Layers and Files

Gerbonara currently supports three file types: RS-274-X Gerber as *specified by Ucamco <https://www.ucamco.com/en/gerber>_* through **GerberFile**, Excellon/XNC through **ExcellonFile**, and IPC-356 netlists through **Netlist**.

Usually, a PCB is sent to a manufacturer as a bundle of several of these files. Such a bundle of files (each of which is either a **GerberFile** or an **ExcellonFile**) is represented by **LayerStack**. **LayerStack** contains logic to automatcally recognize a wide variety of CAD tools from file name and syntactic hints, and can automatically match all files in a folder to their appropriate layers.

CamFile is the common base class for all layer types.

class gerbonara.cam.**CamFile**(*original_path=None*, *layer_name=None*, *import_settings=None*)

Base class for all layer classes (**GerberFile**, **ExcellonFile**, and **Netlist**).

Provides some common functions such as **to_svg()**.

bounding_box(*unit=<LengthUnit millimeter>*, *default=None*)

Calculate the axis-aligned bounding box of file. Returns value given by the `default` argument when the file is empty. This file calculates the accurate bounding box, even for features such as arcs.

> Note:
>
> Gerbonara returns bounding boxes as a `(bottom_left, top_right)` tuple of points, not in the `((min_x, max_x), (min_y, max_y))` format used by pcb-tools.

Parameters: **unit** – **LengthUnit** or str (`'mm'` or `'inch'`). Which unit to return results in. Default: mm
Returns: `((x_min, y_min), (x_max, y_max))` tuple of floats.
Return type: tuple

merge(*other*)

Merge `other` into `self`, i.e. add all objects that are in `other` to `self`. This resets **import_settings** and **generator**. Units and other file-specific settings are handled automatically.

offset(*x=0*, *y=0*, *unit=<LengthUnit millimeter>*)

Add a coordinate offset to this file. The offset is given in Gerber/Excellon coordinates, so the Y axis points upwards. Gerbonara does not use the poorly-supported Gerber file offset options, but instead actually changes the coordinates of every object in the file. This means that you can load the generated file with any Gerber viewer, and things should just work.

Parameters:
- **x** (*float*) – X offset
- **y** (*float*) – Y offset
- **unit** – **LengthUnit** or str (`'mm'` or `'inch'`). Unit `x` and `y` are passed in. Default: mm

rotate(*angle*, *cx=0*, *cy=0*, *unit=<LengthUnit millimeter>*)

Apply a rotation to this file. The center of rotation is given in Gerber/Excellon coordinates, so the Y axis points upwards. Gerbonara does not use the poorly-supported Gerber file rotation options, but instead actually changes the coordinates and rotation of every object in the file. This means that you can load the generated file with any Gerber viewer, and things should just work.

Note that when rotating certain apertures, they will be automatically converted to aperture macros during export since the standard apertures do not support rotation by spec. This is the same way most CAD packages deal with this issue so it should work with most Gerber viewers.

Parameters:
- **angle** (*float*) – Rotation angle in radians, *clockwise*.
- **cx** (*float*) – Center of rotation X coordinate
- **cy** (*float*) – Center of rotation Y coordinate
- **unit** – **`LengthUnit`** or str (`'mm'` or `'inch'`). Unit `cx` and `cy` are passed in. Default: mm

`scale`(*factor*, *unit=<LengthUnit millimeter>*)

Scale all objects in this file by the given factor. Only uniform scaling using a single factor in both directions is supported as for both Gerber and Excellon files, nonuniform scaling would distort circular flashes, which would lead to garbage results.

Parameters:
- **factor** (*float*) – Scale factor
- **unit** – **`LengthUnit`** or str (`'mm'` or `'inch'`). Unit `cx` and `cy` are passed in. Default: mm

`size`(*unit=<LengthUnit millimeter>*)

Get the dimensions of the file's axis-aligned bounding box, i.e. the difference in x- and y-direction between the minimum x and y coordinates and the maximum x and y coordinates.

Parameters: **unit** – **`LengthUnit`** or str (`'mm'` or `'inch'`). Which unit to return results in. Default: mm
Returns: `(w, h)` tuple of floats.
Return type: tuple

`to_excellon`()

Convert to a **`ExcellonFile`**. Returns `self` if it already is one.

