog_update_labels(dialog);
00267 }
00268
00269 static void renderer_settings_dialog_init(RendererSettingsDialog *self)
00270 {
00271         GtkBuilder *builder;
00272         GtkWidget *box;
00273         GtkDialog *dialog;
00274
00275         dialog = &self->parent;
00276
00277         builder = gtk_builder_new_from_resource("/gui/dialog.glade");
00278         box = GTK_WIDGET(gtk_builder_get_object(builder, "dialog-box"));
00279         self->radio_latex = GTK_WIDGET(gtk_builder_get_object(builder, "latex-radio"));
00280         self->radio_cairo_pdf = GTK_WIDGET(gtk_builder_get_object(builder, "cairo-pdf-radio"));
00281         self->radio_cairo_svg = GTK_WIDGET(gtk_builder_get_object(builder, "cairo-svg-radio"));
00282         self->scale = GTK_WIDGET(gtk_builder_get_object(builder, "dialog-scale"));
00283         self->standalone_check = GTK_WIDGET(gtk_builder_get_object(builder, "standalone-check"));
00284         self->layer_check = GTK_WIDGET(gtk_builder_get_object(builder, "layer-check"));
00285         self->shape_drawing = GTK_DRAWING_AREA(gtk_builder_get_object(builder, "shape-drawer"));
00286         self->x_label = GTK_LABEL(gtk_builder_get_object(builder, "x-label"));
00287         self->y_label = GTK_LABEL(gtk_builder_get_object(builder, "y-label"));
00288         self->x_output_label = GTK_LABEL(gtk_builder_get_object(builder, "x-output-label"));
00289         self->y_output_label = GTK_LABEL(gtk_builder_get_object(builder, "y-output-label"));
00290
00291         gtk_dialog_add_buttons(dialog, _("Cancel"), GTK_RESPONSE_CANCEL, _("OK"), GTK_RESPONSE_OK,
      NULL);
00292         gtk_container_add(GTK_CONTAINER(gtk_dialog_get_content_area(dialog)), box);
00293         gtk_window_set_title(GTK_WINDOW(self), _("Renderer Settings"));
00294
00295         g_signal_connect(self->radio_latex, "toggled", G_CALLBACK(latex_render_callback),
      (gpointer)self);
00296         g_signal_connect(G_OBJECT(self->shape_drawing),
00297                          "draw", G_CALLBACK(shape_drawer_drawing_callback), (gpointer)self);
00298
00299         g_signal_connect(self->scale, "value-changed", G_CALLBACK(scale_value_changed),
      (gpointer)self);
00300
00301         /* Default values */
00302         self->cell_width = 1;
00303         self->cell_height = 1;
00304         self->unit_in_meters = 1E-6;
00305         renderer_settings_dialog_update_labels(self);
00306
00307         g_object_unref(builder);
00308 }
00309
00310 RendererSettingsDialog *renderer_settings_dialog_new(GtkWindow *parent)
00311 {
00312         RendererSettingsDialog *res;
00313
00314         res = RENDERER_SETTINGS_DIALOG(g_object_new(RENDERER_TYPE_SETTINGS_DIALOG, NULL));
00315         if (res && parent)
00316                 gtk_window_set_transient_for(GTK_WINDOW(res), parent);
00317
00318         return res;
```

