pyfs-matrix cheatsheet

pyflightstream **0.58.2** every subcommand and option, by stage `pyfs-matrix <cmd> -h` (`--help`) gives the full text

Global refusal policy: `--warning-refusals` before the subcommand of pyfs-matrix, pyfs-workspace or pyfs-mesh restores demoted refusals.

Console warnings: `--warning-off` before any subcommand of every program hides warnings, keeping logs and other output. Matrix and workspace also read `[log] warnings = false`; Python uses `console_warnings`. Strict refusals still raise.

Typical flow: `pyfs-mesh inventory` (new mesh) → `plan` (mandatory, no solver) → `run` (`--resume` to add points) → `collect` (cluster jobs) → `post` | **Safe by default:** the pyfs-workspace storage, sync, restore, rebuild and mark-failed only PREVIEW until `--apply`. `--workspace` defaults to the current folder; `--runs NAME` picks another manifest than `runs.json`; a matrix name is found in the workspace root or `inputs/matrices/`.

1. PREPARE THE WORKSPACE AND THE MATRIX

`plot-sweeps` *draw the design of experiments of run matrices as a pair plot of their parameters; reads the matrices only, no executable needed*

- `--name NAME` (required) the figure's file name; it is written to `post/visual-inspect/NAME.pdf` (a `.pdf` ending is not doubled, a folder in the name is refused)
- `--overwrite` replace the figure when it exists; otherwise the file is kept, warned and nothing is drawn
- `--matrices MATRIX [MATRIX ...]` matrix names of the workspace, with or without `.fs`, or `all`
- `--params PARAM [PARAM ...]` the parameters to draw (default: every one that varies); an unknown one is warned and skipped
- `--finished-by products|sims` what makes a point finished, drawn filled (default `products`: a row in its simulation's table under `post/<matrix>/polars/`; `sims`: a run record that finished successfully); every other point is drawn larger, white-faced and on top
- `--workspace DIR` campaign root (default: the current directory)
- `--verbose`, `--log-level LEVEL` console quiet, normal or verbose, as in `plan`

`support [BUILD]` *print registered build support and its evidence; no solver is launched*

See the website's Upgrading page for supported workspace conversion.

2. PLAN (MANDATORY BEFORE RUN, NO SOLVER)

`plan <matrix>` *bind the matrix to `inputs/`, pre-flight every point, write the receipt `run` requires (pinned to the matrix digest; edit the matrix, plan again). Alias: `inspect-setups`*

- `--workspace DIR` campaign root with `inputs/`
- `--name NAME` campaign name (default: workspace folder name)
- `--fs-version V` fallback build for empty FS_BUILD cells
- `--fs-exe PATH` executable override; mandatory for MANUAL rows (else FS_BUILD via `inputs/executables.toml`)
- `--workflow CODE=NAME` FS_SCRIPT code to run type (repeatable)
- `--recipe CODE=MODULE:FUNCTION` LEGACY recipe (repeatable)
- `--point-name TPL` naming template; default `P<sim>-<point name>`; placeholders `{point}` `{alpha}` `{beta}` `{mach}` `{advance_ratio}` `{sim}` `{campaign}`
- `--ignore-missing-families [true|false]` default true: a family the mesh lacks is skipped; false refuses the point
- `--no-ignore-missing-families` same as `false`
- `--cost` table mesh size, TEs, farfield layers, viscous coupling, steady/unsteady, iterations, CPUs and an EXPECTED wall time fitted on this workspace's runs
- `--cost-file NAME` with `--cost` or a grouped plan: price the points by `inputs/costs/NAME.toml`, the machine's own cost model (mesh, steps, CPUs, flags); needed only when several exist
- `--inflow-fft` for qsteady_rotor wheel points in a custom inflow: blade-frame harmonics, k_{eff}, $n_{\max}$, suggested PASSAGE_POSITIONS (warns if too few)
- `--update-ids` (`--updateIDs`) REWRITE this matrix so no POL repeats one used elsewhere; written before the plan runs
- `--accept-unregistered-build` allow an installed build other than the registered one (your responsibility; recorded)
- `--setup-guidelines` write `inputs/setups/SETUP_GUIDELINES.md`
- `--setup-standards` write commented s9XX setup standards
- `--sims SIM [SIM ...]` plan only these simulations, e.g. `--sims 2031 2032`
- `--points POINT [POINT ...]` with `--sims`: only these points, by the point name the plan prints; the matrix is not edited
- `--polar-sweep` plan each simulation as ONE solver job; steady or unsteady polars, never both in a job; with `--local`
- `--batch N` plan the simulations as N solver jobs; steady or unsteady polars, never both in a job; not with `--polar-sweep`
- `--hpc NAME` use `inputs/hpc/NAME.toml` when the workspace holds several profiles
- `--verbose` full warnings (all detail also in `logs/activity.log`)
- `--log-level LEVEL` console quiet, normal or verbose (default: `pyfs.toml` [log] console)
- `--no-accept-unregistered-build` refuse it, whatever `pyfs.toml` or a build entry says

3. RUN

`run <matrix>` *run every active point. Needs a current plan. Linux with a submission profile submits to the cluster; Windows runs locally*

- `--resume` skip points already recorded (a grown matrix runs only new points); without it a recorded point refuses
- `--force-rerun POINT` REDO a recorded point (by point name, run_id or steady JOB id; repeatable). Old records and outputs are archived, nothing deleted. Not with `--resume`
- `--force-rerun-all` REDO every recorded point, SUBMITTED ones included (prints the license count first); it does not cancel an old job: cancel it in the scheduler first
- `--sims SIM [SIM ...]` run only these simulations, space separated, e.g. `--sims 2031 2032`; with `--force-rerun-all`, redo only these
- `--points POINT [POINT ...]` with `--sims`: run only these points, by the point name the plan prints (not with `--force-rerun-all`)
- `--local` run on THIS machine instead of submitting (Linux workstation or smoke test)
- `--polar-sweep` run each simulation as ONE solver job; steady or unsteady polars, never both in a job; with `--local`
- `--batch N` run the simulations as N solver jobs; steady or unsteady polars, never both in a job; not with `--polar-sweep`
- `--hpc NAME` use `inputs/hpc/NAME.toml` when the workspace holds several profiles
- `--progress-every N` local unsteady progress every N steps (default 10; 0 = silent)
- `--parallel N|auto` local simulations run at once (default 1, at most 8; `auto` = 2 within seats and memory)
- `--workflow CODE=NAME` run type: `steady`, `unsteady`, `unsteady_rotor`, `qsteady_rotor`
- `--workspace`, `--name`, `--fs-version`, `--fs-exe`, `--recipe`, `--point-name`, `--ignore-missing-families`, `--no-ignore-missing-families`, `--accept-unregistered-build`: as in `plan`
- `--pproc-warnings` print grouped post-processing warnings
- `--verbose` full warnings and one line per repeated item
- `--log-level LEVEL` console quiet, normal or verbose, as in `plan`
- `--no-local`, `--no-accept-unregistered-build`, `--no-pproc-warnings` turn off what `pyfs.toml` turns on

4. COLLECT (CLUSTER OR INTERRUPTED RUNS)

`collect` *for every SUBMITTED record, wait until its outputs are present and SETTLED, collect, assess, update the record, then post*

- `--watch` keep sweeping until nothing is outstanding (default: one sweep)
- `--discard-walltime` mark clock-stopped points failed before post; grouped plans rerun them from the start, default-mode plans need an explicit rerun; keep outputs
- `--watch-interval S` seconds between sweeps (default 60)
- `--interval S` seconds between the two size/time observations (default 2)
- `--rounds N` stop a watch after N sweeps
- `--sims IDS` only these simulations, e.g. `2006,2007`, and their post
- `--no-post` collect only, do not rebuild products
- `--archive` keep replaced post products; default: overwrite in place
- `--hpc NAME` use `inputs/hpc/NAME.toml` when the workspace holds several profiles
- `--check-frozen` REFUSE (not warn) averages touched by a frozen solve or an unread log block
- `--workspace`, `--runs NAME`, `--pproc-warnings`, `--verbose`
- `--log-level LEVEL`, `--no-check-frozen`, `--no-pproc-warnings`: as in `run`

`status` *one row per polar: SIM, POLAR, SWEEP, POINTS (recorded/planned), MATRIX, STATUS (one word, or the count of each); a point no record carries reads `planned`; reads only, writes nothing*

- `--sims 2006,2007` (or `[2006,2007]`) `--matrix STEM` `--status WORD` (repeatable, `FAILED_*`) `--failed`
- `--additional` lists the additional register; `--points` one row per datapoint `--json` `--csv` `--workspace`, `--runs NAME`

5. POST-PROCESS (NO SOLVER, NO SEAT)

`post [matrix]` *rebuild polars, sections and plot tables under `post/<matrix>/`; no matrix = every matrix in the manifest. Overwrite in place by default; `--archive` keeps replaced products*

- `--sims IDS` only these simulations' products, in place (super files kept)
- `--runs NAME` read another manifest (e.g. a rebuilt one)
- `--from-sims` ignore runs.json, build records in memory from `sims/` (runs made outside the package); products go to `post/<matrix>@sims/`
- `--steps-per-revolution N` with `--from-sims`: for a LAST_REVS_AVG window
- `--additional-pproc` reopen the final .fsm of rows with ADDITIONAL_PPROC (no solve, one launch per point) and extract that pproc; 26.124 only. With it: `--fs-version`, `--fs-exe`, `--local`, `--recipe`, `--hpc NAME` (select a profile when several exist; refused without additional post).
- `--diagnostics` print recorded post diagnostics as Markdown, change nothing
- `--strict` exit 3 when a product was skipped by design (e.g. polar under sideslip)
- `--check-frozen` refuse instead of warn (see `collect`)
- `--archive` keep the replaced products under `archive/` (default: overwrite in place)
- `--force-overwrite` explicitly select no archive (asks; `--yes` answers it); cannot be combined with `--archive`
- `--workspace`, `--pproc-warnings`, `--verbose`
- `--log-level LEVEL`, `--no-check-frozen`, `--no-local`, `--no-pproc-warnings`: as in `run`

`noise <matrix>` *rebuild exploratory loading-noise products and list comparisons*

`--workspace ROOT`, `--sims IDS`, `--overwrite`, `--verbose`, `--log-level LEVEL`.

6 TO 8. SYNC, RECORDS AND DISK SPACE: `pyfs-workspace`, PAGE 2

Before a licensed run: `plan` with `--cost`; read the warnings; `pyfs-mesh inventory --clean` a geometry that carries saved actions. **Trace:** `runs.json`, `plan.json`, `post/`, `logs/`.

Copyright (c) 2026 Geovana Neves, CC BY 4.0.
https://github.com/nevesgeovana/pyflightstream. Page 2: every other command-line tool.

Every other command-line tool

pyflightstream **0.58.2** every subcommand and option `<tool> <cmd> -h` (`--help`) gives the full text

The tools: `pyfs-workspace` builds the campaign tree and custom free-stream fields; `pyfs-mesh` holds every mesh operation and the workspace geometry build; `pyfs-geom` runs the parametric geometry and rewrites IGES; `pyfs-qa` is the evidence tooling of the command database (probes and physics runs need a licensed machine); `pyfs-fsi` is the coupling executable the solver calls; `pyfs-manual` is a maintainer tool (the `[manual]` extra); the two `python -m` entries keep an optional workbook (the `[excel]` extra). **Safe by default:** the field operations preview until `--apply`; `pyfs-manual` writes only with `--write`.

pyfs-workspace

campaign trees, free-stream fields and disc profiles, and the records, storage and queries of a workspace: sync, restore, rebuild, mark-failed, mark-converged, search-log-batch, show, log, trace, history, diff, space-in-use, free-space, delete-sims
`--warning-refusals` before the subcommand restores demoted refusals. `--warning-off` before the subcommand hides console warnings only. `init [root]` *create or complete the campaign tree:* `inputs/` *with its library folders,* `sims/`, `post/`, `archive/`*; writes an* `inputs/executables.toml` *template when none exists. Default root: the current folder*
`--verbose`, `--log-level LEVEL`: the console level, as in `plan`. `--workspace DIR` names the root; the positional form is deprecated. `archive <sim_id> --workspace <root>` *zip one recorded simulation under* `archive/` *and remove its folder* `sims/sim_<sim_id>/`*; a simulation the manifest does not record is refused*
`--verbose`, `--log-level LEVEL`: the console level, as in `plan`. `--workspace DIR` names the root; the positional form is deprecated. `migrate [root]` *preview conversion of a workspace from the 0.38.0 format; see Upgrading for the format boundary*
`--verbose`, `--log-level LEVEL`: the console level, as in `plan`.