`to_gerber`()

Convert to a **`GerberFile`**. Returns `self` if it already is one.

property `generator`

Return our best guess as to which software produced this file.

Returns: a str like `'kicad'` or `'allegro'`

property `is_empty`

Check if there are any objects in this file.

class gerbonara.rs274x.`GerberFile`(*objects=None*, *comments=None*, *import_settings=None*, *original_path=None*, *generator_hints=None*, *layer_hints=None*, *file_attrs=None*)

A single gerber file.

Variables:
- **objects** – List of objects in this Gerber file. All elements must be subclasses of **`GraphicObject`**.
- **comments** – List of string with textual comments in the source Gerber file. These are not saved by default, but when you call **`GerberFile.save()`** with `drop_comments=False`, the contents of this list will be included as comments at the top of the output file.
- **generator_hints** – List of strings indicating which EDA tool generated this file. Hints are added to this list during file parsing whenever the parser encounters an idiosyncratic file format variation.
- **import_settings** – File format settings used in the original file. This can be empty if this **`GerberFile`** was generated programatically.
- **layer_hints** – Similar to `generator_hints`, this is a list containing hints which layer type this file could belong to. Usually, this will be empty, but some EDA tools automatically include layer information inside tool-specific comments in the Gerber files they generate.

- **file_attrs** – List of strings with Gerber X3 file attributes. Each list item corresponds to one file attribute.

apertures()

Iterate through all apertures in this layer.

dedup_apertures(*settings=None*)

Merge all apertures and aperture macros in this layer that result in the same Gerber definition under the given :py:class:~.FileSettings:.

When no explicit settings are given, uses Gerbonara's default settings.

Parameters: **settings** – settings under which to de-duplicate the apertures.

classmethod **from_string**(*data, enable_include_dir=None, filename=None, override_settings=None*)

Parse given string as Gerber file content. For the meaning of the parameters, see **open()**.

invert_polarity()

Invert the polarity (color) of each object in this file.

map_apertures(*map_or_callable, cache=True*)

Replace all apertures in all objects in this layer according to the given map or callable.

When a map is passed, apertures that are not in the map are left alone. When a callable is given, it is called with the old aperture as its argument.

Parameters:
- **map_or_callable** – A dict-like object, or a callable mapping old to new apertures
- **cache** – When True (default) and a callable is passed, caches the output of callable, only calling it once for each old aperture.

merge(*other, mode='above', keep_settings=False*)

Merge `other` into `self`, i.e. add all objects that are in `other` to `self`. This resets **import_settings** and **generator**. Units and other file-specific settings are handled automatically.

Parameters: **mode** – One of the strings **"above"** (default) or **"below"**, specifying whether the other layer's objects will be placed above this layer's objects (placing them towards the end of the file), or below this layer's objects (placing them towards the beginning of the file). This setting is only relevant when there are overlapping objects of different polarity, otherwise the rendered result will be the same either way.

offset(*dx=0, dy=0, unit=<LengthUnit millimeter>*)

Add a coordinate offset to this file. The offset is given in Gerber/Excellon coordinates, so the Y axis points upwards. Gerbonara does not use the poorly-supported Gerber file offset options, but instead actually changes the coordinates of every object in the file. This means that you can load the generated file with any Gerber viewer, and things should just work.

Parameters:
- **x** (*float*) – X offset
- **y** (*float*) – Y offset
- **unit** – **LengthUnit** or str (`'mm'` or `'inch'`). Unit `x` and `y` are passed in. Default: mm

classmethod **open**(*filename, enable_includes=False, enable_include_dir=None, override_settings=None*)

Load a Gerber file from the file system. The Gerber standard contains this wonderful and totally not insecure "include file" setting. We disable it by default and do not parse Gerber includes because a) nobody actually uses them, and b) they're a bad idea from a security point of view. In case you actually want these, you can enable them by setting `enable_includes=True`.

Parameters:
- **filename** – str or **pathlib.Path**
- **enable_includes** (*bool*) – Enable Gerber `IF` statement includes (default *off*, recommended *off*)
- **enable_include_dir** – str or **pathlib.Path**. Override base dir for include files.

