

Concept of the **packpay** project

v0.72.77.212

Péter Felecan

assisted by Claude Sonnet 5.5 (Anthropic)

October 12, 2026

Contents

1	Introduction	2
2	Origin	3
3	Motivation	3
4	Goal	3
5	Name of the project	4
6	Taxonomy of breadboards	4
6.1	What the taxonomy covers	4
6.2	The axes	5
6.3	The classes	5
6.4	Where this project stands against the classes	6
6.4.1	F1. The classic board is 63 columns wide	6
6.4.2	F2. More than five rows per half is a board that does not exist . .	6
6.4.3	F3. Rails between zones: three buses on a double board	6
6.4.4	F4. Rails on the common boards are split in the middle	7
6.4.5	F5. How boards join	7
6.5	Sources of the taxonomy	7
7	Best practices	8
7.1	On the breadboard	8
7.1.1	Connect both power rails first, plus to plus and minus to minus . .	8
7.1.2	Bridge the split power rails, or check them with a multimeter . . .	9
7.1.3	Wire only power nets to the rails; wire signals with jumpers	9
7.1.4	Straddle ICs across the trench, and never put two leads in one hole	9

7.1.5	Keep jumper wires short and flat, and do not run them over ICs . . .	9
7.1.6	Put a decoupling capacitor at the power pin of every IC	10
7.1.7	Say which way round the LEDs, diodes and electrolytic capacitors go	10
7.1.8	Put an IC in with its pin 1 where the mark says, and number the pins counter-clockwise	10
7.1.9	Never let two pins of one part share a column of holes	11
7.1.10	Colour-code the wires: red for power, black for ground, others for signals	12
7.1.11	Build and test in stages, and check with a multimeter	12
7.1.12	Know what a breadboard cannot do	12
7.1.13	When a circuit outgrows one board, link more boards	13
7.1.14	Name every hole unambiguously	13
7.2	On the scheme	13
7.2.1	Let the flow run left to right, with power at the top and ground at the bottom	13
7.2.2	Divide the design into functional blocks, one sheet for each	13
7.2.3	Join sheets explicitly, and use global labels sparingly	14
7.2.4	Use net labels for long connections, and name nets descriptively . .	14
7.2.5	Give every part a unique reference with the standard prefix	14
7.2.6	Put decoupling capacitors beside their IC, or all together, but be consistent	14
7.2.7	Mark every power net as driven	15
7.2.8	Run the electrical rules check, and fix every error	15
7.2.9	Name power nets for their voltage rather than "VCC" alone	15
7.3	Sources	15
8	Glossary	16
8.1	Naming decisions	17
8.2	Breadboard terms	17
8.3	Scheme terms	20
8.4	Project terms	21
8.5	Sources	21

1 Introduction

The `packpay` project's goal is to provide a netlist-driven schematic and breadboard-diagram generation and auto-layout engine: given a real KiCad netlist, algorithmically produce a placed, wired KiCad schematic and/or a connectivity-focused breadboard-style diagram (and, from that diagram, a step-by-step build guide), with no hand-placed coordinates anywhere in the loop.

This document defines the project’s goal, scope, and boundaries. It is a living document and will be updated as the project evolves. Decisions that shape the content of this document are recorded with rationale in [ResearchJournal.pdf](#).

2 Origin

packpay did not start as its own project. It was extracted, 2026-08-14, from dudhya (a separate project researching the revival of older smartphones), where this same engine was originally built and refined across many sessions: first for a breadboard-style diagram of one real circuit (a Google Pixel 3a battery-free power stage), then generalized and extended to KiCad schematic auto-layout for that same circuit, then tested against synthetic circuits built specifically to stress topology that one real circuit never exercised.

By the time of extraction, the engine had proven itself against three genuinely different circuits with real, automated verification (electrical rules checking, netlist equivalence, geometric bounds/overlap checks) at every step – a reasonable bar for “this generalizes,” not just a hope. Keeping it embedded inside a project about old-phone hardware revival was becoming architecturally awkward: the engine itself has nothing to do with smartphones, and a second, unrelated circuit wanting to use it would have had to depend on the whole dudhya project to get it. See ResearchJournal.org’s own extraction entry for the full account, including what stayed in dudhya (the real, physically- designed Pixel 3a circuit itself, and dudhya’s own hand-placed reference generators) and what moved here (everything genuinely circuit-agnostic).

3 Motivation

Hand-placing a schematic’s own component coordinates, or hand- drawing a breadboard-style companion diagram, is real, repetitive, error-prone work that a real netlist (which component connects to which, on which pin) already contains enough information to automate – if the automation is built to genuinely read that netlist as its own source of truth, not to reproduce a hand-placed layout it was shown once.

A netlist-driven approach also has a real, concrete benefit beyond saving hand-placement time: a schematic and a breadboard diagram generated from the SAME real netlist can never silently drift out of sync with each other, or with the actual, real electrical design, the way two independently hand-maintained representations eventually do.