- `--workspace DIR` the workspace root `--apply` apply the previewed conversion with backups

`calibrate <id>` *derive an unvalidated sector-offset calibration from recorded wheel and sector points*
`--verbose`, `--log-level LEVEL`: the console level, as in `plan`. `--workspace DIR`, `--wheel-run ID`, `--sector-run ID`, `--rotor NAME`, `--matrix NAME`, `--overwrite`. `excel` *the macro-free workbook of the workspace's matrices, the* `[excel]` *extra;* `python -m pyflightstream.workspace.excel` *is the same command (page 2); save and close the workbook first*
`excel create <output>` *create a new workbook; an existing file is refused*
`--verbose`, `--log-level LEVEL`: the console level, as in `plan`. `--workspace DIR` the workspace root (default: the current folder). `excel preview <workbook>` *inspect changes without applying them (a batch JSON and its HTML)*
`--verbose`, `--log-level LEVEL`: the console level, as in `plan`.

- `--workspace DIR` `--batch FILE` the new batch `--direction {read,write}` read imports matrices, write exports
- `--matrix FILE` select a subset (repeatable)

`excel apply <batch>` *apply one fresh reviewed batch (single use)*
`--verbose`, `--log-level LEVEL`: the console level, as in `plan`. `excel cancel <batch>` *cancel a preview without changing inputs*
`--verbose`, `--log-level LEVEL`: the console level, as in `plan`. `excel check <workbook>` *validate its Runs and Dictionary*
`--verbose`, `--log-level LEVEL`: the console level, as in `plan`. `fsi` *structural-only workspace studies*
`fsi campbell <fsi-id>` *write natural frequencies, Southwell fits and crossing provenance*
`--rpm START STOP COUNT`, `--modes N`, `--workspace ROOT`, `--overwrite`. `--verbose`, `--log-level LEVEL`: the console level. `field` *build a custom free-stream file of* `inputs/freestreams/` *from other fields. Every operation below previews until* `--apply`*, writes the file* `<STEM>` *(*`.tst` *STRUCTURED,* `.dat` *UNSTRUCTURED, the form decides) and its provenance record, and takes* `--out STEM` *(required),* `--workspace DIR`, `--apply`, `--overwrite` *(replace an existing* `<STEM>` *file and its record)*
`field mirror <field>` *mirror a field through the plane x = 0, y = 0 or z = 0*
`--verbose`, `--log-level LEVEL`: the console level, as in `plan`.

- `--plane {x,y,z}` **required**: the coordinate whose sign changes
- `--out STEM` `--workspace` `--apply` `--overwrite`

`field move <field>` *translate a field so a source point lands on a target point*
`--verbose`, `--log-level LEVEL`: the console level, as in `plan`.

- `--source-point X Y Z` **required**: the source point (a hub), m, global frame
- `--target-point X Y Z` **required**: where it lands, m, global frame
- `--out STEM` `--workspace` `--apply` `--overwrite`

`field subtract <total> <other>` *total minus (other minus reference), point by point on one grid*
`--verbose`, `--log-level LEVEL`: the console level, as in `plan`.

- `--reference VX VY VZ` **required**: the free stream OTHER was solved in, m/s (0 0 0 only where OTHER is already induced only)
- `--tolerance M` two positions closer than this, per axis, are one point (default 1e-06)
- `--out STEM` `--workspace` `--apply` `--overwrite`

`field time-mean <files>` *the time mean of an unsteady run's per-step fields (files or glob patterns)*
`--verbose`, `--log-level LEVEL`: the console level, as in `plan`.

- `--last K` average only the last K steps given (default: every step)
- `--fluctuation` also write `<stem>.fluctuation.csv`, the per-probe standard deviation (two steps or more)
- `--fluctuation-only` report the fluctuation only, no field (needs `--last`)
- `--vinf M_S` free-stream speed, m/s, the fluctuation summary is also stated against
- `--tolerance M` as in subtract
- `--out STEM` `--workspace` `--apply` `--overwrite`

`field fill-interior <field>` *fill the probes inside the body from the ray outside it*
`--verbose`, `--log-level LEVEL`: the console level, as in `plan`.

- `--r-body M` body radius about the x axis, m (default 0.38)
- `--out STEM` `--workspace` `--apply` `--overwrite`

`profile <shape> [table]` *the actuator-disc profile a row's* `PROFILE` *names,* `inputs/profiles/<STEM>.csv`*, scaled to* `--thrust` *or* `--ct --rho --rpm`*; ELLIPTICAL needs none. sections:* `--pol --family --component --sign --last`*; bp:* `--advance-ratio`*; uni, bp:* `--stations`*; disc:* `--ref --disc`*, or* `--tip-radius --hub-radius --blades`*;* `--out --workspace --apply --overwrite`
`--verbose`, `--log-level LEVEL`: the console level, as in `plan`. `show <run id or alias>` *outcome first, then evidence tails, logs, continuation chain, identity and inputs; coupled points also show the FSI record and last exchange row*

- `pyfs-workspace show 9101_1`; a simulation id alone lists its points and failed cards; `--point NAME` selects one
- `--workspace`, `--runs NAME`, `--json`, `--csv`

`log` *read recorded activity oldest first, including running batches and compacted simulations*

- `pyfs-workspace log --sims IDS --run ID --stage NAME --since 2h --problems --open`
- `--post MATRIX`: groups by category, product family and message shape, with counts and examples; `--sims IDS` narrows them
- `--storage`: storage calls as recorded; `--stage` selects the recorded action
- `--workspace`, `--runs NAME`, `--json`, `--csv`; `--since` also accepts an ISO time

`trace <product path>` *follow the products index to the simulation, post options, effective run identities, digests and provenance paths*

- `pyfs-workspace trace post/demo/polars/polar.csv`; or `--run ID` for the provenance tree (aliases accepted)
- `--workspace`, `--runs NAME`, `--json`, `--csv`

`history <simulation or point>` `pyfs-workspace history` *reads every present and archived record, its continuation and archive stamp; batch folders and compacted simulations are read in place*

- `--workspace DIR` `--runs NAME` `--json` `--csv`

`diff <run-a> <run-b>` `pyfs-workspace diff` *compares versions, digests, solver flags, flight condition and outcome;* `run-id@<archive stamp>` *selects an archived copy; script differences name lines and inputs*

- `--workspace DIR` `--runs NAME` `--json` `--csv`; reads only, writes nothing

`sync {runs|post|fsm|all}` *run on the MAIN workspace; peers in* `inputs/sync-workspaces.toml`*; levels cumulative, runs < post < fsm < all. Nothing in main is deleted;* `inputs/` *untouched but a declared matrix*

- `--apply` change files (default: preview)
- `--from NAME` one peer (default: every non-main one)
- `--prefer-other` on a record conflict take the peer's (default: main wins)
- `--overwrite` on a file conflict archive main's copy and take the peer's
- `--restore` with `--apply`: rebuild records of `sims/` folders no record carries
- `--include-archives` also bring `archive` folders (skipped by default)
- `--workspace`, `--runs NAME`

`restore {runs|storage|products|plan|additional}` *restore a file from* `archive/`

- `--apply` (default: preview) `--stamp STAMP` archive to restore from
- `--matrix STEM` for products and plan, when several matrices have archives
- `--workspace`

`rebuild` *rebuild run records from the folders under* `sims/`

- `--apply` (default: preview) `--out NAME` write records to another manifest (compare before replacing)
- `--all-sims` every simulation folder, recorded or not `--sims 4001,2009` only these
- `--matrix FILE` the matrix the simulations ran from
- `--build-alias BUILD=ALIAS` scheduler name of a build (repeatable), e.g. `26.124=26.12`
- `--inputs-from FOLDER` another origin's `inputs/` for inputs changed after the run
- `--workspace`

`mark-failed` *every record of the named simulations becomes FAILED_MARKED, its old status kept under* `marked`*;* `runs.json` *archived first*

- `--sims 2006,2007` the simulations `--reason TEXT` why, recorded as given
- `--points P1,P2` only these points within the selected simulations
- `--apply` (default: preview) `--workspace`

`mark-converged` *the named points become CONVERGED by your reading, their old status kept under* `marked`*;* `runs.json` *archived first*

- `--sims 2006 2007` `--points AL+020` only these points `--reason TEXT` `--apply` `--workspace`

`search-log-batch` *writes the log of a batch point that has every other file, cut from the job's master log;* `runs.json` *untouched, then* `collect` *judges the point*

- `--sims 2006 2007` only these `--master FILE` the master log in the job folder (several candidates are named, nothing written) `--apply` (default: preview) `--workspace`

`space-in-use` *sizes by top folder, by* `sims/sim_*`*, by extension; changes nothing*

- `--top N` rows per grouping (default 15) `--workspace`

`free-space <m-id>` *run storage recipe* `inputs/management/m<id>.toml`*:* `prune_step_exports, compact_sims` *(zip, restored automatically),* `delete_extensions, post_archives`

- `--apply` (default: preview) `--runs NAME` `--workspace`
- `--list` every path each step touches, with its size and what happens to it

`delete-sims <ids>` *delete simulations, their products and records, e.g. 4001,2009; a note row remains*

- `--sims ID [ID ...]` the simulations to delete (required), e.g. `--sims 4001 2009`
- `--apply` (default: preview) `--runs NAME` `--workspace`
- `--force` also a simulation whose run is still SUBMITTED (refused without it)
- `--matrix-products {points-only|regenerate}` for a matrix product that also holds the deleted points: leave it (marked stale) or rerun its post without them
- `--archive` archive each simulation before deletion; off by default

pyfs-mesh

every mesh operation and the workspace geometry build, mirroring `cfd-mesh`*: refine, audit, inventory and derive panel meshes, trailing edges, morph, repitch, scale, merge, sector to wheel, IGES and parametric meshing; only* `build`, `mark-boundaries` *and* `export-surface-mesh` *with* `--apply` *run the solver*
`--warning-refusals` and `--warning-off` before the subcommand, as in pyfs-workspace. `mark-boundaries <OBJ>` *preview boundary marking from the OBJ sidecar;* `--apply` *runs the licensed solver and saves the marked FSM;* `--fs-exe PATH` *overrides the executable;* `--overwrite` *permits replacing existing products after validation*
`--verbose`, `--log-level LEVEL`: the console level. `export-surface-mesh <FSM>` *preview the surface export of a saved workspace body;* `--apply` *runs the licensed solver and writes* `<stem>.surface.stl` *beside it for probe planning;* `--build ID` *selects the build,* `--fs-exe PATH` *overrides its executable, and* `--overwrite` *replaces an existing mesh only after a successful export*
`--verbose`, `--log-level LEVEL`: the console level. `trailing-edge <mesh> <output>` *write mesh trailing-edge midpoint nodes*
`--from-nodes FILE` import explicit nodes; `--out FILE` name the output. `--verbose`, `--log-level LEVEL`: the console level. `--axis X Y Z`, `--hub X Y Z`, `--unit UNIT`, `--sections N`, `--overwrite`. `convert2midnodes <nodes> <output>` *write a .te.txt from an ANSA SET node list, in any node order*
`--unit UNIT` the coordinates' unit (METER); `--per-family MESH` one file per family from the OBJ's edges; `--overwrite`; `--accept-empty` an empty list writes nothing, exit 0; `--verbose`, `--log-level LEVEL`. `prepare-to-refine <mesh>` *optional: families of an OBJ, or of an ANSA NASTRAN export cut along its FE perimeters, split into <family>_<NNN>_grid or _mixed patches, a refine file and a node list*
`--keep-within-component` select parallel and normal governors per original family, across disconnected layouts. `--max-angle DEG` (45); `--aft X Y Z`; `--unit UNIT` the NASTRAN unit (MILLIMETER), `--obj-unit UNIT` (METER); `--visual-inspect` a PDF with patches then directions per original family, blue parallel and orange normal arcs, P/N labels and bold framed dominant labels; `--overwrite`; `--verbose`, `--log-level LEVEL`. `visual-inspect <mesh>` *a PDF of the top, front, side and isometric views, one color per patch, with marked points and a table*
`--nodes FILE...`, `--te FILE...`; `--cmap NAME`; `--families A,B`; `--box X0 X1 Y0 Y1 Z0 Z1`; `--mesh-unit`, `--points-unit`, `--unit UNIT`; `--iso-from X Y Z`; `--labels`; `--no-edges`; `--perimeter-width PT`; `--dpi N`; `--png`; `--out`; `--overwrite`; `--verbose`, `--log-level LEVEL`. `sector-to-wheel <sector>` *copy one sector (an OBJ with its boundaries file, or a saved simulation, no solver) about an axis into a full wheel of the same kind: seam nodes merged, one blade surface per copy, a boundaries file and rim points files for an OBJ,* `<out stem>.wheel.json` *with the checks; exit 2 for a refusal or a saved layout not recognized*
`--copies N` required; `--blades FAMILY...` required, the surfaces that are blades; `--axis x|y|z|X,Y,Z` (x); `--center X Y Z` (origin; deprecated alias `--hub`); `--tolerance METRES` (1e-4); `--out PATH`; `--overwrite`; `--no-wake-terminations` drops the wake terminations an OBJ sector declares (carried by default); `--verbose`, `--log-level LEVEL` `sector-remesh <sector.fsm>` *rebuild a saved sector's blade, spinner and nacelle as full-quad surfaces (blunt trailing edge, one blade per sector, a nacelle with a flat base; no solver): chord clustered at the leading edge only, constant span cells, every row at one radius from mid blade out, the spinner adapting to the root row;* `<out stem>.remesh.json` *with the checks; a sector of another topology is warned, nothing is written and the exit is 2;* `--verbose`, `--log-level LEVEL`
`--blades FAMILY` required, the one blade; `--chord-cells N`, `--le F`, `--span-cell METRES` required; `--axis x|y|z` (x); `--tip-ratio` (0.6); `--tip-length METRES` (0.2); `--tip-radius METRES` (the blade's largest radius, warned); `--spinner-theta N` (12), `--spinner-layers N` (4), `--spinner-rings N` (7); `--nacelle-theta N` (6), `--nacelle-cells N` (28); `--root-margin-r F` and `--nose-radius-r F` fractions of the tip radius (0.038); `--out PATH`; `--overwrite` `power-off-body <sector.fsm>` *write the power-off body of a saved sector as a saved simulation: a closed spinner and the nacelle of* `sector-remesh` *(the same nodes),* `--copies` *sectors merged across the seams, full quad, no blade;* `<out stem>.body.json`*;* `--verbose, --log-level LEVEL`
`--blades FAMILY` and `--copies N` required; `--axis x|y|z` (x); `--spinner-theta N` (6),