Return type: **GerberFile**

rotate(*angle: radian, cx=0, cy=0, unit=<LengthUnit millimeter>*)

Apply a rotation to this file. The center of rotation is given in Gerber/Excellon coordinates, so the Y axis points upwards. Gerbonara does not use the poorly-supported Gerber file rotation options, but instead actually changes the coordinates and rotation of every object in the file. This means that you can load the generated file with any Gerber viewer, and things should just work.

Note that when rotating certain apertures, they will be automatically converted to aperture macros during export since the standard apertures do not support rotation by spec. This is the same way most CAD packages deal with this issue so it should work with most Gerber viewers.

Parameters:
- **angle** (*float*) – Rotation angle in radians, *clockwise*.
- **cx** (*float*) – Center of rotation X coordinate
- **cy** (*float*) – Center of rotation Y coordinate
- **unit** – **LengthUnit** or str (`'mm'` or `'inch'`). Unit `cx` and `cy` are passed in. Default: mm

save(*filename, settings=None, drop_comments=True*)

Save this Gerber file to the file system. See **generate_gerber()** for the meaning of the arguments.

scale(*factor, unit=<LengthUnit millimeter>*)

Scale all objects in this file by the given factor. Only uniform scaling using a single factor in both directions is supported as for both Gerber and Excellon files, nonuniform scaling would distort circular flashes, which would lead to garbage results.

Parameters:
- **factor** (*float*) – Scale factor
- **unit** – **LengthUnit** or str (`'mm'` or `'inch'`). Unit `cx` and `cy` are passed in. Default: mm

to_excellon(*plated=None, errors='raise'*)

Convert this excellon file into a **ExcellonFile**. This will convert interpolated lines into slots, and circular aperture flashes into holes. Other features such as `G36` polygons or flashes with non-circular apertures will result in a **ValueError**. You can, of course, programmatically remove such features from a **GerberFile** before conversion.

to_gerber(*errors='raise'*)

Counterpart to **to_gerber()**. Does nothing and returns **self**.

write_to_bytes(*settings=None, drop_comments=True*)

Export to Gerber format. Uses either the file's original settings or sane default settings if you don't give any.

Parameters:
- **settings** (*FileSettings*) – override export settings.
- **drop_comments** (*bool*) – If true, do not write comments to output file. This defaults to true because otherwise there is a risk that Gerbonara does not consider some obscure magic comment semantically meaningful while some other Excellon viewer might still parse it.

Return type: str

class gerbonara.excellon.**ExcellonFile**(*objects=None, comments=None, import_settings=None, original_path=None, generator_hints=None*)

Excellon drill file.

An Excellon file can contain both drills and milled slots. Drills are represented by **Flash** instances with their aperture set to the special **ExcellonDrill** aperture class. Drills can be plated or nonplated. This information is stored in the **ExcellonTool**. Both can co-exist in the same file, and some CAD tools even export files like this. **LayerStack** contains functions to convert between a single drill file with mixed plated and nonplated holes and one with separate drill files for each. Best practice is to have separate drill files for slots, nonplated holes, and plated holes, because the board house will produce all three in three separate processes anyway, and also because there is no standardized way to represent plating in Excellon files. Gerbonara uses Altium's convention for this, which uses a magic comment before the tool definition.

append(*obj_or_comment*)

Add a **GraphicObject** or a comment (str) to this file.

drill_sizes(*unit=<LengthUnit millimeter>*)

Return a sorted list of all tool diameters found in this file.

Parameters:	**unit** – **LengthUnit** or str (`'mm'` or `'inch'`). Unit to use for return values. Default: mm
Returns:	list of floats, sorted smallest to largest diameter.
Return type:	list

drills()

Return all drilled hole objects in this file.

Returns:	list of **Flash** instances
Return type:	list

classmethod **from_string**(*data, settings=None, filename=None, plated=None, external_tools=None*)

Parse the given string as an Excellon file. Note that often, Excellon files do not contain any information on which number format (integer/decimal places, zeros suppression) is used. In case Gerbonara cannot determine this with certainty, this function *will* error out. Use **open()** if you want Gerbonara to parse this metadata from the non-standardized text files many CAD packages produce in addition to drill files.

hit_count()

Calculate the number of objects per tool.