4 Goal

Given a real netlist (loaded from an existing KiCad schematic’s own export, or built directly for a circuit that doesn’t have one yet), produce:

- A placed, wired, KiCad-valid schematic (`kicad_layout.py`), with every component's own position and rotation decided algorithmically, verified against real sheet bounds, the title block, and component-to-component overlap before being handed back.
- A connectivity-focused, physically-realistic breadboard-style diagram (`auto_layout.py`, `render.py`), with real footprint- courtyard-derived component sizing where a real footprint is mapped, verified for connectivity, real board-bounds, and hole- collision before being handed back. A single, physical breadboard's own real geometry is described by a config file (`breadboard_config.py`; see `breadboard_config(5)`), and an assembly of several, genuinely different physical boards is supported directly, with components and their own connections placed across the whole assembly, not just one board in isolation.
- A step-by-step, prose build guide (`build_instructions.py`), derived from that same placed breadboard diagram, so a person assembling the real, physical circuit by hand has an ordered set of instructions to follow rather than needing to read the diagram itself.

All three consume the exact same real, shared **Netlist** representation (`netlist.py`), confirmed directly (not assumed) to be the case for every real consumer built so far.

Workflow.org walks this same pipeline concretely, stage by stage, with real commands and their real, verified output; Troubleshooting.org covers what to do when a real circuit doesn't lay out cleanly; LibraryReference.org is the practical reference for each module's own public API; KicadGen.org documents `kicad_gen.py`'s own small language, the layer `kicad_layout.py`'s own generator is built on.

5 Name of the project

The name is a romanized abbreviation of the Sanskrit phrase: प्रवाहित अग्निचित्स्व कृते परिवर्तनयन्त्रम् (`pravahit agnichitrasya krute parivartanayantram`), "transformation engine for flowing fire diagram" – "flowing fire diagram" for a schematic (current, conventionally drawn flowing through a circuit; a "diagram" of switching, connection, transformation), "transformation engine" for what this project actually does: takes a netlist and transforms it into a placed, drawn representation.

6 Taxonomy of breadboards

6.1 What the taxonomy covers

Solderless, plug-in breadboards: a plastic body with spring contacts in a 2.54 mm (0.1 inch) grid. Out of its scope: perfboard and stripboard, whose holes are joined differently or not at all (not researched here). A *solderable* breadboard (a bare circuit board that copies a solderless one) is related but separate: the manufacturer of one says it has the

same pattern and spacing as the solderless board it copies, so that a finished project can be moved onto it (BusBoard's SB1660 for its BB1660, T5).

6.2 The axes

Five properties tell the boards apart; everything sold is a point in this space.

strips per zone How long a zone is: the number of numbered strips (the project's columns).

Real products have 17 (mini), 30 (half-size) and 63 (full-size and double). The reference works give a range, "about 56 to 65 rows of connectors" for a full-size board (S2), which in this project's words is columns.

zones per board One terminal block with its trench, or several stacked: two on a 1660-point board, which the manufacturer describes as "essentially two 830-point breadboards side-by-side" (T7).

rows per half Five on every product found: the listings' "10 columns" are 5 above the trench and 5 below (the project's rows, T1 to T4).

power rails None on a mini board; on the others two strips of long rails, each strip in two halves (the 830's "2 split power buses", T2; its 200 rail points are four rails of 50, derived).

assembly Boards are made to be joined: the sellers say they "snap" or interlock "either way, to make longer and/or wider breadboards" (T1, T4). Some have rails that pull off to make the board thinner (T1).

6.3 The classes

Tie points are the sellers' totals; the arithmetic of the table is checked by the test suite: zones times strips times ten (five holes in each half of a strip), plus the rail points, is the total.

class	tie points	zones	strips per zone	rail points	size (inches)
mini	170	1	17	0	derived: 17 strips
half-size	400	1	30	100	3.3 x 2.2
full-size	830	1	63	200	6.5 x 2.1
double	1660	2	63	400	6.5 x 4.3

Reading of the table. The mini board's 170 points are 17 strips of ten with no rails (derived: SparkFun and Adafruit give only the total, T3). The half-size's rail points are 400 minus 300 (derived). The double board is two terminal strips of 630 points and eight rails of 50 (T5), 6.5 inches long as a full-size board and twice as wide (T5, T6); the double board examined has three distribution buses (F3). Larger boards exist: SparkFun lists a giant one with 7 power buses, 40 columns and 63 rows, whose total was cut off in the

listing read (T3); BusBoard lists a 1460-point board of 1260 circuit points and four rails of 50 (T8). They are not in the table because their structure was not read in full.

6.4 Where this project stands against the classes

The mapping: a zone is a `SectorConfig`, a board a `BoardConfig`, an assembly of boards a `MultiboardConfig`. The optimizer's ladder is this taxonomy read upwards (`classic_board`, `escalation_board_tiers`): the classic board is a full-size one, two zones is a double board, further boards is a snapped-together assembly, and more rows per half (`ROWS_PER_HALF_LADDER`) is something no product found offers. Findings of the comparison, with what the project does about each.