--spinner-rings N (12); --nacelle-theta N (6), --nacelle-cells N (28); --nose-radius-r F (0.038); --tip-radius METRES; --out PATH; --overwrite **merge <a.obj> <b.obj>** *join two trimmed OBJ along their open boundaries within a distance; --threshold LENGTH, --adapt OBJ (the part adapted, the other kept unchanged), --out OBJ, --overwrite; --verbose, --log-level LEVEL*

identify-te <obj> [<nodes>] *write the trailing edges in the solver's node file; found from the mesh, or among an ANSA SET node list*

--families A,B one file each (default: one file of the quad families; without nodes also <obj>.nodes.txt); --all-families every family, not only the quad ones; --max-angle DEG; --aft X Y Z; --overwrite; --accept-empty; --unit UNIT, --simulation-unit UNIT; --verbose, --log-level LEVEL. **repitch <mesh>** *rigid blade pitch change; rebuild the named hub's source quad rings; preserve face order and saved markings; no solver. Requires --morphing FILE and --out PATH; checks in <out>.repitch.json.*

collapse-te <mesh.obj> *collapse the blunt trailing edge of named blades on an OBJ, after any repitch: rim pairs merge at midpoints, the base strip leaves, the face order holds; no solver. Requires --morphing FILE (a [collapse_te] table) and --out PATH; checks in <out>.collapse.json.*

hub-rings <mesh.obj> *rebuild a quad-dominant hub around each blade root as complete rings of quads, the hole remeshed quad-dominant, every other family and the seams kept; no solver. Requires --morphing FILE (a [hub_rings] table) and --out PATH; checks in <out>.hub_rings.json.*

morph <mesh.obj> *morph a sector or a full wheel on the OBJ from one control file: [hub_rings], [repitch], [collapse_te], [wheel] in that order; nothing is written when a step refuses. Requires --morphing FILE and --out PATH; record in <out>.morph.json.*

scale <factor> --mesh <mesh.obj> *scale every vertex of an OBJ uniformly (a positive factor, 0.5 halves every length) about the origin or about --about X,Y,Z; face order, groups, normals and comments kept byte for byte; no solver. --out FOLDER is created and receives the scaled OBJ under the input name and <name>.scale.json (input hash, factor, centre, counts, bounding boxes before and after); default folder <stem>_scaled<factor>/ beside the input. The input is never overwritten; --overwrite replaces a same-name file in the folder.*

iges-grid <a.igs> [<b.igs> ...] *grid the components named on the IGES levels into one OBJ, one family each, junctions conformal, four-sided patches as quads with the chord and span laws; uses the gmsh kernel; no solver. Requires --grid FILE and --out PATH; writes the boundaries file and <out>.json. --component NAME (repeatable) grids only the named components, an unknown or ambiguous name refused with the names found; the junction of two components is measured against both originals. --name-encoding ENC (latin-1; utf-8 for UTF-8 names; never guessed).*

mesh-generated <config.toml> *mesh a parametric configuration into an all-quad OBJ, one group per component, with the acceptance checks in the record; uses the gmsh kernel. Requires --obj PATH; --component NAME meshes one body or lifting surface alone, --controls TOML overrides the mesh controls, --out JSON writes the record.*

prop-slice *cut one blade of a propeller OBJ or IGES at radial stations; write the propeller definition that rebuilds it, one normalized profile per station and the table of chord, pitch, thickness, camber, skew and rake. Requires --geometry FILE, --blades N, --out DIR; --axis DX DY DZ thrust (-1 0 0), --origin X Y Z (mm), --blade-dir DX DY DZ (0 0 1), --stations R_R ..., --unit of an OBJ (mm). A folder already holding an output is refused unless --overwrite.*

mesh <config.toml> *mesh a parametric configuration by strategy 1 from its per-component mesh controls; errors E01 to E16 refuse by name, W01 warns; uses the gmsh kernel. Requires --controls TOML; --dry-run prints the planned counts and the warnings and writes nothing; --out DIR writes <descriptor>.obj and <descriptor>.mesh.json, --descriptor NAME names them, --fsm-template FSM also writes <descriptor>.fsm from a fresh saved simulation. Exit 0 planned or accepted, 1 not accepted, 2 refused.*

build [TARGET] *read <name>.geometry.toml (of the current folder, or of the folder or file TARGET) and print the script, the boundaries after each step and what would be written; with --apply run the solver once with no solve, save the body, read it back and audit it*

--verbose, --log-level LEVEL: the console level.

- --apply run the solver and write the geometry --overwrite replace the outputs of an earlier build
- --build ID the build id, over the instruction's fs_build --fs-exe PATH the executable, over the registry
- --png also draw <name>.preview.png
- --only A,B build only these variants of a [sweep], such as AIL+100 --parallel N up to N variants at once, 1 or 2 (default 1)
- --rows print, per built geometry, the row POL of --matrix with its CONFIGURATION, GEOMETRY and POL cells set; nothing is written into the matrix --matrix FILE the run matrix --rows copies from

accept <names> *record who inspected each named geometry folder and when, in its <name>.build.json*

--verbose, --log-level LEVEL: the console level.

- --workspace DIR the workspace root (default: the current folder)

mark-boundaries <OBJ> *preview offline the marking of the named [trailing_edges.<label>] and [base_regions.<family>] declarations of <stem>.obj.boundaries.toml; with --apply run the licensed solver once and save <stem>.fsm, the file a matrix row names, with its .fsm.boundaries.toml and <stem>.mark.json*

--verbose, --log-level LEVEL: the console level.

- --apply import, mark and save --overwrite mark again after an input changed --fs-exe PATH the executable, over the registry

from-sidecar <geometry> *print the instruction file that builds the same body as the sidecar of a CCS, IGES or raw-mesh geometry; the sidecar and the matrices are never edited*

--verbose, --log-level LEVEL: the console level.

- --apply write <stem>.geometry.toml --build ID the build id (else the rows' newest) --workspace DIR

refine <MESH> [FACTOR] *refine or coarsen a panel mesh from its OBJ into a new geometry folder <stem>_<tag>/ (OBJ, trailing-edge points, the quad families' trailing edges as solver node files and ANSA node lists, boundaries, refine.json, audit); FACTOR multiplies every selected family, else the factors come from the refinement file; the source is never modified*

--keep-within-component share directional governors across each declared component's grid patches. --audit-off writes the level without auditing it and prints the later audit command. --visual-inspect writes <level>.refine-governors.pdf beside its OBJ: the governing families per direction, then the patches and directions pages.

- --families A,B --parallel F --normal F --accept-remesh --accept-unknown --strict-factors --config FILE
- --out DIR --overwrite
- --verbose --log-level LEVEL

audit <OBJ> *audit a panel mesh (topology, trailing edge, panel quality) and write <stem>.audit.json beside it; exits 0 pass, 1 a gate or check failed, 2 refused*

- --against SOURCE --out FILE the figures as CSV
- --verbose --log-level LEVEL

inventory <geometry> *write <stem>.fsm.boundaries.toml beside a saved simulation (from its mesh block) or <stem>.obj.boundaries.toml beside an OBJ (from its groups); names the solver actions saved in a .fsm; no executable needed; --refresh rewrites only the mesh counts and hash of an existing OBJ sidecar*

- --overwrite --clean --verbose --log-level LEVEL

degenerate <geometry> *derive a degenerate geometry, the thin blade first, from a blade mesh (.fsm or OBJ) and write <stem>_thin_blade.obj and its .boundaries.toml beside it; no executable needed*

- --root-offset LENGTH required --kind thin-blade --boundary NAME --overwrite --verbose --log-level LEVEL

pyfs-geom

the parametric geometry pipeline outside the run matrix: run, loop, inspect and move a configuration. Exit 0 accepted, 1 a run failed at a stage, 2 an input refused, 3 incomplete (a check of the stage table not measured). The record is printed as JSON.

run <config.toml> *run the stages S0 to S13 of a configuration and write one record per stage and the report. Requires --out DIR; --set K=V overrides one value, component.key=value (repeatable; the value is TOML), --configuration NAME picks an entry of the registry, --variant V (clean), --until Sxx ends at that stage (S13; S9 needs no kernel).*

doe <config.toml> <sets.toml> *run one report per [[sets]] table of the second file, each in its own worker process, and continue after a failed set; the exit code is the worst of the sets. Requires --out DIR (set-0001, set-0002, ...); --workers N sets in parallel, --configuration NAME, --variant V, --until Sxx.*

validate <config.toml> *resolve the configuration (stage S0 only) and print every value with its unit and source; no kernel.*

configurations <config.toml> *list the components, the variants and the configurations of a file with their members.*

stages *list the stages S0 to S13 with their checks and limits, and which are built in this tree.*

identify <original.igs> <component> *read the planform and the root, middle and tip CST sections of a lifting surface from its trimmed skin; uses the gmsh kernel. --host NAME (repeatable; FILE#NAME also), --root auto|plane|host|boundary, --mirrored and --no-mirrored, --cut plain|bridge_te|mirror, --seed NACA, --span-axis X Y Z, --stations N, --grid N, --section-tolerance-mm MM, --section-degree N, --section-max-degree N, --host-tolerance M, --no-converge (skip the coarse pass), --out JSON, --config-out TOML (a configuration that rebuilds the surface).*

check <rebuilt.igs> <original.igs> *measure the rebuilt surface against the original, forward and restricted reverse, under a named profile; sampled upper bounds; exit 0 accepted, 1 refused, 2 refused; uses the gmsh kernel. The local chord is --chord-m M or --stations-m S... with --chords-m C... along --axis 0|1|2. --component NAME, --exclude-face INDEX (repeatable), --forward-count N, --reverse-count N, --edge-count N, --fold-y (half-body original), --profile NAME (study_default), --out JSON.*

schema *print the JSON Schema of the component parameter record; no gmsh. --out JSON writes it, --validate RECORD checks a record instead (exit 1 on a failed pin), --asset-root DIR re-reads every pinned file against its SHA-256.*

add-control-surface <iges> <component> <entry.toml> *add one control surface, the entry of a [[control]] table, to a component of an IGES file and write the edited IGES, with the closure checks measured at 0.01 mm; the source is never changed. --out IGES the file to write.*

split-symmetry <iges> *cut a full model at its symmetry plane and write the half model: faces crossing the plane are cut on it, the positive side is kept with its component names and the cut edges stay open as the symmetry boundary; surfaces only, the source is never changed. --out IGES, --plane x|y|z (y), --component NAME cuts one component only.*

iges-convert <a.igs> [<b.igs> ...] *rewrite IGES files as one IGES of B-spline faces, one NAME per component on its own level, the first file's unit; uses the gmsh kernel. Requires --out PATH; record in <out>.json. --name-encoding ENC (latin-1; utf-8 for UTF-8 names; never guessed).*

pyfs-fsi

the coupling executable the solver's aeroelastic toolbox calls once per time step. With no arguments it runs one coupling step in the current folder: coupled when config.json is there, the dummy (zero displacements) otherwise