Return type:	collections.Counter

merge(*other, mode='ignored', keep_settings=False*)

Merge `other` into `self`, i.e. add all objects that are in `other` to `self`. This resets **import_settings** and **generator**. Units and other file-specific settings are handled automatically.

offset(*x=0, y=0, unit=<LengthUnit millimeter>*)

Add a coordinate offset to this file. The offset is given in Gerber/Excellon coordinates, so the Y axis points upwards. Gerbonara does not use the poorly-supported Gerber file offset options, but instead actually changes the coordinates of every object in the file. This means that you can load the generated file with any Gerber viewer, and things should just work.

Parameters:

- **x** (*float*) – X offset
- **y** (*float*) – Y offset
- **unit** – **LengthUnit** or str (`'mm'` or `'inch'`). Unit `x` and `y` are passed in. Default: mm

classmethod **open**(*filename, plated=None, settings=None, external_tools=None*)

Load an Excellon file from the file system.

Certain CAD tools do not put any information on decimal points into the actual excellon file, and instead put that information into a non-standard text file next to the excellon file. Using **`open()`** to open a file gives Gerbonara the opportunity to try to find this data. In contrast to pcb-tools, Gerbonara will raise an exception instead of producing garbage parsing results if it cannot determine the file format parameters with certainty.

> **Note:**
>
> This is preferred over loading Excellon from a str through **`from_string()`**.

Parameters:
- **filename** – `str` or `pathlib.Path`.
- **plated** (*bool*) – If given, set plating status of any tools in this file that have undefined plating. This is useful if you already know that this file contains only e.g. plated holes from contextual information such as the file name.
- **settings** (*FileSettings*) – Format settings to use. If None, try to auto-detect file settings.

`path_lengths(unit=<LengthUnit millimeter>)`

Calculate path lengths per tool.

This function only sums actual cut lengths, and ignores travel lengths that the tool is doing without cutting to get from one object to another. Travel lengths depend on the CAM program's path planning, which highly depends on panelization and other factors. Additionally, an EDA tool will not even attempt to minimize travel distance as that's not its job.

Parameters: **unit** – **`LengthUnit`** or str (`'mm'` or `'inch'`). Unit to use for return value. Default: mm
Returns: `{ tool: float(path length) }`
Rtype dict:

`rotate(angle, cx=0, cy=0, unit=<LengthUnit millimeter>)`

Apply a rotation to this file. The center of rotation is given in Gerber/Excellon coordinates, so the Y axis points upwards. Gerbonara does not use the poorly-supported Gerber file rotation options, but instead actually changes the coordinates and rotation of every object in the file. This means that you can load the generated file with any Gerber viewer, and things should just work.

Note that when rotating certain apertures, they will be automatically converted to aperture macros during export since the standard apertures do not support rotation by spec. This is the same way most CAD packages deal with this issue so it should work with most Gerber viewers.

Parameters:
- **angle** (*float*) – Rotation angle in radians, *clockwise*.
- **cx** (*float*) – Center of rotation X coordinate
- **cy** (*float*) – Center of rotation Y coordinate
- **unit** – **`LengthUnit`** or str (`'mm'` or `'inch'`). Unit `cx` and `cy` are passed in. Default: mm

`save(filename, settings=None, drop_comments=True)`

Save this Excellon file to the file system. See **`generate_excellon()`** for the meaning of the arguments.

`slots()`

Return all milled slot objects in this file.

Returns: list of **`Line`** or **`Arc`** instances
Return type: list

`split_by_plating()`

Split this file into two **`ExcellonFile`** instances, one containing all plated objects, and one containing all nonplated objects. In this function, objects with undefined plating are considered

nonplated.

> **Note:**
>
> This does not copy the objects, so modifications in either of the returned files may clobber the original file.

Returns: (nonplated_file, plated_file)
Return type: tuple

to_excellon(*plated=None*, *errors='raise'*)

Counterpart to **to_excellon()**. Does nothing and returns **self**.

to_gerber(*errros='raise'*)

Convert this excellon file into a **GerberFile**.

write_to_bytes(*settings=None*, *drop_comments=True*)

Export to Excellon format. This function always generates XNC, which is a well-defined subset of Excellon. Uses sane default settings if you don't give any.