6.4.1 F1. The classic board is 63 columns wide

What `CLASSIC_COLUMNS` is 63. Every full-size and double board found has 63 strips (S2 gives 56 to 65 as a range). The two boards the project's configurations describe (breadboard1.conf and breadboard2.conf) have 65 strips, inside that range, which is why those configurations say 65. A layout made for 63 fits a board of 65; the reverse is not true.

Measured handed a 63-column board instead of a 65-column one, the optimizer returned the same tier and the same placement for all seven shipped designs, rp2040 (two zones) included. The widest of the six single-zone designs reaches column 62.

6.4.2 F2. More than five rows per half is a board that does not exist

What `ROWS_PER_HALF_LADDER` is 5, 7, 10, 15, 20. Every product found has five rows per half, so the ladder's last resort describes a virtual board, like an unbounded-width one.

Consequence the ladder is kept as a last-resort fallback, tried only after every arrangement of zones and boards at five rows. Stating in the instructions' header that such a board is virtual would be an option.

6.4.3 F3. Rails between zones: three buses on a double board

What The model has one rail shared between each pair of adjacent zones, $N+1$ rails for N zones. BusBoard describes the 1660-point board as four 100-point distribution strips for two terminal strips (T5); a distributor's table lists three distribution buses (T6); the double board examined has three, which is the model's count. The manufacturer's four strips agree if its middle bus counts as two adjacent strips (not verified).

6.4.4 F4. Rails on the common boards are split in the middle

What Both kinds exist. Sellers' manuals say most current 830-point boards split each rail in the middle, 50 points and 50, and that some versions are continuous (T9); a seller's page shows a gap at strip 30 of 63, and its customers complain that all past full boards had full-length rails (T10); a builder found the buses of a board made of four clipped boards interrupted in the middle (T11); a reference says "on many full-size boards" (T12). The boards the project's configurations describe are continuous: the splitting is a feature of some current products, not of boards in general.

The model RailConfig has the flag `continuous`, with a note that where a break lies would need a parameter of its own. The build instructions read the flag: a board declared not continuous says so in its header and asks for a multimeter check and a jumper (the best practice "bridge the split power rails" is guarded by `check_split_rail_note`). Not modelled: where the break lies.

6.4.5 F5. How boards join

What Sellers say boards interlock lengthwise and widthwise (T1, T4, T10). Clipping is mechanical only: a builder found that four clipped boards had independent rails (T11).

The model each board of a `MultiboardConfig` is independent, with its own rails and zones, and the instructions say "this wire needs to go to board N": right for boards that are only clipped together. Not modelled: where the boards sit, which would matter only for the length of the wires between them.

6.5 Sources of the taxonomy

Checked on 2026-10-08, and paraphrased here. S1 to S14 are the sources of the best practices; the taxonomy has its own.

T1 [Adafruit, breadboards](#) (400 and 830 points, sizes, boards that snap together, rails that pull off).

T2 [A full-size breadboard listing](#) (830 points: 2 split power buses, 10 columns, 63 rows; interlocking; 6.5 x 2.1 inches).

T3 [SparkFun, breadboards](#) (mini 170, full-size, a giant board).

T4 [Keyestudio KS0331](#) (400 points: 30 rows, 10 columns, four bus lines, interlocking).

T5 [BusBoard, BB1660](#) (the manufacturer: 1660 points, two 630-point areas, eight rails of 50, 36 V and 2 A) and its [SB1660 page](#).

T6 [DigiKey, BB1660](#) (a distributor: 2 terminal strips, 3 distribution buses, 6.50 x 4.30 inches).

- T7** [Jameco, BB1660](#) (essentially two 830-point breadboards side-by-side).
- T8** [Eleshop, BusBoard BB1460](#) (1460 points).
- T9** [ShillehTek, 830-point breadboard manual](#) (most 830 boards split each rail in the middle, some versions are continuous: check with a multimeter).
- T10** [SparkFun, full-size breadboard](#) (interlocking; its customers' comments on the split rails).
- T11** [Maker Pro forum, a board of four clipped boards](#) (buses interrupted in the middle).
- T12** [Tinkered, full-size breadboard](#) (rails "usually split mid-length").

7 Best practices

A practice says what "good" means and why, which is what justifies the rules the engine follows and the checks the test suite makes. The [workflow document](#) says how to apply the practices, step by step. The terms are those of the [Glossary](#).

Every practice has the same three lines. *Why* gives the reason, with the numbers where there are any. *Sources* names where the practice comes from (listed at the end; they do not all agree, and where they differ the entry says so). *In packpay* says what this project does about it, in one of four words, and the test suite checks the words: *guarded by* names the tests that fail if the practice is broken, and each must exist; *encoded* means the code does it but no test is aimed at it; *open* means the project does not do it yet and names the research item that holds the gap; *outside the tool* means it is the builder's or the designer's business. A practice the tool neglects is listed as open, not left out.

