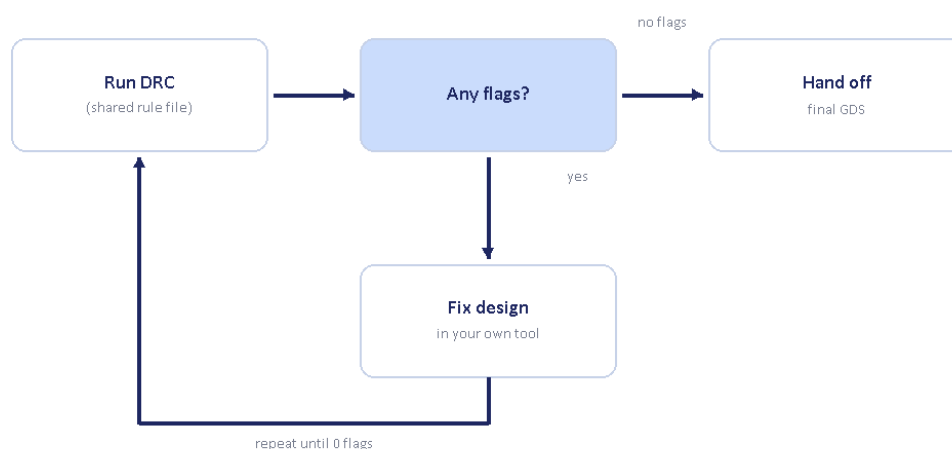




How to Use This Shared DRC Rule File

A step-by-step guide for groups submitting a design to this multi-project wafer (MPW) run

This folder contains a DRC (design rule check) rule file — a plain-text `.yaml` file — that defines the fixed layer stack and design rules for this MPW run. Every participating group runs their own design against the *same* rule file before handing it back, using a small checker script built on KLayout's own Python engine (the `klayout` package — the same geometry engine behind KLayout's DRC feature, so this genuinely is “running it in KLayout,” just via its scriptable engine rather than its interactive menus).



The loop every submission goes through: run, check, fix, repeat until clean, then hand off.

Before you start

You need three things, all provided in this same folder:

- **The rule file** — `mpw_rules.yaml`. Fixed by the MPW organizer — **don't edit the rule values**.
- **The checker toolkit** — `engine.py`, `run_drc.py`, and `view_report.py`, right alongside this guide. This is what actually runs the checks; the `.yaml` file alone doesn't do anything without it.
- **Python 3.9+** with two packages installed.

Two `.yaml` files ship in this folder — here's the difference

`mpw_rules.yaml` — the one you actually use. This is the fixed, active rule table for this MPW run: every layer number and threshold in it is already set by the organizer. The only thing you touch is its name: field (Step 1), then you run the checker against it (Step 2).

`_template.yaml` — not used for this submission at all. It's the same rule schema with every section left blank, kept in the folder purely as a starting point for the rare case where you (or the organizer) need to build a brand-new rule table later, for a different process or design family this run doesn't cover. You can safely ignore it unless you're doing that — see “Behind the scenes” below for the full build-from-scratch process.

Install the two required Python packages

```
pip install klayout pyyaml
```

That's it — `klayout` is KLayout's official Python package; no separate application license or install is required just to *run* the checks.

Optional: install the KLayout desktop application

Not required to run the checks, but strongly recommended for actually *looking at* what failed — the checker produces a report you can load directly onto your layout. Free download: www.klayout.de.

Step-by-step

1 Fill in your design name

Open `mpw_rules.yaml` in any text editor and replace the placeholder on the first real line of content:

```
name: CHANGE_ME          →   name: your_design_name
```

This is for your own records only — the checker doesn't use it for anything.

2 Run the checker against your GDS

From inside this folder:

```
python run_drc.py --gds path\to\your_design.gds \  
                  --rules mpw_rules.yaml \  
                  --out  reports\your_design
```

Or just run `python run_drc.py` with no arguments and it will prompt you for each path instead.