Parameters: **drop_comments** (*bool*) – If true, do not write comments to output file. This defaults to true because otherwise there is a risk that Gerbonara does not consider some obscure magic comment semantically meaningful while some other Excellon viewer might still parse it.
Return type: str

property **generator**

Return our best guess as to which software produced this file.

Returns: a str like `'kicad'` or `'allegro'`

property **is_mixed_plating**

Test if there are multiple plating values used in this file.

property **is_nonplated**

Test if *all* holes or slots in this file are non-plated.

property **is_plated**

Test if *all* holes or slots in this file are plated.

property **is_plating_unknown**

Test if *all* holes or slots in this file have no known plating.

class gerbonara.ipc356.**Netlist**(*test_records=None*, *conductors=None*, *outlines=None*, *comments=None*, *adjacency=None*, *params=None*, *import_settings=None*, *original_path=None*, *generator_hints=None*)

merge(*other*, *our_prefix=None*, *their_prefix=None*)

Merge other netlist into this netlist. The respective net names are prefixed with the given prefixes (default: None). Garbles other.

offset(*dx=0*, *dy=0*, *unit=<LengthUnit millimeter>*)

Add a coordinate offset to this file. The offset is given in Gerber/Excellon coordinates, so the Y axis points upwards. Gerbonara does not use the poorly-supported Gerber file offset options, but instead actually changes the coordinates of every object in the file. This means that you can load the generated file with any Gerber viewer, and things should just work.

Parameters:
- **x** (*float*) – X offset
- **y** (*float*) – Y offset

- **unit** – **LengthUnit** or str (`'mm'` or `'inch'`). Unit `x` and `y` are passed in. Default: mm

rotate(*angle: radian, center=(0, 0), unit=<LengthUnit millimeter>*)

Apply a rotation to this file. The center of rotation is given in Gerber/Excellon coordinates, so the Y axis points upwards. Gerbonara does not use the poorly-supported Gerber file rotation options, but instead actually changes the coordinates and rotation of every object in the file. This means that you can load the generated file with any Gerber viewer, and things should just work.

Note that when rotating certain apertures, they will be automatically converted to aperture macros during export since the standard apertures do not support rotation by spec. This is the same way most CAD packages deal with this issue so it should work with most Gerber viewers.

Parameters:
- **angle** (*float*) – Rotation angle in radians, *clockwise*.
- **cx** (*float*) – Center of rotation X coordinate
- **cy** (*float*) – Center of rotation Y coordinate
- **unit** – **LengthUnit** or str (`'mm'` or `'inch'`). Unit `cx` and `cy` are passed in. Default: mm

class gerbonara.layers.**LayerStack**(*graphic_layers=None, drill_pth=None, drill_npth=None, drill_layers=(), netlist=None, board_name=None, original_path=None, was_zipped=False, generator=None, courtyard=False, fabrication=False, adhesive=False*)

LayerStack represents a set of Gerber files that describe different layers of the same board.

Variables:
- **graphic_layers** – **dict** mapping **(side, use)** tuples to the Gerber layers of the board. **side** can be one of **"top"**, **"bottom"**, **"mechanical"**, or a numbered internal layer such as **"inner2"**. **use** can be one of **"silk", :py:obj:`mask**, **paste** or **copper**. For internal layers, only **copper** is valid.
- **board_name** – Name of this board as parse from the input filenames, as a **str**. You can overwrite this attribute with a different name, which will then be used during saving with the built-in file naming rules.
- **netlist** – The **Netlist** of this board, or **None**
- **original_path** – The path to the directory or zip file that this board was loaded from.
- **was_zipped** – True if this board was loaded from a zip file.
- **generator** – A string containing an educated guess on which EDA tool generated this file. Example: **"altium"**

board_bounds(*unit=<LengthUnit millimeter>, default=None*)

Calculate and return the bounding box of this board's outline. If this board has no outline, this function falls back to **bounding_box()**, returning the bounding box of all objects on all layers and drill files instead.

Parameters:
- **unit** – **LengthUnit** or str (`'mm'` or `'inch'`). Which unit to return results in. Default: mm
- **default** – Default value to return if there are no objects on any layer.

Returns: `((x_min, y_min), (x_max, y_max))` tuple of floats.