7.1 On the breadboard

7.1.1 Connect both power rails first, plus to plus and minus to minus

Why Poor power distribution is blamed for more breadboard failures than anything else, and connecting a supply to the wrong rail is an immediate short. The red and blue markings are only a reference, so keep them in order.

Sources S1, S3

In packpay guarded by `check_rail_direct_power_source_kicad_side` and `check_rail_direct_power_source` a source wired to a rail (`rail_direct_power_source`) is named in the instructions, with its polarity read from its pins.

7.1.2 Bridge the split power rails, or check them with a multimeter

Why Many full-size boards split their rails at the middle, so a rail that looks continuous is two. Bridging the break on both the plus and the ground rail avoids a supply that quietly stops halfway.

Sources S1, S3, S7, S15, S16

In packpay guarded by `check_split_rail_note`: a board whose configuration says its rails are not continuous (`continuous`) says so in the instructions' header and asks for a multimeter check and a jumper; a board that is not declared split, the default and the maker's own boards, says nothing. The tool cannot know a board is split unless the configuration tells it, and where the break lies is not modelled.

7.1.3 Wire only power nets to the rails; wire signals with jumpers

Why The rails are there to give access to power wherever it is needed. A signal wired to a rail is shorted to the supply. This project learned it the hard way: it once treated every net of more than four members as a rail, and told the builder to put two charlieplex LED lines on the plus rail.

Sources S1

In packpay guarded by `check_breadboard_rails_are_power`: a net is wired to a rail only if it is a power net (`breadboard_rail_nets, is_rail_like_name`).

7.1.4 Straddle ICs across the trench, and never put two leads in one hole

Why The trench isolates the two sides so each leg of an IC is its own connection, and a five-hole strip takes up to five leads. A lead standing on another part's wire, or two leads in one hole, is a short or a bad contact.

Sources S1, S2

In packpay guarded by `check_rendered_layouts_have_no_hole_collisions` and `check_link_rows_stay_in_hole`: anchors straddle the trench by construction, and every hole a link names is a real hole of its strip.

7.1.5 Keep jumper wires short and flat, and do not run them over ICs

Why A jumper adds inductance (10 to 20 nH is quoted for one) and picks up noise. Experienced builders add that at low frequency it matters little, the exception being around a crystal oscillator.

Sources S3, S4, S7

In packpay open ("Left-side bias" in FutureResearch.org): the layout puts a part in the column of the pin it shares a net with and joins distant columns with a jumper, but nothing measures or limits the length of a wire.

7.1.6 Put a decoupling capacitor at the power pin of every IC

Why About 100 nF ceramic between each IC's power pin and ground suppresses supply noise. On a breadboard it is not trivial, and its absence is met long before stray capacitance becomes a problem.

Sources S3, S4, S6

In packpay open ("Gaps the best-practices research found" in FutureResearch.org): a capacitor on the IC's power net is placed in that pin's column, but the engine has no notion of decoupling, so nothing checks that an IC has one or that it is close.

7.1.7 Say which way round the LEDs, diodes and electrolytic capacitors go

Why Reverse polarity damages these parts. A good instruction names the anode and the cathode, as a tutorial does when it sends the LED's anode to the supply rail and its cathode to the resistor.

Sources S1

In packpay guarded by `check_lead_polarity`: a step names the role of each lead of a two-lead part whose pins the netlist names as polar (A and K, + and -), with what tells the leads apart: an LED's anode is the longer lead and its cathode the shorter one on the flat side of the case, a diode's cathode is the end marked with a band, a polarised capacitor's positive lead is the one marked +, and a part with no known marking (a photodiode) gets the bare role (`polarity_word`, `lead_polarity`). KiCad names both pins of a polarised capacitor "", so its positive pin is taken from the library's convention (pin 1, confirmed against the symbol's own drawing: `~polar_pins`). A resistor says nothing. Not described: which way round an IC or a connector sits (its pin 1).

7.1.8 Put an IC in with its pin 1 where the mark says, and number the pins counter-clockwise

Why A through-hole IC is a dual in-line package (DIP): two rows of pins 0.1 inch apart, wide enough to straddle the trench. A notch at one end, or a dot, a triangle or a bevel beside one corner, marks where pin 1 is, and the pins are numbered counter-clockwise seen from above: pin 1 is next to the mark on one row and the highest number is opposite it on the other. An IC put in the wrong way round is powered the wrong way round and is likely to be ruined. The mark labels the corner where pin 1 is, not one pin; some parts carry no mark and are oriented by their printed

text; and connectors do not all number their pins counter-clockwise (a pin header's are numbered along its length).