--warning-off before the subcommand hides console warnings only. **init-dummy** *write the dummy configuration into a run folder*

- --node-count N **required** --out DIR run folder (default: current); --dir is its deprecated alias

step *execute one coupling step explicitly*

- --out DIR run folder (default: current); --dir is its deprecated alias

external *run one coupling call of an external structural program: run it, archive and validate the call*

- --out DIR run folder (default: current); --dir is its deprecated alias

beam <geometry> *cut each blade family's sections from the mesh into <geometry>.<family>.beam.toml; --families A,B, --stations N; --visual-inspection (or --visual-inspect) also draws .beam.pdf; --verbose, --log-level LEVEL*

pyfs-qa

evidence tooling of the command database: probes run the solver locally on a licensed machine

--warning-off before the subcommand hides console warnings only. **probe** *run command-validity probes and write the compat report pair*

- --fs-version V **required**: canonical build, e.g. 26.120
- --fs-exe PATH **required**: the executable (never guessed)
- --commands LIST comma-separated subset (default: every command with a probe spec)
- --identity-only judge no command: capture the solver's identity banner, the way a new build gets its number into a committed report
- --fsm FILE local saved simulation for preludes that need one
- --workroot DIR scratch root (default runs/probes)
- --timeout S per-probe limit, seconds (default 120); solved probes keep their own limits (180/240 seconds; 26.125 hang guards: 60 seconds)
- --report-dir DIR (default reports/compat) --label TEXT report stem suffix

apply-compat <report> *promote database statuses from a committed compat report*

- --root DIR repository root the report is cited against (default .)
- --commands-dir DIR chapter YAML directory to promote into, under --root (default <root>/src/pyflightstream/commands)

physics *run the physics matrix of a workspace, reduce its records, write the physics report*

- --workspace DIR workspace with the physics matrix at its root (default: current folder)
- --matrix FILE (default matriz_physics.fs) --name NAME campaign name
- --resume skip points already recorded
- --report-dir DIR (default reports/physics) --label TEXT

drift *run the physics matrix on two builds, each under its own registry, and diff the reductions*

- --fs-versions A,B **required**: baseline first
- --fs-exe VERSION=PATH **required**, repeatable: one executable per version
- --workroot DIR where the two sides are made (default runs/drift); a side that exists is refused
- --name NAME campaign name (default: a_<build> and b_<build> per side)
- --workspace, --matrix, --report-dir, --label: as in physics

cases *print the Tier 3 test matrix, one line per case id; nothing runs*

- --include-smi also the local-geometry class, never listed implicitly

cost <campaign> *wall-time cost per simulation and point from runs.json, one column per build; runs nothing*

- --compare BASELINE,CANDIDATE two build columns, baseline first

update-reference <case> *update or seed one physics reference from a committed physics report*

- --from-report FILE **required** --reason TEXT **required**, recorded in the reference
- --root DIR repository root the report is recorded relative to (default .)
- --references-dir DIR reference directory to write (default: the package's qa/references/)
- --drop METRIC a reference metric the report does not carry, removed deliberately; repeatable

pyfs-manual

maintainer tool, the `[manual]` extra: compares a vendor manual with the command database. The manual path is passed explicitly and nothing of it is copied
`--warning-off` before the subcommand hides console warnings only. `coverage` *what the manual documents and the database does not*

- `--manual PDF` `--source ID` `--chapter-pages FIRST-LAST` all **required**
- `--index-pages FIRST-LAST` the Script Index (optional)

`draft` *render database entries for the absent commands, for review*

- `--manual`, `--source`, `--chapter-pages`, `--index-pages`: as in coverage
- `--fs-version CANONICAL` **required**, repeatable
- `--only TEXT` filter on the command name `--out FILE` destination
- `--write` actually write (default: print, touch nothing) `--overwrite` replace an existing `--out` file

`sweep` *read every edition; report what no entry covers*

- `--editions MANIFEST` **required**: YAML, one row per build
- `--by-section` group by section `--fail-if-absent` exit 1 when a documented command cannot be emitted

`surface` *the command surface of each edition and what changed between builds*

- `--editions MANIFEST` **required**, in release order
- `--names` list the names, not only counts `--markdown` render as Markdown

`citations` *re-read the database's page citations against the manuals*

- `--editions MANIFEST` **required**

`register` *write documented rows for a new edition where the documentation did not change*

- `--editions MANIFEST` **required** `--fs-version CANONICAL` **required**
- `--commands-dir DIR` chapter files to edit (default: the installed package)
- `--write` actually write (default: print)

python -m pyflightstream.workspace.excel

the optional macro-free workbook, the `[excel]` extra; save and close it first

`create <output>` *create a new workbook*

- `--workspace DIR` the workspace root (default: the current folder)

`preview <workbook>` *inspect changes without applying them (a batch JSON and its HTML)*

- `--workspace DIR` (default: the current folder) `--batch FILE` **required**
- `--direction {read,write}` **required**: read imports matrices, write exports
- `--matrix FILE` select a subset (repeatable)

`apply <batch>` *apply one fresh reviewed batch (single use)*
`cancel <batch>` *cancel a preview without changing inputs*
`check <workbook>` *validate its Runs and Dictionary*

python -m pyflightstream.workspace.excel_bridge

the text bridge the workbook's own controls call: `<action>` is `preview`, `apply` or `check`; `--request FILE`, `--response FILE` and `--batch FILE` are all ***required***

Copyright (c) 2026 Geovana Neves, CC BY 4.0.
https://github.com/nevesgeovana/pyflightstream

1. The workspace and its input library

A workspace is the folder the package owns. The input library is what you bring, staged once and cited by identifier; the run records are what the package writes. Folder names are never authoritative.

What it is for

Create the tree once, stage each geometry under `inputs/geometries/`, write its boundary inventory, and build any custom free-stream field the study needs. Every later stage cites these files by identifier (a geometry by its file name, a reference, setup or pproc by its coded id), never by a path.

Figure: the workspace tree

```
<workspace>/
├─ inputs/ the input library
│  ├─ geometries/ .fsm, .obj, .stl and <stem>.boundaries.toml
│  ├─ references/ setups/ pproc/ r###, s###, p### (stage 3)
│  ├─ profiles/ freestreams/ custom free-stream fields
│  ├─ calibrations/ fsi/ structural inputs
│  ├─ hpc/ one submission profile (stage 5)
│  ├─ matrices/ a home of the run matrices
│  ├─ management/ storage recipes m<id>.toml
│  ├─ executables.toml the build registry
│  └─ INPUTS.md input_template.md generated guides
├─ sims/sim_<POL>/ one folder per simulation
│  ├─ inputs/ scripts/ staged inputs, solver scripts
│  └─ datapoints/DP-<point>/ one folder per point
├─ post/<matrix>/ plan receipt and products
├─ archive/ archived records and simulations
├─ logs/activity.log every command's log
├─ runs.json the run records
└─ storage_management.json every storage call
```

Commands

`pyfs-workspace init [root]`
Create or complete the tree: every `inputs/` folder, `sims/`, `post/`, `archive/`, an `inputs/executables.toml` template when none exists, and the generated guides. Safe to run again; the default root is the current folder.

`pyfs-workspace migrate [root]`
Preview the 0.38.0 workspace conversion (see Upgrading).
- `--workspace DIR` the workspace root.
- `--apply` apply the conversion with backups.

`pyfs-mesh inventory <geometry>`
Write `<stem>.<kind>.boundaries.toml` beside a saved simulation (kind `fsm`, from its mesh block) or an OBJ (`obj`, from its groups), and name the solver actions a saved simulation keeps. Needs no executable.
- `--overwrite` rewrite a saved simulation's sidecar that already exists.
- `--clean` remove the saved solver actions, keeping `<file>.bak-<stamp>` (stage 8).

`pyfs-mesh degenerate <geometry>`
Derive the thin blade of a blade mesh (a saved simulation or an OBJ): the sheet midway between its two sides, written beside the source with its boundary inventory. Needs no executable.
- `--root-offset LENGTH` required: how far the root moves outward along the span, in the mesh's unit.
- `--kind thin-blade` the one kind, the default.
- `--boundary NAME` the blade's boundary, for a file that also holds a spinner or a nacelle.
- `--overwrite` rewrite an existing output.

Custom free-stream fields

Each operation writes one file of `inputs/freestreams/`, named by `--out STEM` (`.txt` STRUCTURED, `.dat` UNSTRUCTURED, the form decides), with its provenance record. Each previews until `--apply`.

`pyfs-workspace field mirror <field>`
Mirror a field through the plane x = 0, y = 0 or z = 0.
- `--plane {x,y,z}` required: the coordinate whose sign changes.
- `--out STEM` required: the stem of the file written.
- `--workspace DIR --apply` writes (default: preview); `--overwrite` replaces an existing file and its record.

`pyfs-workspace field move <field>`
Translate a field so a source point lands on a target point.
- `--source-point X Y Z` required: the source point (a hub), m, global frame.
- `--target-point X Y Z` required: where it lands, m, global frame.
- `--out STEM --workspace`, `--apply`, `--overwrite`: as in mirror.

`pyfs-workspace field subtract <total> <other>`
Total minus (other minus reference), point by point on one grid.
- `--reference VX VY VZ` required: the free stream OTHER was solved in, m/s.
- `--tolerance M` positions closer than this per axis are one point (default 1e-06).
- `--out STEM --workspace`, `--apply`, `--overwrite`: as in mirror.

`pyfs-workspace field time-mean <files>`
The time mean of an unsteady run's per-step fields (files or glob patterns).
- `--last K` average only the last K steps given.
- `--fluctuation` also write `<stem>.fluctuation.csv`, the per-probe standard deviation.
- `--fluctuation-only` the fluctuation report alone, no field; needs `--last`.
- `--vinf M_S` the free-stream speed the fluctuation is also stated against.
- `--tolerance M` as in subtract.
- `--out STEM --workspace`, `--apply`, `--overwrite`: as in mirror.

`pyfs-workspace field fill-interior <field>`
Fill the probes inside the body from the ray outside it.
- `--r-body M` the body radius about the x axis, m (default 0.38).
- `--out STEM --workspace`, `--apply`, `--overwrite`: as in mirror.

Actuator-disc profiles

`pyfs-workspace profile` sections (`--pol`), uni or bp writes `inputs/profiles/<STEM>.csv`, a row's `PROFILE`, scaled to `--thrust` or `--ct`. ELLIPTICAL needs no file.

Mesh levels

`pyfs-mesh refine <MESH> [FACTOR]`
A level in a new geometry folder `<stem>_<tag>/` (OBJ, trailing-edge points, boundaries, `refine.json`, audit); the source is never modified. FACTOR multiplies every selected family.
- `--families A,B` only these; `--parallel F`, `--normal F`: one direction of the grid families.
- `--keep-within-component` prepare-to-refine and refine: share parallel and normal governors across each component's grid patches, including disconnected layouts.
- `--audit-off` Write the level without running its audit; print the command for auditing it later.
- `--accept-remesh` a directional family with no grid is remeshed, warned, not refused.
- `--config FILE` default `<stem>.refine.toml`: `[families.<name>]` `factor`, `parallel_factor`, `normal_factor`, `method`, `elements`; `[refine]` `tag`, `elements`; `[components]`; `[[perimeter]]` `file`, `export_te_per_family`.
- `--out DIR` default: beside the source's folder; `--overwrite` replaces a level.