Return type: tuple

bounding_box(*unit=<LengthUnit millimeter>, default=None*)

Calculate and return the bounding box of this layer stack. This bounding box will include all graphical objects on all layers and drill files. Consider using **board_bounds()** instead if you are interested in the actual board's bounding box, which usually will be smaller since there could be graphical objects sticking out of the board's outline, especially on drawing or silkscreen layers.

Parameters:
- **unit** – **LengthUnit** or str (`'mm'` or `'inch'`). Which unit to return results in. Default: mm
- **default** – Default value to return if there are no objects on any layer.

Returns: `((x_min, y_min), (x_max, y_max))` tuple of floats.
Return type: tuple

classmethod **from_files**(*files*, *board_name=None*, *lazy=False*, *original_path=None*, *was_zipped=False*, *overrides=None*, *autoguess=True*)

Load a board from a directory. Refer to **`open()`** for the meaning of the options.

Parameters:
- **files** – List of paths of the files to load.
- **original_path** – Override the **`original_path`** of the resulting **`LayerStack`** with the given value.
- **was_zipped** – Override the **`was_zipped`** attribute of the resulting **`LayerStack`** with the given value.

Return type: **`LayerStack`**

merge(*other*, *mode='above'*)

Merge `other` into `self`, i.e. for all layers, add all objects that are in `other` to `self`. This resets **`import_settings`** and **`generator`** on all layers. Units and other file-specific settings are handled automatically. For the meaning of the `mode` parameter, see **`GerberFile.merge()`**.

Layers are matched by their logical side and function as they are found in **`LayerStack.graphic_layers()`**. Drill layers are normalized before merging, which splits them into exactly three drill layers: An non-plated one, a plated one, and a (hopefully empty) unknown plating one.

merge_drill_layers()

Merge all drill layers of this board into a single drill layer containing all objects. You can access this drill layer under the **`LayerStack.drill_mixed`** attribute. The original layers are removed from the board.

normalize_drill_layers()

Take everything from all drill layers of this board, and sort it into three new drill layers: One with all non-plated objects, one with all plated objects, and one for all leftover objects with unknown plating. This method replaces the board's drill layers with these three sorted ones.

offset(*x=0*, *y=0*, *unit=<LengthUnit millimeter>*)

Move all objects on all layers and drill files by the given amount in X and Y direction.

Parameters:
- **x** – **`float`** with length to move objects along X axis.
- **y** – **`float`** with length to move objects along Y axis.
- **unit** – **`LengthUnit`** or str (`'mm'` or `'inch'`). Which unit `x` and `y` are specified in. Default: mm

classmethod **open**(*path*, *board_name=None*, *lazy=False*, *overrides=None*, *autoguess=True*)

Load a board from the given path.

- The path can be a single file, in which case a **`LayerStack`** containing only that file on a custom layer is returned.
- The path can point to a directory, in which case the content's of that directory are analyzed for their file type and function.
- The path can point to a zip file, in which case that zip file's contents are analyzed for their file type and function.
- Finally, the path can be the string **`"-"`**, in which case this function will attempt to read a zip file from standard input.

Parameters:
- **path** – Path to a gerber file, directory or zip file, or the string **`"-"`**
- **board_name** – Override board name for the returned **`LayerStack`** instance instead of guessing the board name from the found file names.

- **lazy** – Do not parse files right away, instead return a **LayerStack** containing :py:class:~.cam.LazyCamFile` instances.
- **overrides** – **dict** containing a filename regex to layer type mapping that will override gerbonara's built-in automatic rules. Each key must be a **str** containing a regex, and each value must be a **(side, use) tuple** of **str**.
- **autoguess** – **bool** to enable or disable gerbonara's built-in automatic filename-based layer function guessing. When **False**, layer functions are deduced only from **overrides**.

Return type: **LayerStack**

classmethod **open_dir**(*directory*, *board_name=None*, *lazy=False*, *overrides=None*, *autoguess=True*)

Load a board from a directory. Refer to **open()** for the meaning of the options.

Parameters: **directory** – Path of the directory to process.
Return type: **LayerStack**

classmethod **open_zip**(*file*, *original_path=None*, *board_name=None*, *lazy=False*, *overrides=None*, *autoguess=True*)

Load a board from a ZIP file. Refer to **open()** for the meaning of the other options.