```
00319 }
00320
00321 void renderer_settings_dialog_get_settings(RendererSettingsDialog *dialog, struct render_settings
      *settings)
00322 {
00323
00324         if (!settings || !dialog)
00325                 return;
00326         settings->scale = gtk_range_get_value(GTK_RANGE(dialog->scale));
00327
00328         /* Get active radio button selection */
00329         if (gtk_toggle_button_get_active(GTK_TOGGLE_BUTTON(dialog->radio_latex)) == TRUE)
00330                 settings->renderer = RENDERER_LATEX_TIKZ;
00331         else if (gtk_toggle_button_get_active(GTK_TOGGLE_BUTTON(dialog->radio_cairo_pdf)) == TRUE)
00332                 settings->renderer = RENDERER_CAIROGRAPHICS_PDF;
00333         else if (gtk_toggle_button_get_active(GTK_TOGGLE_BUTTON(dialog->radio_cairo_svg)) == TRUE)
00334                 settings->renderer = RENDERER_CAIROGRAPHICS_SVG;
00335
00336         settings->tex_pdf_layers =
      gtk_toggle_button_get_active(GTK_TOGGLE_BUTTON(dialog->layer_check));
00337         settings->tex_standalone =
      gtk_toggle_button_get_active(GTK_TOGGLE_BUTTON(dialog->standalone_check));
00338 }
00339
00340 void renderer_settings_dialog_set_settings(RendererSettingsDialog *dialog, struct render_settings
      *settings)
00341 {
00342         if (!settings || !dialog)
00343                 return;
00344
00345         gtk_range_set_value(GTK_RANGE(dialog->scale), settings->scale);
00346         gtk_toggle_button_set_active(GTK_TOGGLE_BUTTON(dialog->layer_check),
      settings->tex_pdf_layers);
00347         gtk_toggle_button_set_active(GTK_TOGGLE_BUTTON(dialog->standalone_check),
      settings->tex_standalone);
00348
00349         switch (settings->renderer) {
00350         case RENDERER_LATEX_TIKZ:
00351                 gtk_toggle_button_set_active(GTK_TOGGLE_BUTTON(dialog->radio_latex), TRUE);
00352                 show_tex_options(dialog);
00353                 break;
00354         case RENDERER_CAIROGRAPHICS_PDF:
00355                 hide_tex_options(dialog);
00356                 gtk_toggle_button_set_active(GTK_TOGGLE_BUTTON(dialog->radio_cairo_pdf), TRUE);
00357                 break;
00358         case RENDERER_CAIROGRAPHICS_SVG:
00359                 hide_tex_options(dialog);
00360                 gtk_toggle_button_set_active(GTK_TOGGLE_BUTTON(dialog->radio_cairo_svg), TRUE);
00361                 break;
00362         }
00363 }
00364
00365 void renderer_settings_dialog_set_cell_width(RendererSettingsDialog *dialog, unsigned int width)
00366 {
00367         if (!dialog)
00368                 return;
00369
00370         if (width == 0)
00371                 width = 1;
00372
00373         dialog->cell_width = width;
00374         renderer_settings_dialog_update_labels(dialog);
00375 }
00376
00377 void renderer_settings_dialog_set_cell_height(RendererSettingsDialog *dialog, unsigned int height)
00378 {
00379         if (!dialog)
00380                 return;
00381
00382         if (height == 0)
00383                 height = 1;
00384
00385         dialog->cell_height = height;
00386         renderer_settings_dialog_update_labels(dialog);
00387 }
00388
00389 void renderer_settings_dialog_set_database_unit_scale(RendererSettingsDialog *dialog, double
      unit_in_meters)
00390 {
00391         if (!dialog)
00392                 return;
00393
00394         if (unit_in_meters < 0)
00395                 unit_in_meters *= -1;
00396
00397         dialog->unit_in_meters = unit_in_meters;
00398         renderer_settings_dialog_update_labels(dialog);
```

```
00399 }
00400
```

13.102 layer-element.c File Reference

Implementation of the layer element used for configuring layer colors etc.

```
#include <gds-render/widgets/layer-element.h>
#include <glib/gi18n.h>
```

Include dependency graph for layer-element.c:

Functions

- static void layer_element_dispose (GObject *obj)
- static void layer_element_constructed (GObject *obj)
- static void layer_element_class_init (LayerElementClass *klass)
- static void layer_element_init (LayerElement *self)
- GtkWidget * layer_element_new (void)

  Create new layer element object.
- const char * layer_element_get_name (LayerElement *elem)

  get name of the layer
- void layer_element_set_name (LayerElement *elem, const char *name)

  layer_element_set_name
- void layer_element_set_layer (LayerElement *elem, int layer)

  Set layer number for this layer.
- int layer_element_get_layer (LayerElement *elem)

  Get layer number.
- void layer_element_set_export (LayerElement *elem, gboolean export)

  Set export flag for this layer.
- gboolean layer_element_get_export (LayerElement *elem)

  Get export flag of layer.
- void layer_element_get_color (LayerElement *elem, GdkRGBA *rgba)

  Get color of layer.
- void layer_element_set_color (LayerElement *elem, GdkRGBA *rgba)

  Set color of layer.
- void layer_element_set_dnd_callbacks (LayerElement *elem, struct layer_element_dnd_data *data)

  Setup drag and drop of `elem` for use in the LayerSelector.

13.102.1 Detailed Description

Implementation of the layer element used for configuring layer colors etc.

Author

Mario Hüttel `mario.huettel@gmx.net`

Definition in file layer-element.c.

13.103 layer-element.c

Go to the documentation of this file.