Sources S17, S18, S19

In packpay open ("IC orientation and physical fidelity" in FutureResearch.org). The instructions say which anchors do not plug straight into a breadboard (`check_part_fit`), and list each pin of a part with its hole and the pins on each rail, so that a schematic can be rebuilt from them without being told which pin is which (`check_anchor_pin_identities`); a connector or a cell wired to the rails names its pins on each rail, and a lead whose pin the default numbering would get wrong names it (`check_pin_identities_non_anchors`). A dual-row DIP is laid out where its footprint puts its pins, two in each column, and the instructions say which way the notch faces and where pin 1 is (`check_physical_dip_layout`, `check_synthetic_dip_circuit`); a single-row header stands in one row of holes with pin 1 at the left (`check_physical_headers`); which half of the board a header stands in makes no difference to what a builder sees (measured on every design that has headers: it stays the upper half); `PACKPAY_PHYSICAL_ANCHORS=0` and `PACKPAY_PHYSICAL_HEADERS=0` restore the earlier diagram layout. A two-row socket or header cannot be plugged into a breadboard at all, since two of its pins would share a column: it is flagged and keeps the diagram layout, one column per pin, which is what a breakout board or jumper leads give. A DIP whose rows are further apart than 7.62 mm (a 28-pin one is 15.24 mm) can be plugged in, across the trench, but its pins stand one hole out from the inner rows, which an anchor cannot yet express (anchors have only row 0), so it still follows the diagram (`check_ic_orientation_gap` pins the diagram layout, measured on camera-trap's PC817). Everything the example designs need is done; what stays open waits for a design that has the case: the wide DIP above, and a two-row part whose rows are not one hole apart.

7.1.9 Never let two pins of one part share a column of holes

Why The five holes of a column in one half of the board (a strip) are one connection, so two pins of the same part that land in one strip are shorted together, whatever the circuit says. A part whose rows of pins are one hole apart, such as a 2x4 socket, puts a pin of each row in every column and so cannot be plugged into a half of a breadboard in either direction; only a part whose pins stand in one row (a one-row header) or whose two rows are far enough apart to straddle the trench (a DIP, rows three holes apart or more) has a clean way to go in. Such a part goes on a breakout board, or each of its pins gets its own jumper lead.

Sources S1, S2

In packpay guarded by `check_part_fit`: a part with no clean way to be plugged in is flagged in its step ("Its two rows of pins are only one hole apart, so two of its pins

would share a column of holes and be shorted ...”) and named in the line under the title; the rule tries each axis and each place the trench can fall (rp2040’s two-row socket, which the first version of the classification had called plug-in).

7.1.10 Colour-code the wires: red for power, black for ground, others for signals

Why A wire’s colour tells a reader what it carries without tracing it, and a different colour for another supply voltage (orange for 3.3 V, for instance) prevents mixing two supplies.

Sources S1, S3, S4

In packpay open (“Gaps the best-practices research found” in FutureResearch.org): the drawing colours the rails red and blue and every jumper one colour, and the instructions say nothing about the colour of a wire.

7.1.11 Build and test in stages, and check with a multimeter

Why Check the continuity of the rails, and the voltage between them, before relying on them: a rail may be split without looking it.

Sources S1, S3

In packpay outside the tool: the instructions are ordered by position on the board, not by stages of a test, and the builder decides how to test.

7.1.12 Know what a breadboard cannot do

Why It is not for high current, high voltage, RF, fast clocks or precision analogue work. The numbers are not settled: one manufacturer rates its board at 36 V and 2 A and quotes 2 to 25 pF of stray capacitance per contact; a hobbyist reference gives 0.5 to 1 A per contact and about 1 MHz as a practical ceiling; experienced builders find ordinary digital logic workable at tens of MHz, with missing decoupling and bad probing biting first.

Sources S5, S6, S7

In packpay outside the tool: the tool neither models these limits nor warns about them. A suitability warning is a research item (“Gaps the best-practices research found” in FutureResearch.org).

7.1.13 When a circuit outgrows one board, link more boards

Why Breadboards are made to be joined, and a circuit that does not fit one is better on two than crammed into one.

Sources S1

In packpay guarded by `check_escalation_policy`: a classic board first (one zone, five rows per half, 63 columns), then two zones, then further boards added one at a time, and more rows only as the last resort (`classic_board`, `escalation_board_tiers`, `walk_escalation`).

7.1.14 Name every hole unambiguously

Why Letters and numbers are marked on the board so that a wire is plugged into a named hole rather than eyeballed. With more than one zone, a name alone is ambiguous, and the zone must be said.

Sources S1

In packpay guarded by `check_zones_are_named_in_the_instructions` and `check_instructions_board_ge`: a hole of a zone after the first reads "a5 (zone 2)" (`hole_name`, `parse_hole`), and a board that is not the default is stated in the header (`board_note`).

7.2 On the scheme**7.2.1 Let the flow run left to right, with power at the top and ground at the bottom**

Why A reader should not have to reconstruct the circuit before reading it. Inputs go on the left and outputs on the right, power enters from the top and ground sits at the bottom.

Sources S8, S9, S10, S11, S14

In packpay open ("Gaps the best-practices research found" in FutureResearch.org): whether the schematic generator respects this convention has not been measured.