Parameters:
- **file** – file-like object
- **original_path** – Override the **original_path** of the resulting **LayerStack** with the given value.

Return type: **LayerStack**

outline_polygons(*tol=0.01*, *unit=<LengthUnit millimeter>*)

Iterator yielding this boards outline as a list of ordered **Arc** and **Line** objects. This method first sorts all lines and arcs on the outline layer into connected components, then orders them such that one object's end point is the next object's start point, flipping them where necessary. It yields one list of (likely mixed) **Arc** and **Line** objects per connected component.

This method exists because the only convention in Gerber or Excellon outline files is that the outline segments are *visually contiguous*, but that does not necessarily mean that they will be in any particular order inside the G-code.

Parameters:
- **tol** – **float** setting the tolerance below which two points are considered equal
- **unit** – **LengthUnit** or str (`'mm'` or `'inch'`). SVG document unit. Default: mm

outline_svg_d(*tol=0.01*, *unit=<LengthUnit millimeter>*)

Return this board's outline as SVG path data.

Parameters:
- **tol** – **float** setting the tolerance below which two points are considered equal
- **unit** – **LengthUnit** or str (`'mm'` or `'inch'`). SVG document unit. Default: mm

rotate(*angle*, *cx=0*, *cy=0*, *unit=<LengthUnit millimeter>*)

Rotate all objects on all layers and drill files by the given angle around the given center of rotation (default: coordinate origin (0, 0)).

Parameters:
- **angle** – Rotation angle in radians.
- **cx** – **float** with X coordinate of center of rotation. Default: **0**.
- **cy** – **float** with Y coordinate of center of rotation. Default: **0**.
- **unit** – **LengthUnit** or str (`'mm'` or `'inch'`). Which unit `cx` and `cy` are specified in. Default: mm

save_to_directory(*path*, *naming_scheme={}*, *overwrite_existing=True*, *board_name=None*, *gerber_settings=None*, *excellon_settings=None*)

Save this board into a directory at the given path. If the given path does not exist, a new directory is created in its place.

Parameters:
- **path** – Output directory
- **naming_scheme** – **dict** specifying the naming scheme to use for the individual layer files. When not specified, the original filenames are kept where available, and a default naming scheme is used. You can provide your own **dict** here, mapping **"side use"** strings to filenames, or use one of **kicad** or **kicad**.
- **board_name** – Board name to use when naming the Gerber/Excellon files
- **overwrite_existing** – Bool specifying whether override an existing directory. If **False** and **path** exists, a **ValueError** is raised. Note that a **ValueError** will still be raised if the target exists and is not a directory.
- **gerber_settings** – **FileSettings** to use for Gerber file export. When not given, the input file's original settings are re-used if available. If those can't be found anymore, sane defaults are used. We recommend you set this to the result of **defaults()**.

save_to_zipfile(*path*, *prefix=''*, *overwrite_existing=True*, *board_name=None*, *naming_scheme={}*, *gerber_settings=None*, *excellon_settings=None*)

Save this board into a zip file at the given path. For other options, see **save_to_directory()**.

Parameters:
- **path** – Path of output zip file
- **overwrite_existing** – Bool specifying whether override an existing zip file. If **False** and **path** exists, a **ValueError** is raised.
- **board_name** – Board name to use when naming the Gerber/Excellon files
- **prefix** – Store output files under the given prefix inside the zip file

scale(*factor*, *unit=<LengthUnit millimeter>*)

Scale all objects on all layers and drill files by the given scaling factor. Only uniform scaling with one common factor for both X and Y is supported since non-uniform scaling would not work with either arcs or apertures in Gerber or Excellon files.

Parameters:
- **factor** – Scale factor. **1.0** for no scaling, **2.0** for doubling in both directions.
- **unit** – **LengthUnit** or str (`'mm'` or `'inch'`) for compatibility with other transform methods. Default: mm

to_pretty_svg(*side='top'*, *margin=0*, *arg_unit=<LengthUnit millimeter>*, *svg_unit=<LengthUnit millimeter>*, *force_bounds=None*, *tag=<class 'gerbonara.utils.Tag'>*, *inkscape=False*, *colors=None*, *use=True*)

Convert this layer stack to a pretty SVG string that is suitable for display or for editing in tools such as Inkscape. If you want to process the resulting SVG in other tools, consider using **to_svg()** instead, which produces output without color styling or blending based on SVG filter effects.