```
00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018  Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter.  If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00020 /*
00021  * The drag and drop implementation is adapted from
00022  * https://gitlab.gnome.org/GNOME/gtk/blob/gtk-3-22/tests/testlist3.c
00023  *
00024  * Thanks to the GTK3 people for creating these examples.
00025  */
00026
00039 #include <gds-render/widgets/layer-element.h>
00040 #include <glib/gi18n.h>
00041
00042 G_DEFINE_TYPE(LayerElement, layer_element, GTK_TYPE_LIST_BOX_ROW)
00043
00044 static void layer_element_dispose(GObject *obj)
00045 {
00046         /* destroy parent container. This destroys all widgets inside */
00047         G_OBJECT_CLASS(layer_element_parent_class)->dispose(obj);
00048 }
00049
00050 static void layer_element_constructed(GObject *obj)
00051 {
00052         G_OBJECT_CLASS(layer_element_parent_class)->constructed(obj);
00053 }
00054
00055 static void layer_element_class_init(LayerElementClass *klass)
00056 {
00057         GObjectClass *oclass = G_OBJECT_CLASS(klass);
00058
00059         oclass->dispose = layer_element_dispose;
00060         oclass->constructed = layer_element_constructed;
00061 }
00062
00063 static void layer_element_init(LayerElement *self)
00064 {
00065         GtkBuilder *builder;
00066         GtkWidget *glade_box;
00067
00068         builder = gtk_builder_new_from_resource("/gui/layer-widget.glade");
00069         glade_box = GTK_WIDGET(gtk_builder_get_object(builder, "box"));
00070         gtk_container_add(GTK_CONTAINER(self), glade_box);
00071
00072         /* Get Elements */
00073         self->priv.color = GTK_COLOR_BUTTON(gtk_builder_get_object(builder, "color"));
00074         self->priv.export = GTK_CHECK_BUTTON(gtk_builder_get_object(builder, "export"));
00075         self->priv.layer = GTK_LABEL(gtk_builder_get_object(builder, "layer"));
00076         self->priv.name = GTK_ENTRY(gtk_builder_get_object(builder, "entry"));
00077         self->priv.event_handle = GTK_EVENT_BOX(gtk_builder_get_object(builder, "event-box"));
00078
00079         g_object_unref(builder);
00080 }
00081
00082 GtkWidget *layer_element_new(void)
00083 {
00084         return GTK_WIDGET(g_object_new(TYPE_LAYER_ELEMENT, NULL));
00085 }
00086
00087 const char *layer_element_get_name(LayerElement *elem)
00088 {
00089         return gtk_entry_get_text(elem->priv.name);
00090 }
00091
00092 void layer_element_set_name(LayerElement *elem, const char *name)
00093 {
00094         gtk_entry_set_text(elem->priv.name, name);
```

```
00095 }
00096
00097 void layer_element_set_layer(LayerElement *elem, int layer)
00098 {
00099         GString *string;
00100
00101         string = g_string_new_len(NULL, 100);
00102         g_string_printf(string, _("Layer: %d"), layer);
00103         gtk_label_set_text(elem->priv.layer, (const gchar *)string->str);
00104         elem->priv.layer_num = layer;
00105         g_string_free(string, TRUE);
00106 }
00107
00108 int layer_element_get_layer(LayerElement *elem)
00109 {
00110         return elem->priv.layer_num;
00111 }
00112
00113 void layer_element_set_export(LayerElement *elem, gboolean export)
00114 {
00115         gtk_toggle_button_set_active(GTK_TOGGLE_BUTTON(elem->priv.export), export);
00116 }
00117
00118 gboolean layer_element_get_export(LayerElement *elem)
00119 {
00120         return gtk_toggle_button_get_active(GTK_TOGGLE_BUTTON(elem->priv.export));
00121 }
00122
00123 void layer_element_get_color(LayerElement *elem, GdkRGBA *rgba)
00124 {
00125         if (!rgba)
00126                 return;
00127
00128         gtk_color_chooser_get_rgba(GTK_COLOR_CHOOSER(elem->priv.color), rgba);
00129 }
00130
00131 void layer_element_set_color(LayerElement *elem, GdkRGBA *rgba)
00132 {
00133         if (!elem || !rgba)
00134                 return;
00135
00136         gtk_color_chooser_set_rgba(GTK_COLOR_CHOOSER(elem->priv.color), rgba);
00137 }
00138
00139 void layer_element_set_dnd_callbacks(LayerElement *elem, struct layer_element_dnd_data *data)
00140 {
00141         if (!elem || !data)
00142                 return;
00143
00144         /* Setup drag and drop */
00145         gtk_style_context_add_class(gtk_widget_get_style_context(GTK_WIDGET(elem)), "row");
00146         gtk_drag_source_set(GTK_WIDGET(elem->priv.event_handle), GDK_BUTTON1_MASK,
00147                             data->entries, data->entry_count, GDK_ACTION_MOVE);
00148         g_signal_connect(elem->priv.event_handle, "drag-begin", G_CALLBACK(data->drag_begin), NULL);
00149         g_signal_connect(elem->priv.event_handle, "drag-data-get", G_CALLBACK(data->drag_data_get),
      NULL);
00150         g_signal_connect(elem->priv.event_handle, "drag-end", G_CALLBACK(data->drag_end), NULL);
00151
00152 }
00153
```

Index