`pyfs-mesh audit <OBJ>`
Writes `<stem>.audit.json`: G1, G2, G4 alone, G1 to G6 and relative checks with `--against SOURCE`. Exit 0 passed, 1 failed, 2 refused.
- `--out FILE` also the figures as CSV, never a file the audit reads.

Files

- **R** `inputs/geometries/` the saved simulation or mesh `pyfs-mesh inventory` reads.
- **W** `<stem>.<kind>.boundaries.toml` beside the geometry, kind `fsm` or `obj`: the boundary names every alias and pproc family cites.
- **W** `inputs/executables.toml` the build registry template: each build a row may name, its executable.
- **R** `pyfs.toml` with `pyfs.local.toml` (this machine's, read over it): `[defaults]` of the options, `[log] console`; a typed option wins. `accept_unregistered_build = true` on an `inputs/executables.local.toml` entry accepts that build only.
- **W** `inputs/geometries/README.md` how to lay out the geometries.
- **W** `inputs/INPUTS.md` the glossary of every matrix column and input key.
- **W** `inputs/input_template.md` an example of every input file.
- **W** `inputs/pproc/VARIABLES.md` with `WRITING-EQUATIONS.md`: pproc variables and equations.
- **W** `inputs/freestreams/<STEM>` a custom free-stream field and its record.

Common errors

! `pyfs-workspace init`: an ambiguous artifact id, two files of `geometries/` or `profiles/` with one stem (`wing.fsm`, `wing.stl`).
Fix: the id is the stem: rename or remove one. The tree is still completed.

! The inventory refuses: the sidecar already exists.
Fix: `--overwrite` rewrites a saved simulation's sidecar; an OBJ's never: edit or remove it.

! The field file `<STEM>` already exists.
Fix: another `--out`, or `--overwrite` it and its record.

! `--fluctuation-only` refuses.
Fix: it needs `--last` K: the fluctuation of the last K steps.

! A GEOMETRY key naming an OBJ, or a bare stem, is refused.
Fix: mark an OBJ once, `pyfs-mesh mark-boundaries --apply`, and name the `.fsm` with its extension.

Where to read more

Documentation: *The workspace and the workflow*, *The input library and the solver presets*, *Field operations*; guide 2, *Workspaces*.

Copyright (c) 2026 Geovana Neves, CC BY 4.0.

https://github.com/nevesgeovana/pyflightstream. Every command and option: `<tool> <cmd> -h`, or pages 1 and 2 of this cheatsheet.

2. The run matrix and its two homes

The run matrix is the study: one row per polar, each naming its geometry, its three input artifacts, its flight condition and what it sweeps. The row says what is different; the library says what is shared.

What it is for

A matrix is a pipe-delimited text file, `<name>.fs`, one row per simulation. `POL` becomes the simulation id (`sims/sim_<POL>/`). `FLIGHT_CONDITION` states the flow as `KEY:value` pairs, and exactly one key carries the word `sweep`; `SWEEP_VALUES` lists its values, so one row is several points. `-` in a column states nothing.

The columns, in order

`POL` `HIDDEN` `RUN`	id; solver window shown; does the row take part
`AIRCRAFT_CONFIGURATION`	what the row is (labels only)
`DESCRIPTION`	your words
`FLIGHT_CONDITION`	the flow, and what is swept
`SWEEP_VALUES`	the swept values
`GEOMETRY`	the mesh file, by its file name
`REF` `SET` `PPROC`	the coded ids `r###`, `s###`, `p###`
`SYMMETRY` `SYMMETRY_LOADS`	what was meshed, which loads come back
`NCPUS` `WALLTIME`	processors; wall clock with its unit (`4h`)
`FS_BUILD`	the build the row wants, e.g. `26.123`
`WORKFLOW`	the run type: `steady`, `unsteady`, `unsteady_rotor`, `qsteady_rotor`, or `LEGACY` (a recipe)
`VAR_NAMES_VALUES`	every other key, `KEY: value` pairs separated by `/`

Reading one row

`POL`	`7002`: the simulation `sims/sim_7002/`
`FLIGHT_CONDITION`	`TASmps:30.0, REmi:1.20, ALPHA:sweep`
`SWEEP_VALUES`	`0.0,2.0`: two angles, two points
`GEOMETRY`	`wing.fsm`, staged in `inputs/geometries/`
`REF` `SET` `PPROC`	`r003 s002 p001`
`FS_BUILD` `WORKFLOW`	`26.120 steady`
`VAR_NAMES_VALUES`	`-`: velocity comes from `FLIGHT_CONDITION`

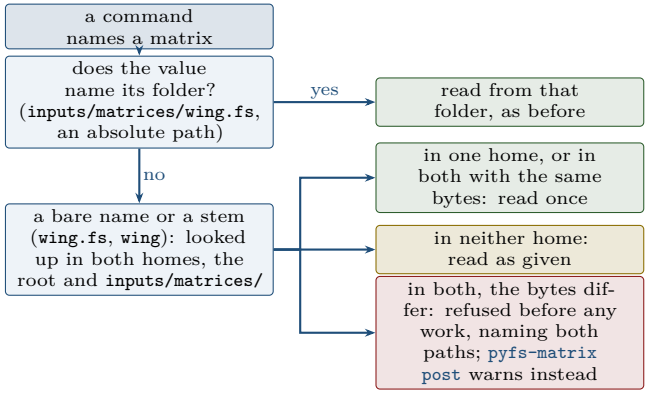
Several matrices in one workspace

Each matrix keeps its own `post/<matrix>/` folder, named after its file stem, with its own plan receipt. A POL may be used by one row of one matrix only: `pyfs-matrix plan` finds a repeat, and its option to renumber is on stage 4.

How a point is named

The point name writes each flight-condition variable as a code and a fixed-width integer: `M144RE438AL+000BE+000J+080`. Scripts and exports are `P<sim>-<name>`, the run id is `<campaign>/sim_<POL>/<name>`, and a steady row is one job, `<campaign>/sim_<POL>/sweep`.

Figure: the two homes and the one lookup



Matrix commands use this lookup, including `pyfs-matrix plan`, `pyfs-matrix inspect-setups`, `pyfs-matrix run`, `pyfs-matrix post` and `pyfs-workspace rebuild`.

Commands

Workspace conversion is documented on the website's Upgrading page.

The optional workbook (the excel extra)

Save and close the workbook first. A read imports matrices into its Runs sheet, a write exports Runs to matrices; both go through a preview, then an apply.

`python -m pyflightstream.workspace.excel create <output>`
Create a new macro-free workbook.
`--workspace DIR` required: the campaign it belongs to.

`python -m pyflightstream.workspace.excel preview <workbook>`
Write a batch, its JSON and an HTML beside it; change nothing.
`--workspace DIR` required.
`--direction {read,write}` required: read imports, write exports.
`--batch FILE` required: the pending transaction; keep it unchanged.
`--matrix FILE` select a subset (repeatable).

`python -m pyflightstream.workspace.excel apply <batch>`
Apply one fresh reviewed batch; each batch is single use.

`python -m pyflightstream.workspace.excel cancel <batch>`
Cancel a preview; nothing changes.

`python -m pyflightstream.workspace.excel check <workbook>`
Validate its Runs and Dictionary.

Files

R `<name>.fs` in the workspace root or in `inputs/matrices/`, the two homes.
R `inputs/sync-workspaces.toml` names the workspace that owns each matrix (stage 6).

Common errors

! One stem in both homes with different bytes: every command refuses, naming both paths.
Fix: keep one copy, or make the two identical.

! The plan warns that the matrix lies outside the workspace's matrix folders.
Fix: move it to the root or `inputs/matrices/`: `pyfs-workspace sync` and the repeated-POL check see only those two.

! A WALLTIME cell is a bare number.
Fix: write its unit: `240m`, `4h`, `90s`, `1d`.

! A fact is stated in its column and as a key of `VAR_NAMES_VALUES`.
Fix: refused naming both: keep one home for it.

! A row moved off LEGACY keeps an `FSM_FILE` key.
Fix: refused at plan naming the key: rename it to `GEOMETRY`.

Copyright (c) 2026 Geovana Neves, CC BY 4.0.
https://github.com/nevesgeovana/pyflightstream. Every command and option: `<tool> <cmd> -h`, or pages 1 and 2 of this cheatsheet.

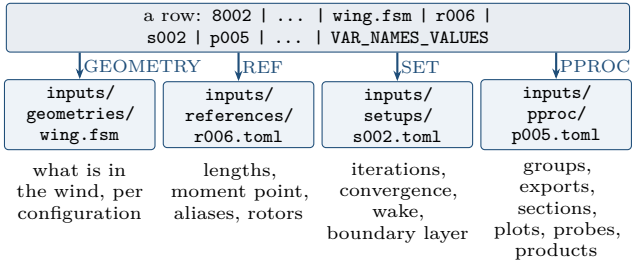
3. The reference, setup and pproc artifacts

Three small files a row cites by coded id: the reference **r###** (what the numbers are measured against and what the parts are called), the setup **s###** (how the solver runs) and the pproc **p###** (what is written down afterwards).

What it is for

A study is many runs that share almost everything. What they share lives in these files, split by how often it changes, and a row names one of each: `REF`, `SET` and `PPROC`. The letter of a coded id declares its kind, so a mistyped number cannot reach another kind's file.

Figure: what a row resolves to



The reference r###

`area_m2`, `chord_m`, `span_m` and `[moment_point]` are what every coefficient is divided by and taken about. `[aliases]` names sets of boundaries once (`airframe = ["Body", "Base"]`); a member the opened mesh lacks is left out, so one reference serves several geometries. `[study]` with `surfaces = ["Wing", "Body", "Sting"]` lists every surface of a study once: a setup or pproc name that a row's mesh lacks is then skipped (and said once in the plan) where the study holds it, and refused as a misspelling where it does not, so one setup, one pproc and one reference serve every model. A block with `kind = "rotor"` is a rotor: hub, `axis`, `rpm_sign`, `diameter_m`, its blade families; a row then names the rotor and nothing else about it.

A rotor block of the reference

```
[PORT]
kind = "rotor"
x_m = 4.5            # the hub; y_m, z_m likewise
axis = "X"           # what it turns about
rpm_sign = 1         # which way
diameter_m = 3.6576 # what J resolves against
families_blades = ["Blade1"] # the count is its length
```

Starting a library

Copy a working `inputs/` whole, delete the artifacts you do not need, edit the reference to your own lengths and boundary names, then write one matrix row and plan it.

The setup s###

The knobs set once for a study: iterations, convergence, the wake, the boundary-layer type, `solver_model`, `moments_model` (`PRESSURE` or `VORTICITY`). The time step is not here: it is what a point is, so it lives on the row. A row may state setup keys over its preset in `VAR_NAMES_VALUES`, under their native names, for that row only: the plan warns, a key the preset does not state or states equally is accepted, and the run record lists them under `from_row`.

The pproc p###

Boundary groups, exports, sectional distributions, force plots, probe lines, sampled fields and the product tables. The frame decides how an entry expands:

`frame =`	you get
`MRP`, or a declared frame	one, over the whole cited set
`SMRP` or `RMRP`	one per rotor, in its own frame
`LOCAL_AXIS`	one per blade, in its frame

A second artifact can be extracted after the run, with no solve: `ADDITIONAL_PPROC: p<id>` on the row and `pyfs-matrix post` with its additional post (stage 7).

Commands

These options of `pyfs-matrix plan` concern the artifacts; the rest of the plan is stage 4.

`pyfs-matrix plan <matrix>`
Bind the matrix to the library and pre-flight every point.

- `--setup-guidelines` write `inputs/setups/SETUP_GUIDELINES.md`, keeping your edits.
- `--setup-standards` write the commented s9XX setup standards, keeping your edits.
- `--ignore-missing-families [true|false]` true (default): a family the opened mesh lacks is left out; false refuses the point.
- `--no-ignore-missing-families` the same as `false`, as a flag.

Files

- **R** `inputs/references/r006.toml` the reference a row's REF cell names.
- **R** `inputs/setups/s002.toml` the setup preset of the SET cell.
- **R** `inputs/pproc/p005.toml` the post-processing artifact of the PPROC cell.
- **R** `<stem>.boundaries.toml` the boundary names aliases and families must match.
- **W** `inputs/setups/SETUP_GUIDELINES.md` the setup guidance, on request.
- **W** `inputs/pproc/VARIABLES.md` rewritten when a pproc gains a glossary.