Parameters:
- **side** – One of the strings **"top"** or **"bottom"** specifying which side of the board to render.
- **margin** – Export SVG file with given margin around the board's bounding box.
- **arg_unit** – **LengthUnit** or str (`'mm'` or `'inch'`). Which unit `margin` and `force_bounds` are specified in. Default: mm
- **svg_unit** – **LengthUnit** or str (`'mm'` or `'inch'`). Which unit to use inside the SVG file. Default: mm
- **force_bounds** – Use bounds given as **((min_x, min_y), (max_x, max_y))** tuple for the output SVG file instead of deriving them from this board's bounding box and `margin`. Note that this will not scale or move the board, but instead will only crop the viewport.
- **tag** – Extension point to support alternative XML serializers in addition to the built-in one.
- **inkscape** – **bool** enabling Inkscape-specific markup such as Inkscape-native layers

- **colors** – Colorscheme to use, or **None** for the built-in pseudo-realistic green solder mask default color scheme. When given, must be a dict mapping semantic **"side use"** layer names such as **"top copper"** to a HTML-like hex color code such as **#ff00ea**. Transparency is supported through 8-digit color codes. When 8 digits are given, the last two digits are used as the layer's alpha channel. Valid side values in the layer name strings are **"top"**, **"bottom"**, and **"mechanical"** as well as **"inner1"**, **"inner2"** etc. for internal layers. Valid use values are **"mask"**, **"silk"**, **"paste"**, and **"copper"**. For internal layers, only **"copper"** is valid.
- **use** – Enable/disable `<use>` tags for aperture flashes. Defaults to **True** (enabled).

Return type: **str**

to_svg(*margin=0*, *side_re='.*'*, *drills=True*, *arg_unit=<LengthUnit millimeter>*, *svg_unit=<LengthUnit millimeter>*, *force_bounds=None*, *colors=None*, *tag=<class 'gerbonara.utils.Tag'>*)

Convert this layer stack to a plain SVG string. This is intended for use cases where the resulting SVG will be processed by other tools, and thus styling with colors or extra markup like Inkscape layer information are unwanted. If you want to instead generate a nice-looking preview image for display or graphical editing in tools such as Inkscape, use **to_pretty_svg()** instead.

WARNING: The SVG files generated by this function preserve the Gerber coordinates 1:1, so the file will be mirrored vertically.

Parameters:
- **margin** – Export SVG file with given margin around the board's bounding box.
- **side_re** – A regex, such as `'top'`, `'bottom'`, or `'.*'` (default). Selects which layers to export. The default includes inner layers.
- **drills** – **bool** setting if drills are included (default) or not.
- **arg_unit** – **LengthUnit** or str (`'mm'` or `'inch'`). Which unit `margin` and `force_bounds` are specified in. Default: mm
- **svg_unit** – **LengthUnit** or str (`'mm'` or `'inch'`). Which unit to use inside the SVG file. Default: mm
- **force_bounds** – Use bounds given as **((min_x, min_y), (max_x, max_y))** tuple for the output SVG file instead of deriving them from this board's bounding box and `margin`. Note that this will not scale or move the board, but instead will only crop the viewport.
- **colors** – Dict mapping `f'{side} {use}'` strings to SVG colors.
- **tag** – Extension point to support alternative XML serializers in addition to the built-in one.

Return type: **str**

property **bottom_side**

Return a dict containing the subset of layers from **graphic_layers()** that are on the board's bottom side. Includes the board outline layer, if available.

property **copper_layers**

Return all copper layers of this board as a list of ((side, use), layer) tuples. Returns an empty list if the board does not have any copper layers.

property **drill_layers**

Generator iterating all of this board's drill layers.

property **inner_layers**

Return all inner copper layers of this board as a list of ((side, use), layer) tuples. Returns an empty list if the board does not have any inner layers.

property **outline**

Return this board's outline layer if available, or **`None`**.

property **top_side**

Return a dict containing the subset of layers from **`graphic_layers()`** that are on the board's top side. Includes the board outline layer, if available.