3 Read the result

The console prints one line per violation category, then a summary:

```
DRC: your_design.gds against mpw_rules.yaml  
[width] GP_RESONATOR: 3 location(s) narrower than 2.0 um  
-> 3 geometric violation(s), 0 hierarchy note(s)  
reports written to reports\your_design.lyrdb / reports\your_design.json
```

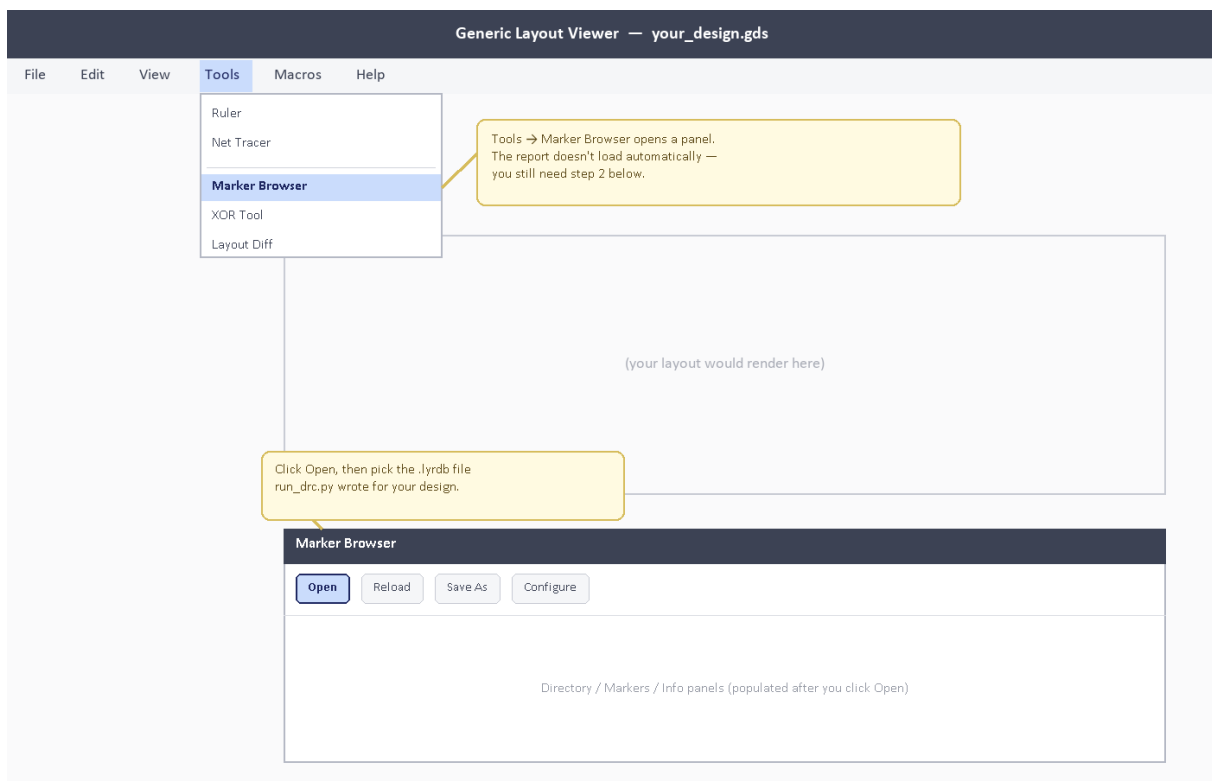
Zero violations and zero hierarchy notes → the checker exits with code 0 (clean). Anything else → exit code 1, and every flagged category is listed above.

Note: the two output files serve different purposes. The `.json` file is a flat, readable summary. The `.lyrdb` file is a KLayout “marker database” — load it alongside your GDS to see exactly where each violation is, which is far faster than reading coordinates.

4 (Recommended) Look at the violations visually

Open your GDS in the KLayout desktop application. Loading the `.lyrdb` report is a two-step process, confirmed against KLayout’s own manual (klayout.de/doc/manual/marker_browser.html):

- 1 **Tools** → **Marker Browser** — opens the Marker Browser panel (it starts empty; this step alone doesn’t load anything).
- 2 Inside that panel, click **Open** and pick the `.lyrdb` file `run_drc.py` wrote for your design.



*Illustrative diagram of a generic layout-viewer menu — **this is not an actual KLayout screenshot**, it's a schematic of the two-step path above. Exact panel layout differs by KLayout version.*

Once loaded, each violation highlights directly on your layout, with the rule category shown in the Marker Browser's Directory/Markers/Info panels you can click through one at a time.

5 Fix, re-run, repeat

Correct the flagged geometry in your own design tool, re-export the GDS, and go back to Step 2. Repeat until Step 3 reports zero violations and zero hierarchy notes.

6 Hand off

Once the checker reports a clean pass, send the final GDS back to the MPW organizer. Keep the last .json report — it's useful evidence that the submitted design was actually checked clean, not just visually inspected.

Behind the scenes: where these two files come from

You don't need this section to complete Steps 1–6 above — it's here for anyone who wants to understand (or later build) the files themselves, rather than just use the ones provided.