7.2.2 Divide the design into functional blocks, one sheet for each

Why Each page should hold complete functional blocks, not pieces of several; a top-level block diagram sheet shows how they join.

Sources S8, S9, S10, S11, S14

In packpay guarded by `check_source_sheet_grouping`, `check_sheet_partition`, `check_sheet_reuse` and `check_hierarchical_generation`: a design's own functional sheets are kept, and a sheet used more than once is generated once.

7.2.3 Join sheets explicitly, and use global labels sparingly

Why Hierarchical labels and sheet pins make a connection between sheets visible; global labels, which include power symbols, connect everywhere and weaken a hierarchical design.

Sources S8, S10, S12

In packpay guarded by `check_multisheet_hierarchy_kicad_cli` and `check_nested_hierarchy_kicad_cli`. KiCad's own netlist of the generated hierarchy equals the input's.

7.2.4 Use net labels for long connections, and name nets descriptively

Why A label is clearer than a long wire that crosses others, and a net named for what it carries documents itself.

Sources S9, S10, S11

In packpay guarded by `check_label_collision_detection` and `check_dense_circuit_fixes`: the generator connects by labels at the pins, and refuses a schematic whose geometry would join two different nets (`geometric_shorts`).

7.2.5 Give every part a unique reference with the standard prefix

Why R for resistors, C for capacitors, U for ICs, and so on, numbered in reading order, so a part can be found on the board, in the list and on the sheet.

Sources S9

In packpay encoded: references come from the source netlist and are never renumbered, and a multi-unit part keeps its real reference (`expand_multi_unit`).

7.2.6 Put decoupling capacitors beside their IC, or all together, but be consistent

Why Beside the IC, the reader sees which capacitor serves which IC; in a block of their own the schematic stays tidy. The sources differ on which is better.

Sources S9, S10, S11

In packpay open ("Gaps the best-practices research found" in FutureResearch.org): not measured, as on the breadboard.

7.2.7 Mark every power net as driven

Why KiCad checks that a power net has a source: a net with no power-output pin on it needs a power flag so the checker is satisfied.

Sources S12

In packpay guarded by `check_pwr_flag_on_reverse_loop` and `check_hierarchical_generation`: a flag is placed where a net would otherwise be undriven, once per net across the sheets.

7.2.8 Run the electrical rules check, and fix every error

Why The check finds unconnected pins, conflicting power nets and similar mistakes before the design goes further.

Sources S13

In packpay guarded by `check_dense_circuit_fixes`, `check_multisheet_hierarchy_kicad_cli` and `check_pwr_flag_on_reverse_loop`: each runs KiCad's own check on what the tool generated and asserts that it reports no error.

7.2.9 Name power nets for their voltage rather than "VCC" alone

Why Two personal guideline collections say never to use "VCC" as a net name: a name such as "+3V3" says which supply it is. Many real designs, this project's own examples among them, keep "VCC", so the advice is contested.

Sources S11, S14

In packpay outside the tool: the names the source uses are kept as they are, and a rail-like name is recognised as a power net (`is_rail_like_name`).

7.3 Sources

Checked on 2026-10-08, and paraphrased here. They are of different kinds, which is why the entries say who claims what.

S1 [SparkFun, How to Use a Breadboard](#) (a tutorial by a supplier, licensed CC BY-SA 4.0).

S2 [Wikipedia, Breadboard](#).

S3 [Zbotic, a beginner's guide to the breadboard](#) (a vendor's guide).

S4 [All About Circuits forum, Tips on Breadboard Wiring](#) (builders' discussion).

- S5 [BusBoard, a 1660-point breadboard datasheet](#) (a manufacturer's specification and FAQ).
- S6 [EEVblog forum, the capacitance of breadboard contacts](#) (builders' discussion).
- S7 [A hobbyist's breadboard reference](#) (an unreviewed markdown note: its figures are the least certain here).
- S8 [Altium, What is schematic capture](#).
- S9 [LCSC, five best practices for professional schematics](#).
- S10 [PCBWay, how to make clear and well organized schematics](#).
- S11 [A personal collection of PCB design guidelines](#).
- S12 two threads of the KiCad forum, on [hierarchical or flat design](#) (global labels, power symbols) and on [power symbols and PWR_FLAG](#) (users quoting KiCad's documentation).
- S13 [A mirror of KiCad's schematic editor documentation](#).
- S14 [Another personal collection of design tips](#) (flow, functional blocks, and never naming a net VCC).
- S15 [ShillehTek, 830-point breadboard manual](#) (a seller's manual: rails often split, some versions continuous).
- S16 [SparkFun, full-size breadboard](#) (a supplier's page and its customers' comments on the gap in the rails).
- S17 [SparkFun, integrated circuit polarity](#) (DIPs straddle the centre of a breadboard; a notch or a dot marks pin 1; numbering is counter-clockwise).
- S18 [Evil Mad Scientist, finding pin 1](#) (notch, dot, triangle or text orientation; the mark labels the corner, not the pin).
- S19 [Embedded.fm, looking out for number 1](#) (pin 1 on chips and connectors; connectors that do not number counter-clockwise).