Common errors

- ! A row states a setup key the preset states with another value.
  Fix: refused at plan, naming both values: change the row or the preset.
- ! A rotor row's setup states `vorticity_drag_families` and `moments_model = PRESSURE`.
  Fix: refused at plan: state `VORTICITY` or drop the drag list. Unstated, a rotor row with the list takes `VORTICITY` and the plan warns.
- ! The plan warns that a section or plot group names a family a row's geometry lacks.
  Fix: a misspelled family is named the same way: check the name against `<stem>.boundaries.toml`. The point plans; `--ignore-missing-families false` refuses it instead.
- ! Both `--ignore-missing-families` and `--no-ignore-missing-families` are stated.
  Fix: refused: state one.
- ! A coefficient looks off by a constant factor.
  Fix: check `area_m2`, `chord_m` and `span_m` of the cited reference: every coefficient is divided by them.
- ! An unsteady row states no averaging window.
  Fix: every unsteady row must state one: `LAST_REVS_AVG` in revolutions, or `LAST_ITERS_AVG` in time iterations.

Where to read more

Guide 5, *References*; guide 6, *Solver setup*; guide 7, *Post-processing definitions*; the documentation pages *The reference artifact* and *The post-processing artifact and its products*.

Copyright (c) 2026 Geovana Neves, CC BY 4.0.
https://github.com/nevesgeovana/pyflightstream. Every command and option: `<tool> <cmd> -h`, or pages 1 and 2 of this cheatsheet.

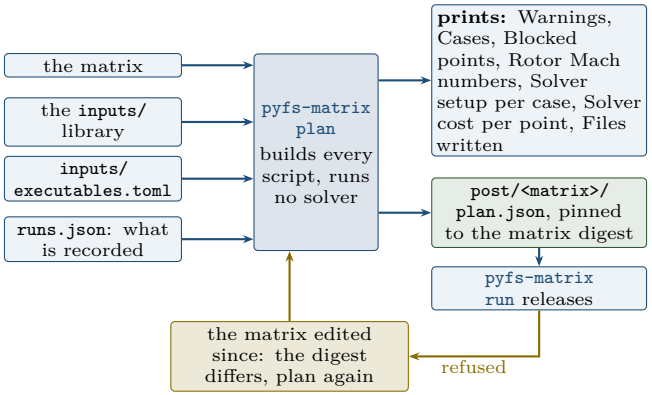
4. Plan: the pre-flight

The plan binds every row to the library, builds every script in a dry run and says READY or BLOCKED with the reason, spending no solver time. It is mandatory: it writes the receipt a run asks for.

What it is for

Always plan before you run: a blocked point costs a message, a bad run costs the license. The plan prints its answer in blocks, writes `post/<matrix>/plan.json` pinned to the digest of the matrix it read, and a run refuses a matrix edited since. Its warnings refuse nothing; read them before you run. Exit code 1 when a point is blocked, 0 otherwise.

Figure: the plan and its receipt



A typical sequence

```
pyfs-matrix plan study.fs --workspace .  --fs-version 26.123
pyfs-matrix plan study.fs --workspace .  --cost
pyfs-matrix run  study.fs --workspace .  --fs-version 26.123
```

The first plan names what blocks; fix it and plan again until every point is READY. The second adds the cost table. The run gives the plan's values again.

The output blocks

Warnings go to standard error; everything else to standard output. **Cases** counts the points ready, blocked and already recorded. **Blocked points** names each one and why. **Rotor Mach numbers**: tip and helical Mach per rotor and point. **Solver setup per case**: model, boundary layer, viscous coupling, aliases. **Files written**: the receipt and any guide. `logs/activity.log` keeps every line.

A grouped plan: several points, one solver

A grouped run solves several points of a polar, or several polars, in ONE solver instance. `--polar-sweep` plans one job per polar; `--batch N` plans N jobs of whole polars, grouped by processors, build, kind (steady or unsteady) and user actions, the longest as short as it can be (at least one job per group, warned). An acoustic polar is a job of its own. The plan prints one line per job (`batch id sims cpus points estimate walltime source`) and every polar *left out of the grouping* with its reason, and records the split in `plan.json`; the run states the same option.

Commands

`pyfs-matrix plan <matrix>`

Pre-flight every point; no solver runs. Its alias is `pyfs-matrix inspect-setups`. The artifact options are on stage 3.

- `--workspace DIR` the campaign root with `inputs/` (default: current folder).
- `--name NAME` the campaign name (default: the workspace folder's name).
- `--fs-version V` the build for rows whose FS_BUILD cell is empty, e.g. `26.123`.
- `--fs-exe PATH` an executable override; mandatory for a row whose FS_BUILD says MANUAL, otherwise the build resolves through `inputs/executables.toml`.
- `--workflow CODE=NAME` a legacy script code to a run type (repeatable), the mapping `pyfs-matrix run` takes.
- `--sims SIM [SIM ...]` plan only these simulations, space separated (`2031 2032`); the matrix is not edited.
- `--points POINT [POINT ...]` with `--sims`: only these points, by the point name the plan prints.
- `--recipe CODE=MODULE:FUNCTION` a LEGACY row's recipe (repeatable).
- `--point-name TPL` the naming template; default `{polar}`, which is `P<sim>-<point name>`; also `{point} {alpha} {beta} {mach} {advance_ratio} {sim} {campaign}`.
- `--cost` table each point's expected cost: mesh size, marked trailing edges, farfield layers, viscous coupling, steady or unsteady, time iterations, processors, and an EXPECTED wall time fitted on this workspace's own runs (`unknown` without a comparable run).
- `--cost-file NAME` price by `inputs/costs/NAME.toml` (`--cost`, grouped estimate) when several exist.
- `--polar-sweep` one job per polar; `--batch N`: N jobs (left).
- `--hpc NAME` `inputs/hpc/NAME.toml`, when several exist.
- `--inflow-fft` for each `qsteady_rotor` wheel point in a custom inflow: the harmonics one blade meets, `k_eff`, `n_max` and the suggested `PASSAGE_POSITIONS`; warns when the row states fewer.
- `--update-ids` (also `--updateIDs`) REWRITE this matrix so no POL repeats one used elsewhere in the workspace; written before the plan runs, and kept if the plan then refuses.
- `--accept-unregistered-build` allow an installed build other than the registered one; its compatibility is your responsibility and every record says so.
- `--verbose` every warning in full, and the started and finished lines of each continuation.

Reading the cost table

The expected wall time is an extrapolation from this workspace's recorded runs, and the table carries the number of samples behind it. A new workspace reads `unknown`: run a few points first, then plan again with `--cost`. A cost file, one per machine, replaces that fit; only a synthetic `examples/costs/c000.toml` ships.

WALLTIME and BEST in a grouped job

A job asks for the sum of its points' `WALLTIME` cells (each budgets ONE datapoint), capped at the profile's `max_walltime` with a warning (a larger N helps); a cell above it is refused. `BEST` prices the job: estimate times 1.25 plus `walltime_margin_s`, up to the minute; an unknown point takes `max_walltime`, warned, or is refused without one. A plan without `--batch` or `--polar-sweep` refuses `BEST`.

Files

- **R** `<matrix>.fs` from either home (stage 2).
- **R** `inputs/` every artifact a row names, and `inputs/executables.toml` for each FS_BUILD.
- **R** `inputs/costs/c<NNN>.toml` the machine's cost file.
- **R** `inputs/hpc/<NAME>.toml` its `max_walltime`.
- **R** `runs.json` which points are already recorded.
- **W** `post/<matrix>/plan.json` the receipt `pyfs-matrix run` asks for, pinned to the matrix digest; grouped: its jobs, polars left out.
- **W** `logs/activity.log` every line the plan printed, and the detail `--verbose` adds.
- **W** `<matrix>.fs` rewritten only by `--update-ids`, each change printed.

Common errors

- ! A point is BLOCKED, with the reason under *Blocked points*.
  Fix: fix the row or the input it names, then plan again; a blocked point never runs.
- ! The build installed is not the one registered for the row's version.
  Fix: refused: install the registered build, or pass `--accept-unregistered-build`, recorded in `plan.json` and on every record.
- ! A row's FS_BUILD cell is empty and no fallback is given.
  Fix: fill the cell, or pass `--fs-version`.
- ! A run type refuses the build: outside the range it covers.
  Fix: the refusal names the builds it covers and the commands that decided it: choose one of them in FS_BUILD.
- ! A POL repeats one stated in another matrix or earlier in this file.
  Fix: `--update-ids` renumbers the repeated rows; a row whose POL already has runs of this matrix is refused rather than moved.
- ! The plan warns: helical Mach of 1 or more on some points.
  Fix: a warning, not a refusal: check the rotor speed of those points.
- ! The plan lists points a `RESTART` key does not continue.
  Fix: `FINISH_PENDING` on a CONVERGED run is refused with the reason; use `ADDITIONAL_REVS` or `ADDITIONAL_ITERS`.
- ! The plan warns that an unsteady row's geometry carries saved solver actions.
  Fix: clean it with `pyfs-mesh inventory --clean` (stage 8); only a grouped plan blocks on it.
- ! A polar is *left out of the grouping*.
  Fix: named, never refused: LEGACY, `RESTART`, `COLD_START` false and coupled `steady/qsteady_rotor` rows; unsteady rows on a build without the action command; polars that do not splice or are recorded. Run them point by point.

Where to read more

The documentation page *Planning a campaign and what it costs.*

Copyright (c) 2026 Geovana Neves, CC BY 4.0.
https://github.com/nevesgeovana/pyflightstream. Every command and option: `<tool> <cmd> -h`, or pages 1 and 2 of this cheatsheet.

5. Run: on this machine or on the cluster

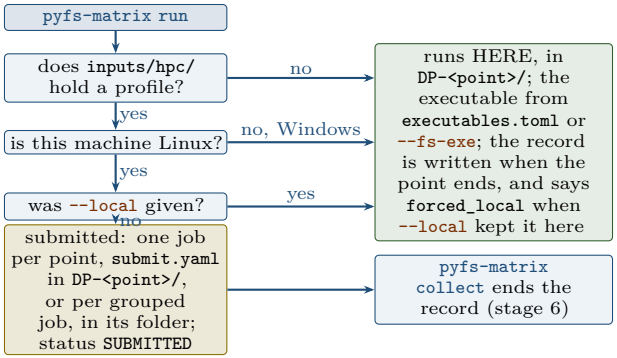
pyflightstream **0.58.2** the campaign workflow by stage, stage 5 of 8

The run builds every active point's script and executes it: on Windows locally, on Linux with a submission profile by submitting each point, or each grouped job, to the scheduler. Every point lands in the records with exactly one status.

What it is for

A run needs a current plan. Each point runs in its own datapoint folder, `sims/sim_<POL>/datapoints/DP-<point>/`, where its outputs are filed; a steady row is one job over all its points, run in the simulation folder. A local point is recorded when it ends; a submitted point is recorded `SUBMITTED` and finished by `pyfs-matrix collect` (stage 6). There is no separate submit command.

Figure: where a point runs



A session on the cluster

```
pyfs-matrix plan study.fs --workspace .
pyfs-matrix run  study.fs --workspace .
pyfs-matrix collect --workspace .  --watch
```

On Linux with a profile the run submits every point and returns; collect then sweeps until nothing is outstanding. On Windows the same run executes the points here and needs no collect.

The submission profile

One `inputs/hpc/<NAME>.toml` per cluster; one needs no option, several need `--hpc NAME` (else refused rather than guessed): the descriptor and its fields, the submit command, `[builds]` (the scheduler's name of each build, e.g. `"26.123" = "26.1"`), `[application_ids]` (the application name of a build where it differs from `application_id`), `[log]` (`export_log`, `native_log`, `job_end_files`) and `max_walltime`.

A grouped run: one solver, many points

`--batch N` or `--polar-sweep` runs the jobs of the plan's receipt, refusing one made for another mode, N or selection, one with no job, or one whose job folders hold files: plan again. A polar's job is `sims/sim_<POL>/FULL-POLAR.txt`; a batch's is `BATCH-<first>-<last>.txt` in `sims/batch/<matrix>_b<ID>/`, the ID the package's. The job's clock replaces each point's WALLTIME. Every point is recorded `SUBMITTED`, also when the job runs here (one instance, one license, one job at a time): collect, then post. Not with `--force-rerun` or `--force-rerun-all`.