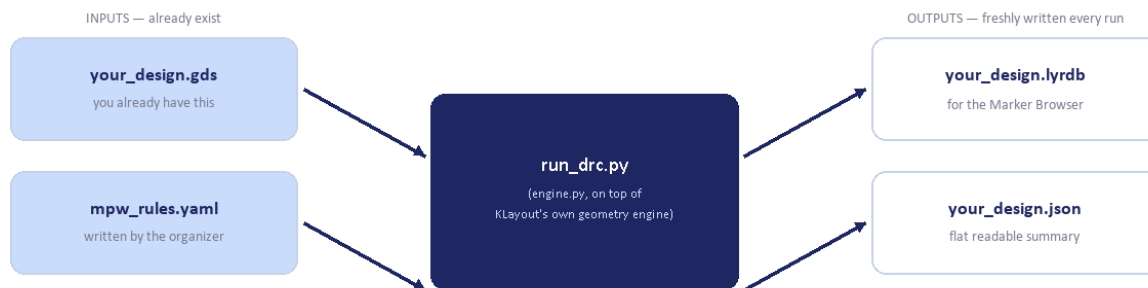
The .yaml rule file — for this run, you fill one in, you don't build one

For this MPW run, `mpw_rules.yaml` already exists — the organizer fixed every layer number and threshold in it. Your only edit is the `name:` field in Step 1. It's a plain text file with no special tooling: open it in any editor, following the schema already there (`layers`, `width`, `overlap_forbidden`, and so on — see the reference table on the next page for what each one means).

If you ever need to build a *new* rule file from scratch — for your own project, or a design family this guide doesn't cover — the process is:

- 1 Copy `_template.yaml` (provided in this folder) — the same schema with every section left blank.
- 2 Fill in `layers:` with the real GDS layer/datatype numbers for each name you introduce.
- 3 Fill in whichever rule categories apply, using the values from your own process spec — not guessed numbers.
- 4 Validate the syntax (see Troubleshooting on the last page) before running it for real.
- 5 Run it against a design you already believe is clean, and treat every hit as a question, not a confirmed bug — a theoretically-correct value doesn't always give zero violations on real geometry (e.g. a label engraved into the same metal layer as a trace will legitimately trip a blanket width rule). Refine the rule table, don't just delete the finding.

The .lyrdb file — you never write this by hand, it's generated output



Both output files are written fresh by `run_drc.py` every time you run it — re-running after a fix overwrites the previous report at that path, it doesn't accumulate.

Concretely: `run_drc.py` reads your GDS and the rule file, runs every check in `engine.py` using KLayout's own geometry engine, and turns every violation into an entry in a KLayout “Report Database” object — which is what gets saved as the `.lyrdb` file. It's an XML format under the hood, but you never need to open it as text; the Marker Browser (Step 4) is the intended way to read it. The `.json` file is the same findings written out as plain, script-friendly data instead.

What's actually being checked

Every category that can appear in the console output, in plain English. The exact layers and thresholds are fixed in `mpw_rules.yaml` for this run.

Category	What it means
width	A feature on this layer is narrower than the minimum allowed.
space	Two shapes on the same layer are closer together than the minimum gap.
enclosure	One layer doesn't sit far enough inside another where it's meant to nest (e.g. a via inside a pad).
overlap	Two layers that must never touch at all are overlapping.
min_overlap	Two layers that ARE meant to overlap don't overlap by enough — usually a sign of misregistration, not a clean overlap.
min_area	An individual shape is smaller than the minimum allowed area.
angle	A shape on a Manhattan-only layer has a non-90° edge.
no_touching	Two originally-separate shapes on the same layer are touching/overlapping when they shouldn't be — a likely short.
orientation	A shape's asymmetry (e.g. a tapered lead) points the opposite way from the majority of its peers — a likely mirrored instance.
duplicate	The exact same shape appears twice at the same location — usually a copy-paste or array mistake.
hierarchy notes	Structural issues in the GDS itself: unreferenced cells, empty cells, or a cell that references itself.

Troubleshooting

“ModuleNotFoundError: No module named 'klayout'” (or 'yaml')

The two packages from the Prerequisites section aren't installed in the Python environment you're running from. Re-run `pip install klayout pyyaml` in that same environment.

The rule file loads but every layer shows 0 shapes

Your GDS almost certainly uses different layer/datatype numbers than the ones fixed in `mpw_rules.yaml`. Layer numbers are fixed by the organizer for this MPW run — double-check your export settings match them, rather than editing the rule file's layer numbers to match yours.

A YAML edit broke the file

YAML is indentation-sensitive. Validate before re-running:

```
python -c "import yaml; yaml.safe_load(open('mpw_rules.yaml'))"
```

If that errors, it names the line/column to check.