8 Glossary

The one place the terms of the project are defined; the other documents point here. Code names are written between tildes, as `rows_per_half`. Where the world outside uses a different word for the same thing, the entry says so: breadboard vocabulary is not consistent even among the vendors and the reference works (see the sources at the end).

8.1 Naming decisions

Four words are used in more than one way, by the world or by this project. They are settled here, once.

zone (and sector) The documents, the build instructions ("hole a5 (zone 2)") say *zone*. The code says *sector* for the very same thing (`SectorConfig`, `sector_of`, the `sectors` field of `BoardConfig`). One zone is one block of a board: a trench with a half above it and a half below it, a number of rows in each half, and a number of columns. Vendors and reference works call such a block a "terminal strip" (a board "with two terminal strips" is a two-zone board), but other sources use "terminal strip" for the five-hole group, so the project avoids the word: that group is a *strip* here. In prose, write zone; write sector only when naming the code.

column and row In this project the *column* is the numbered position along the long axis of the board (1 to 63 on a classic board) and the *row* is the lettered position across it (a to e above the trench and f to j below, on a five-row board). Many maker sources use the opposite words (numbered rows, lettered columns: hole "C12" is "column C, row 12"); others number the columns as the project does. The project keeps its own words, and a hole is written the same way under both conventions: letter then number.

strip Five (or `rows_per_half`) holes of one column in one half, which are connected to each other inside the board. The two halves of a column are NOT connected across the trench. The reference works variously call this a terminal strip, a terminal or a five-point terminal.

scheme and schematic *Schematic* is KiCad's word for the circuit diagram (a `.kicad_sch` file). *Scheme* is the project's informal word for the same drawing as an idea, used in prose; the files, the tools and the code always say schematic.

8.2 Breadboard terms

anchor A part with at least three real connections (`anchor_min_pins` is 3): an IC, a connector, a module. It straddles the trench in a block of columns of its own, its pins on row 0 of each half. Every other part is a *passive*. See `classify`, `anchor_pin_col`.

bend The sideways run of a passive's leg. From its hole the near leg goes out for `size_rows` minus one rows, then runs along one row (the bend) to the far lead's column. Nothing else may stand on that run: the hole-collision check refuses it. See `leg`.

board One physical breadboard: a rail, a zone, a rail, a zone, and so on, with a rail before the first zone, one shared between each pair of zones and one after the last. See `BoardConfig`. A design that needs more than one is a multi-board configuration.

- body** The room a passive takes in its column, `size_rows` rows (usually three) starting at its near hole; its far lead stands one row beyond. No other lead or wire lands under it.
- breakout board** A small board that carries a surface-mount part, or one whose pins are not 0.1 inch apart, out to pins 0.1 inch apart so that it plugs into a breadboard. The build instructions say which parts need one.
- classic board** The board the optimizer starts from when the user gives none: one zone, five rows per half, 63 columns. See `classic_board`.
- column** See the naming decisions. Local numbers run 1 to the zone's width; the global number adds the zone's index times `SECTOR_COL_STRIDE` (100000), which is how the layout tells the zones apart.
- DIP** A dual in-line package: an IC with two rows of pins 0.1 inch apart, wide enough to straddle the trench (the usual breadboard IC; the rows are 7.62 mm apart on a "wide" DIP). It puts two pins in each column, one in each half, and its pin 1 is marked by a notch or a dot.
- far leg** The other lead of a passive, in another column, one row beyond the body (near row plus `size_rows`). Its column is chosen when the part is drawn, unless it was reserved. See `leg`.
- half** One side of the trench: north (above, "N" in the code) or south (below, "S"). Each half has `rows_per_half` rows.
- hole** One socket of the board, named by its row letter and its local column number: "a5". A hole of a zone after the first is followed by the zone: "a5 (zone 2)". See `hole_name`, `parse_hole`. The world also calls it a tie point.
- hub** The column of a net that the link pass joins the net's other columns to. When the hub has no free hole left the join continues as a chain from the nearest column that has one.
- jumper** A wire between two holes. In the diagram a purple arc (see `jumper`); in the build instructions "Connect a jumper wire from hole X to hole Y".
- landing row** The row of the hole in which a jumper ends. Chosen by `wire_row` from the free holes of the column (`free_rows`); row 0 beside the trench counts wherever no anchor pin is there.
- lead** A wire leg of a component.
- link** A jumper that joins two columns of one net which column sharing alone did not unite. See `extra_links`.

multi-board configuration Several boards used together, wires running between them ("offboard" steps). See `MultiboardConfig`.

near leg The lead of a passive that goes into the column the part was attached to, at its near row.

passive Any part with fewer than three connections (a resistor, a capacitor, a diode, an LED). See *anchor*.