Commands

`pyfs-matrix run <matrix>`
Run every active point.

`--resume` skip points already recorded, so a grown matrix runs only its new points; without it a recorded point refuses.
`--force-rerun POINT` REDO a recorded point, by point name, full run id or a steady row's job id (repeatable): the records and outputs are archived first, nothing is deleted.
`--force-rerun-all` REDO every recorded point, SUBMITTED ones included; the count of points and jobs, the licenses it spends, prints first. It does not cancel an old job: cancel it in the scheduler first.
`--sims SIM [SIM ...]` run only these simulations, space separated (2031 2032); with `--force-rerun-all`, redo only these.
`--points POINT [POINT ...]` with `--sims`: run only these points, by the point name the plan prints; refused beside `--force-rerun-all`.
`--local` run on THIS machine instead of submitting: a Linux workstation, or a smoke test.
`--polar-sweep` run the receipt's jobs, one per polar; `--batch N`: its N jobs (left).
`--hpc NAME` `inputs/hpc/NAME.toml`, when several exist.
`--progress-every N` a local unsteady point reports every N steps (default 10; 0 is silent).
`--workflow CODE=NAME` a legacy script code to a run type: `steady`, `unsteady`, `unsteady_rotor`, `qsteady_rotor` (repeatable).
`--workspace DIR` `--name NAME`, `--fs-version V`, `--fs-exe PATH`, `--recipe CODE=MODULE:FUNCTION`, `--point-name TPL`: as in plan; give the plan's values.
`--ignore-missing-families [true|false]` and `--no-ignore-missing-families`: as in plan.
`--accept-unregistered-build` as in plan; every record says it was used and carries the build the solver printed.
`--pproc-warnings` print the post-processing warnings, grouped.
`--verbose` every warning in full, one line per repeated item.

Redoing a point

A correction that does not change a point's name (a pproc, a geometry, a solver variable, a wall clock) keeps its identity: plan again, then `--force-rerun` it. A correction to `FLIGHT_CONDITION` changes the name: it is a new point, and `--resume` runs it beside the old record.

Files

R `post/<matrix>/plan.json` must be current: the digest of the matrix it read.
R `inputs/executables.toml` each FS_BUILD to its executable.
R `inputs/hpc/<NAME>.toml` the submission profile, when the workspace has one.
W `sims/sim_<POL>/scripts/` the generated solver scripts, `P<sim>-<point>.txt`.
W `sims/sim_<POL>/datapoints/DP-<point>/` the point's outputs, its descriptor when submitted, its wall-clock state.
W `runs.json` one record per point or steady job; `runs.json.lock` while it is written.
W `archive/runs-<stamp>.json` the manifest before a forced re-run; each redone point's outputs go to its own `archive/<stamp>/`.

Common errors

! *no plan for this matrix, and a run needs one.*
Fix: run `pyfs-matrix plan` with the same matrix first.

! The matrix was edited after the plan.
Fix: refused until planned again: the receipt is pinned to the matrix's digests; only its identifiers and the spaces around its pairs may change.

! A point is already recorded and refuses.
Fix: the refusal counts the points recorded and those `--resume` would run, and prints the command that continues; `--force-rerun` redoes a point. `--resume` and `--force-rerun` together are refused.

! `--sims` names an id, or `--points` a point, the matrix does not carry; or `--points` comes without `--sims`.
Fix: refused before anything runs, naming the ids or points that exist.

! `--force-rerun` names a point no record carries.
Fix: refused: copy the name or run id the refusal printed.

! `--force-rerun-all` is refused.
Fix: beside `--resume`, `--force-rerun` or `--points`, for a `--sims` id the matrix does not carry, or when nothing of the selection is recorded.

! Several profiles in `inputs/hpc/`, no `--hpc`.
Fix: refused rather than guessed: name one.

! The profile has no `[builds]` line for a row's build.
Fix: refused before any point is submitted: add the line.

! A WALLTIME cell written as a bare number is refused by name.
Fix: write it with its unit (stage 2).

! `--resume` on a simulation a rebuild refused (stage 8).
Fix: never: without a record the resume runs it again. Settle the rebuild first.

Copyright (c) 2026 Geovana Neves, CC BY 4.0.
https://github.com/nevesgeovana/pyflightstream. Every command and option: `<tool> <cmd> -h`, or pages 1 and 2 of this cheatsheet.

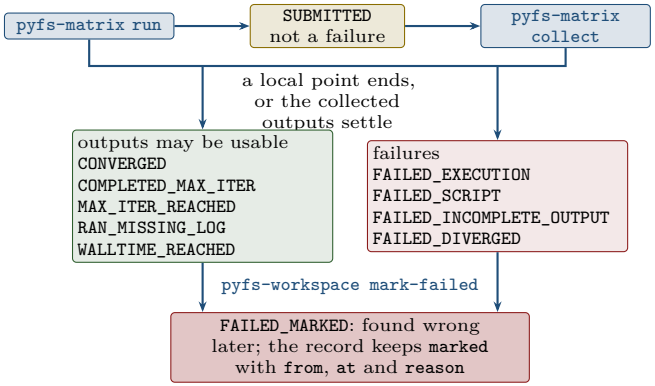
6. Collect and the run records

A submitted point comes back when its outputs land: collect waits for them, files them, judges the run and updates its record. The records, `runs.json`, are the study's evidence; sync keeps a workstation and a cluster in agreement.

What it is for

For every `SUBMITTED` record, `pyfs-matrix collect` waits until the declared outputs are present and SETTLED (the same size and time at two observations), collects them, assesses the run, writes its terminal status, then posts (stage 7). A local run needs no collect. Every executed point lands in exactly one status: a silently skipped point is impossible.

Figure: the life of a run record



undo: `pyfs-workspace restore runs`, the copy archived before the marking; redo: `--force-rerun` archives the record and the point runs again

What a record carries

Which run it was (`run_id`, `point_name`, `sweep_name`), which build ran it, the hashes of the script and the inputs, how the solver was called (`executor`), where it ran (`working_dir`), how it ended (`status`), how long it took and which files it produced. Nothing is inferred from a folder name afterwards.

Read the recorded evidence. `pyfs-workspace show 9101_1` prints the outcome, evidence tails, logs, chain, identity and inputs (coupled: the FSI record, last exchange row); a simulation id lists its points and failed cards, or add `--point NAME`. `pyfs-workspace log` reads activity oldest first: `--sims IDS`, `--run ID`, `--stage NAME`, `--since 2h` (or an ISO time), `--problems`, `--open`. `--post MATRIX` groups saved post warnings by category, family and message shape with a count and example; `--sims IDS` narrows the groups. `--storage` reads storage calls as recorded; `--stage` selects their action. `pyfs-workspace trace post/demo/polars/polar.csv` follows the products index to run identities, digests and provenance; `--run ID` prints a run's provenance tree. All three accept `--workspace`, `--runs NAME`, `--json`, `--csv`, and run ids or aliases where a run is named. `pyfs-matrix status --additional` prints the additional register without deriving fields. They read running batch folders and compacted simulations in place, write nothing, never ask the scheduler.

`COMPLETED_MAX_ITER`: ended, not converged. `MAX_ITER_REACHED`: steady cap, unsettled. `RAN_MISSING_LOG`: outputs, no log evidence. `WALLTIME_REACHED`: clock interruption. `FAILED_MARKED`: a person's failure verdict. A forced re-run archives the record.

Commands

`pyfs-matrix collect`
Collect the outputs of every SUBMITTED record when they land, then post.

- `--watch` keep sweeping until nothing is outstanding (default: one sweep).
- `--discard-walltime` mark each `WALLTIME_REACHED` point `FAILED_MARKED` before post, outputs kept: the next grouped plan (stage 4) runs it again from the start; a default-mode plan needs a forced re-run (stage 5).
- `--hpc NAME` `inputs/hpc/NAME.toml`, when several exist.
- `--watch-interval S` seconds between sweeps under watch (default 60).
- `--interval S` seconds between the two observations that decide settled (default 2).
- `--rounds N` stop a watch after N sweeps.
- `--sims IDS` only the SUBMITTED records of these simulations (2006,2007), and their post.
- `--no-post` collect only, do not rebuild the products.
- `--archive` keep replaced post products; default: overwrite in place.
- `--check-frozen` REFUSE, not warn, averages touched by a frozen solve or an unread log block.
- `--workspace DIR --runs NAME` another manifest; `--pproc-warnings`; `--verbose`.

Tip: `nohup pyfs-matrix collect --watch > logs/collect.out 2>&1 &`

`pyfs-workspace sync {runs|post|fsm|all}`
Run on the MAIN workspace: bring records and results from the others named in `inputs/sync-workspaces.toml`. Levels are cumulative.

- `--apply` change files (default: preview).
- `--from NAME` one workspace (default: every non-main one).
- `--prefer-other` on a record conflict take the other's (default: main wins, unless main's is still SUBMITTED).
- `--overwrite` on a file conflict archive main's copy to `archive/sync-<stamp>/` and take the other's.
- `--restore` with `--apply`: rebuild the records of the `sims/` folders no record carries.
- `--include-archives` also bring folders named `archive` (skipped by default).
- `--workspace DIR --runs NAME`: merge into another manifest.

Levels: `runs` the records with each simulation's scripts and logs; `post` adds `post/`; `fsm` adds each datapoint's saved simulation; `all` adds the rest of `sims/`. A matrix is owned by one workspace, and the owner's copy wins.

Files

- **R** `sims/sim_<POL>/datapoints/DP-<point>/` the outputs and the solver log a SUBMITTED point waits for.
- **R** `sims/batch/<matrix>_b<ID>/` a batch job's folder: its script, descriptor, log, end files and, while it runs, its simulations.
- **R** `inputs/hpc/<NAME>.toml [log] job_end_files`: the files a scheduler writes when a job ends.
- **W** `runs.json` the records: `run_id`, `status`, the build, the hashes of script and inputs, `working_dir`, times; `runs.json.lock` while written.
- **R** `inputs/sync-workspaces.toml main`, each workspace's `path` and the `matrices` it owns.
- **W** `archive/runs-<stamp>.json` the manifest before a sync rewrites it.
- **W** `storage_management.json` one entry per sync source.

Collecting a grouped job

Collect reads the job's descriptor, log and end files in its folder and completes each point as soon as its files settle, while the job still runs. Until a batch ends it copies each finished simulation to `sims/` and leaves the batch folder alone; once it ends, it moves them. It cuts the job's log into one log per point. A point that failed without stopping the instance does not stop the job; the points a stopped job never started are named, and the next grouped run takes them again. A point the job's clock stopped is `WALLTIME_REACHED`: see `--discard-walltime`.

Common errors

! A job died before its log: the point stays SUBMITTED for ever.
Fix: name the scheduler's end-of-job files in the profile, `[log] job_end_files = ["FTS{sim}.o*", ...]`: when they all exist and the log does not, collect records `FAILED_EXECUTION` with their last 20 lines.

! Averages touched by a frozen solve.
Fix: by default the products are written and `post.log` warns; `--check-frozen` refuses them instead.

! The sync is refused: `runs.json.lock` in main.
Fix: a run is writing here: wait for it. A lock in a source skips that source, recorded.

! A stem is declared by two workspaces, or a matrix by none.
Fix: refused: each matrix belongs to exactly one workspace in `inputs/sync-workspaces.toml`.

! `--restore` with `--runs`.
Fix: refused: the restore appends to `runs.json` only.

! A `runs.json` that holds `FAILED_MARKED` is unreadable by an older release.
Fix: keep 0.33.0 or later on every machine sync joins, or restore the archived copy before going back.

Copyright (c) 2026 Geovana Neves, CC BY 4.0.
https://github.com/nevesgeovana/pyflightstream. Every command and option: `<tool> <cmd> -h`, or pages 1 and 2 of this cheatsheet.

7. Post and products

The post builds the product tables from the records, with no solver and no seat: polars, sections, probe tables, per-step series and sampled fields, each with its provenance. A rebuild overwrites derived products in place by default; add `--archive` to keep a copy of what is replaced.

What it is for

`pyfs-matrix run` and `pyfs-matrix collect` post on their own; `pyfs-matrix post` rebuilds by hand, from `runs.json`, under `post/<matrix>/`, the folder named after the matrix stem. Without a matrix it posts every matrix the manifest names. By default what is there is overwritten in place with no archive; with `--archive` it moves to `post/<matrix>/archive/<stamp>/` first. Exit codes: 0 done; 2 refused, nothing written; 3 with `--strict`, a product skipped by design.

Figure: the products of one matrix

```
post/<matrix>/
├─ plan.json the plan receipt (stage 4)
├─ products.json every file, its run ids, skips
├─ post.log post.log.json warnings and refusals
├─ polars/ one polar table per group
├─ sections/ <point>_sections.csv
├─ probes/ <point>_plots.csv, <point>_probes.csv
├─ series/ per-step series of an unsteady point
├─ fields/ sampled velocity fields
│  ├─ <stem>.vtk VTK vertex cloud
│  ├─ <stem>.dat Tecplot
│  ├─ <stem>.inflow.dat reusable inflow, SI
│  └─ <file>.provenance.json beside each field file
└─ archive/<stamp>/ what each rebuild replaced
post/<matrix>@sims/ products of --from-sims
```

Sampled fields and their provenance

Positions are in REFERENCE coordinates and meters, velocity absolute in REFERENCE components and m/s, one file per unsteady STEP. The provenance names the original frame, its recorded motion, the STEP, the coordinate conversion, the source-file hash and the velocity evidence. The convention was measured on one 26.124 build; a field of another 26.124 build is written with one warning per point and, in its `products.json` entry, `velocity_convention` with `proven` false. To reuse an inflow file, state `FREESTREAM_UNITS:SI` on the row.

Commands

`pyfs-matrix post [matrix]`

Rebuild the post-processed products from the records, with no solver.

- `--workspace DIR` the campaign root (default: current folder).
- `--sims IDS` only these simulations' products, in place; the super files stay as the last whole post wrote them.
- `--runs NAME` read another manifest, e.g. one a rebuild wrote.
- `--from-sims` ignore `runs.json`: records assembled in memory from `sims/`, for runs made outside the package; products to `post/<matrix>@sims/`.
- `--steps-per-revolution N` with `--from-sims`: for a `LAST_REVS_AVG` window.
- `--additional-pproc` first reopen the final `.fsm` of each recorded point whose row states `ADDITIONAL_PPROC`, with no solve, and extract that pproc; one solver launch per point; 26.124 only.
- `--hpc NAME` with `--additional-pproc`, select the HPC profile when the workspace holds several; refused for a products-only rebuild.
- `--fs-version V --fs-exe PATH`, `--local`, `--recipe CODE=MODULE:FUNCTION`: for the additional post, as in run.
- `--diagnostics` print the recorded post diagnostics as Markdown, change nothing.
- `--strict` exit 3 when a product was skipped by design, after writing everything.
- `--check-frozen` refuse instead of warn (stage 6).
- `--archive` keep replaced products under `archive/<stamp>/`; default: overwrite in place.
- `--force-overwrite` explicitly select no archive; asks first, and `--yes` answers. Cannot be combined with `--archive`.
- `--pproc-warnings` grouped post-processing warnings; `--verbose` every one in full.

What the tables hold

Every table opens with `POL`. A polar table has one row per point: the reference block, `J`, and the body, stability and wind axis coefficients. The probe table opens with `PROBE, X, Y, Z, FRAME, STEP`; a cell the package cannot fill reads `NA`, never blank.

Files

- **R** `runs.json` the records the products derive from.
- **R** `sims/sim_<POL>/datapoints/DP-<point>/` each point's collected exports.
- **R** `sims/sim_<POL>/profiles/<POL>_probe_points.csv` where each probe was, recorded at emission.
- **W** `post/<matrix>/` the products, `products.json`, `post.log`, `post.log.json`.
- **W** `post/<matrix>/archive/<stamp>/` with `--archive`: what the rebuild replaced.
- **W** `additional.json` the record of the additional post; its files go to `datapoints/DP-<point>/additional/<pid>/`.

Common errors

- ! `--yes` without `--force-overwrite`.
  Fix: refused: it answers a question only that option asks.
- ! A product is skipped by design (a polar under sideslip).
  Fix: named on standard error and under `skipped` in `products.json`; `--diagnostics` prints why. Exit 0, or 3 with `--strict`.
- ! A field of another solver version, unit or export family.
  Fix: refused: *native velocity convention has no evidence for this export/build/unit.* Measure the convention on that build first.
- ! The matrix is in both homes with different bytes.
  Fix: the post warns naming both and builds from the run records; keep one copy.
- ! The additional post on a workspace that submits from here.
  Fix: refused without `--local`: only the local half is built.
- ! A point's `.fsm` does not hash as its record says.
  Fix: skipped by name: the additional post reads only the file the run saved.
- ! An old product is gone after a rebuild.
  Fix: it is in `post/<matrix>/archive/<stamp>/`; `pyfs-workspace restore products` brings back `products.json`.

Where to read more

Guide 7 and the page *Post-processing definitions*, the definition of record of every product; the pages *Sampled fields* and *Extracting more from a finished point*.

Copyright (c) 2026 Geovana Neves, CC BY 4.0.
https://github.com/nevesgeovana/pyflightstream. Every command and option: `<tool> <cmd> -h`, or pages 1 and 2 of this cheatsheet.

8. Maintenance: space, records and saved geometries

pyflightstream **0.58.2** the campaign workflow by stage, stage 8 of 8

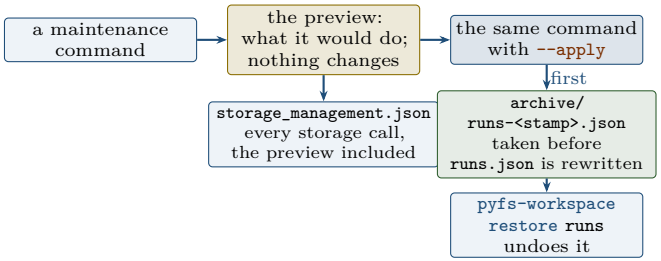
A workspace grows and records go wrong. These commands measure and free space, delete simulations, mark a run failed after the fact, restore or rebuild the records, and clean a saved geometry. Each previews until `--apply`.

1. Workspace and inputs → 2. Run matrix → 3. Reference setup pproc → 4. Plan → 5. Run → 6. Collect and records → 7. Post and products → **8. Maintenance**

What it is for

Run the preview, read it, then apply. Every storage call, the preview included, is recorded in `storage_management.json`, and every rewrite of `runs.json` archives the previous copy first, so `pyfs-workspace restore` can undo it.

Figure: preview, apply, undo



A storage recipe, `inputs/management/m<id>.toml`, runs its tables in this order:

1. `prune_step_exports`: keep only the last step of each per-step export
2. `compact_sims`: zip a simulation folder, restored automatically when needed
3. `delete_extensions`: delete the files of the named extensions
4. `post_archives`: compact or delete the post's archived folders

never touched by a recipe: `inputs/`, a matrix, `storage_management.json`, a saved `.fsm`, `scripts/`, every log, every file a record names or hashes

`pyfs-mesh inventory <geometry>`
Clean a saved simulation that keeps a previous run's unsteady solver actions.
- `--clean` remove them, keeping `<file>.bak-<stamp>`; a 26.120 file in meters is also reset to a fresh import, meshes and boundary conditions kept.
- `--overwrite` also rewrite the existing sidecar.

Commands

`pyfs-workspace history <simulation or point>`
Every present and archived record, continuation link and archive stamp; reads batch folders and compacted simulations in place.
- `--workspace DIR` workspace root; `--runs NAME` selects another manifest and its archives.
- `--json, --csv` the same rows as machine data; writes nothing.

`pyfs-workspace diff <run-a> <run-b>`
Compare versions, digests, solver flags, flight condition and outcome. A run id with `@<archive stamp>` selects an archived copy.
- `--workspace DIR` workspace root; `--runs NAME` selects another manifest.
- `--json, --csv` the same changes as machine data; script changes name lines and inputs.

`pyfs-workspace space-in-use`
Sizes on disk by top folder, by `sims/sim_*`, by extension; changes nothing.
- `--top N` rows per grouping (default 15). `--workspace DIR`.

`pyfs-workspace free-space <m-id>`
Run the recipe `inputs/management/m<id>.toml`.
- `--apply` change files (default: preview).
- `--list` every path each step touches, its size and what happens to it.
- `--runs NAME` also read this manifest; `--workspace DIR`.

`pyfs-workspace delete-sims --sims <ids>`
Delete simulations (`--sims 4001 2009`), their products and records; a note row keeps the ids retired.
- `--apply` change files (default: preview).
- `--force` also a simulation whose run is still SUBMITTED.
- `--matrix-products {points-only,regenerate}` a product that also holds other points: leave it, marked stale, or post it again without the deleted ones.
- `--runs NAME` read and edit another manifest; `--workspace DIR`.

`pyfs-workspace mark-failed`
Every record of the named simulations becomes FAILED_MARKED, its old status kept under `marked`.
- `--sims IDS` required: `2006,2007`.
- `--reason TEXT` why, recorded as given.
- `--apply` write (default: preview); `--workspace DIR`.

`pyfs-workspace mark-converged`
The named points become CONVERGED by your reading, their old status kept under `marked`.
- `--sims IDS` required: `2006 2007`; `--points NAMES` only these points.
- `--reason TEXT` required, recorded as given.
- `--apply` write (default: preview); `--workspace DIR`.

`pyfs-workspace search-log-batch`
Writes the log of a batch point that has every other file, cut from the job's master log; then collect judges it.
- `--sims IDS` only these simulations (default: all).
- `--master FILE` the master log in the job folder, when several candidates exist.
- `--apply` write (default: preview); `--workspace DIR`.

`pyfs-workspace restore {runs|storage|products|plan|additional}`
Bring a records file back from the archive, byte for byte.
- `--apply` write (default: preview); the current file is archived first.
- `--stamp STAMP` the copy to restore (default: the newest).
- `--matrix STEM` for products and plan, when several matrices have archives; `--workspace DIR`.

`pyfs-workspace rebuild`
Make run records again from `sims/`: each matrix row is rendered again and compared with the script that ran.
- `--apply` append the rebuilt records to `runs.json` (default: preview).
- `--out NAME` write them to another manifest instead, to compare.
- `--all-sims` every simulation folder, recorded or not; needs `--out`.
- `--sims IDS` only these: `4001,2009`.
- `--matrix FILE` the matrix the simulations ran from.
- `--build-alias BUILD=ALIAS` the scheduler's name of a build then (repeatable).
- `--inputs-from FOLDER` another origin's `inputs/` for inputs changed after the run; `--workspace DIR`.

`pyfs-workspace archive <sim_id>`
Zip one recorded simulation under `archive/` and remove its folder.

Files

- **R** `inputs/management/m<id>.toml` the storage recipe.
- **W** `storage_management.json` every storage call, `force` and the statuses included.
- **W** `runs.json` rewritten by `pyfs-workspace delete-sims`, `pyfs-workspace mark-failed`, `pyfs-workspace rebuild`, `pyfs-workspace restore`.
- **W** `archive/runs-<stamp>.json` the copy taken before each rewrite.
- **W** `sims/sim_<POL>.zip` a simulation compacted by `compact_sims`, restored automatically when needed.
- **W** `<file>.bak-<stamp>` the geometry before `--clean`.

Common errors

! *have a run still SUBMITTED; collect it first.*
Fix: run `pyfs-matrix collect`, or delete it anyway with `--force`.

! `--apply` refuses, naming a product shared with other simulations.
Fix: choose `--matrix-products` `points-only` or `regenerate`; `regenerate` is refused with `--runs`.

! A simulation id with no record (mark-failed).
Fix: refused by name before anything is written; a record already FAILED_MARKED is left as it is.

! The rebuild refuses an edited or drifted script, naming the input.
Fix: the inputs changed after the run: `--inputs-from` the folder they came from.

! The profile no longer maps an old build.
Fix: `--build-alias` `26.123=26.12`.

! A POL no current matrix holds.
Fix: `--matrix` the file the simulations ran from.

! `--out` names `runs.json` or a file that exists.
Fix: refused before any work: choose a new name.

! The restore is refused while `runs.json.lock` exists.
Fix: a run, collect or sync is writing: wait for it.

! A cleaned geometry has a new hash.
Fix: runs staged before the clean keep theirs; a new campaign records the new one.

Copyright (c) 2026 Geovana Neves, CC BY 4.0.
https://github.com/nevesgeovana/pyflightstream. Every command and option: `<tool> <cmd> -h`, or pages 1 and 2 of this cheatsheet.