pin 1 The first pin of an IC or a connector, marked by a notch at one end, a dot, a triangle or a square, or told by the orientation of the printed text. On a dual-row IC the other pins are numbered counter-clockwise seen from above; a connector may number its pins another way. The build instructions list each pin of a part with its hole and, for a dual-row DIP, say which way the notch faces and where pin 1 is, and for a single-row header that pin 1 is at the left (see the best practices).

polarity Which lead of a two-lead part is the positive one. An LED or a diode has an anode (positive) and a cathode; a polarised capacitor has a positive and a negative lead. The build instructions name the role of each lead of such a part, with what tells the leads apart (`polarity_word`, `lead_polarity`); a resistor has no polarity and its step names none.

rail A long power strip along the board, marked red (+) and blue (-). Also called a bus strip, a power rail or a distribution strip. See `RailConfig`. A part wired straight to a rail by the instructions is a rail-direct power source (`rail_direct_power_source`). A net is wired to the rails only if it is a power net (`breadboard_rail_nets`).

reserved far column A column promised, while parts are being placed, to the far lead of a passive, so another part can attach to it; the far-lead search of every other part must keep out of it. See `bridged_far_col`.

row See the naming decisions. Row 0 is the one beside the trench; higher rows lie farther from it.

rows per half The number of rows in one half: 5 on almost every real board. See `rows_per_half`; the optimizer goes beyond 5 only as a last resort (`ROWS_PER_HALF_LADDER`).

size of a part `size_rows`, see *body*.

split rail A rail cut in two partway along its length, so that its halves are not connected unless a jumper bridges them. Most current 830-point boards are said to have them; older boards, some versions and the boards of the project's own configurations do not. A configuration states it with `continuous` set to false, and the build instructions' header then says so.

strip See the naming decisions.

tie point The reference works' word for a hole of the board.

tier One rung of the optimizer's ladder: a number of rows per half and an arrangement of zones and boards. The ladder tries one zone, two, then boards added one at a time (each with one zone, then two), and more rows last. See `escalation_board_tiers`, `walk_escalation`, `compute_optimal_config`.

trench The channel along the middle of a terminal block (the notch, the centre channel, the ditch), across which an IC straddles and which breaks every column in two.

zone See the naming decisions.

8.3 Scheme terms

The scheme side follows KiCad's own words.

footprint The physical land pattern a symbol maps to; a field of the component, carried through the netlist.

global label A label whose name connects every label of the same name in the whole project, across sheets. Power symbols act as global labels.

hierarchical label A label inside a child sheet that connects to the sheet pin of the same name on the parent. See *sheet*.

hierarchical sheet A sheet that stands for a child schematic file; the project can read, partition and generate them.

label Text on a wire that names its net. A local label works within one sheet only.

net One electrical connection: all the pins joined together.

netlist The list of components and the nets that join their pins; what every layout starts from. See `load_from_schematic`.

power flag A symbol (`PWR_FLAG`) that tells the checker a power net is driven although no power-output pin is on it.

reference The unique name of a part in a design (R1, U3), also called the reference designator.

sheet One page of a schematic. A hierarchy has a root sheet and child sheets, joined by sheet pins and hierarchical labels.

symbol The drawn shape of a part, with its pins. The ones a schematic uses are stored inside it (embedded symbols, `embedded_symbol_libs`).

tstamp The timestamp-style unique id KiCad gives a sheet or a part, used to recognise the same child sheet used more than once.

unit One of the parts of a multi-unit symbol (an op-amp with two amplifiers in one package). The project expands each used unit into its own component (`expand_multi_unit`).

value What the part is: 10k, LM358. The instructions carry it.

8.4 Project terms

build instructions The numbered, plain-text steps that tell a person which part goes into which hole. See `hole_name`, `board_note`. The six kinds of step are anchor, passive, `extra_link`, `offboard`, `rail_jumper` and `rail_power_source`.

the loop Schematic to build instructions and back to a schematic, compared with the original netlist: the project's standing proof that the instructions say what the diagram shows. See `sources_from_schematic` and `ppextract_sources` for the files the way back needs.

A/B The same inputs through the old and the new code (a module switch turns the change on and off), compared. A change that should alter nothing must show identical output.

control A test run with the fix switched off, which must reproduce the original failure: it proves the test can fail.

oracle A source of truth independent of the code under test, such as the text of the SVG the code drew.

tripwire A test that passes while a known gap stays open and fails the day it closes, so the gap cannot be forgotten.

8.5 Sources

Outside usage was checked on 2026-10-08, and paraphrased here.

- [Wikipedia: Breadboard](#) (terminal strip, bus strip, tie point; a full-size board has around 56 to 65 rows of connectors).
- [SunFounder](#) (numbered rows, lettered columns; hole C12).
- [Dummies](#) (groups of five called terminal strips; the ditch; bus strips).
- [Keyestudio](#) (five-point terminals; distribution strips).
- [A Scribd document](#) (columns numbered 1 to 50: the opposite convention).

-
- KiCad's own schematic-editor documentation (labels: local, global, hierarchical; sheets; power symbols; PWR_FLAG).