Guide 5: The reference file

What `inputs/references/r<id>.toml` holds, and why

Written for pyflightstream 0.58.2.

Sources: the package pages "The workspace and the workflow" and "Getting started", and the input glossary [1, 2].

pyflightstream is the work of Geovana Neves: `https://github.com/nevesgeovana/pyflightstream`, concept DOI `10.5281/zenodo.21482924`.
Copyright (c) 2026 Geovana Neves. Licensed under CC BY 4.0
(`https://creativecommons.org/licenses/by/4.0/`). See LICENSE-AND-AUTHORSHIP.md.

Contents

The reference file

Reference lengths and the moment point

The study's own words

One block per rotor

Actuator discs

Reading the reference back

A complete reference file

Sources under `src/pyflightstream/`: `workspace/inputs.py`; `cases/__init__.py`; `cases/workflows.py`; `cases/matrix.py`.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Part 1

The reference file

Sources under `src/pyflightstream/`: `workspace/inputs.py`; `cases/__init__.py`; `cases/workflows.py`; `cases/matrix.py`.

Where the reference file lives

- A matrix row's `REF` cell names it: `REF r006` selects `inputs/references/r006.toml`. The general filename is `r<id>.toml`.
- It changes *per configuration*, not per flight condition or per study: one aircraft shape, one reference file, cited by every row that flies that shape at whatever condition.
- It does two jobs. It says what the coefficients are divided by (`SREF`, `CREF`, `BREF`, the moment point), and it says what the parts of that configuration are CALLED (the aliases, the rotor blocks, the actuator discs, the custom frames).
- Write it once per configuration, before the rows: everything that follows cites these names and these lengths, and nothing else.

Takeaway *A reference file is the study's vocabulary and its measuring stick, kept apart from the setup (how the solver runs) and the pproc (what gets written down), which change with the study instead.*

Sources under `src/pyflightstream/`: `workspace/inputs.py`; `cases/__init__.py`, the reference model classes.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

How a row names its reference

- `resolve_matrix` narrows the workspace to a resolved matrix: each case in it carries the codes of the row it came from, and the full reference artifact survives there, keyed by `REF`.
- A recipe (a function you write, called with the case and the script) does *not* receive the reference directly: `case.reference.rotor` does not exist.
- A recipe that wants a rotor's block, or its `rpm_sign`, reads it from the resolved matrix the recipe closes over, keyed by the case's own reference code.

Why the split

A recipe is written once and run against whatever reference a row cites. Handing it the artifact directly would let the same recipe silently answer differently for two configurations; reading it back by the row's own code keeps the recipe explicit about which configuration it is asking about.

Sources under `src/pyflightstream/`: `workspace/matrix.py`, `resolve_matrix`; `cases/workflows.py`.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Part 2

Reference lengths and the moment point

Sources under `src/pyflightstream/`: `workspace/inputs.py`; `cases/__init__.py`; `cases/workflows.py`; `cases/matrix.py`.

SREF, CREF, BREF: what the coefficients divide by

Key	Command	Type and meaning
`area_m2`	`SOLVER_SET_REF_AREA`	Float > 0, m^2; SREF, the reference area.
`chord_m`	`SOLVER_SET_REF_LENGTH`	Float > 0, m; CREF, the reference chord.
`span_m`	(recorded)	Float > 0, m; BREF, the reference span.

- All three are required and must be positive. They live in the reference file rather than the row because they answer for a whole configuration, not for one flight condition.
- Change one of these and every coefficient in your results changes: `CL`, `CD`, `CM` and the rest are all divided by them. That is why a study writes them once, cited by every row.

Sources under `src/pyflightstream/`: `cases/__init__.py`, the reference model; `workspace/inputs.py`.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

The fourth length: `rotor_diameter_m`

- Optional top-level field, in metres. A divisor of published numbers exactly like the area and the chord: it is what an advance ratio (`ADVANCE_RATIO`) is a ratio against, and what the rotor coefficients normalise on.
- A row stating `ADVANCE_RATIO` against a reference without this field is refused, naming the field to add.
- It answers for a row that names no rotor alias: one number for the whole configuration. A rotor block (`kind = "rotor"`) carries its own `diameter_m` instead, which wins for a motion citing that block, so rotors of different sizes take different speeds from one written ratio.
- It is optional because a configuration with no rotor has no diameter, and a placeholder value would be worse than nothing.

Sources under `src/pyflightstream/`: `cases/__init__.py`, the reference model; `cases/workflows.py`.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

The moment point and the MRP frame

- [moment_point] states x_m, y_m, z_m: where moments are taken about, in the simulation geometry frame, in metres.
- The package creates a frame named MRP at the moment point on every run type, before any custom frame and before any motion.
- A pproc entry cites it by name: frame = "MRP" is the default for a surface-section distribution, a probe plane and an unsteady force-plot group, so leaving frame unstated already means the moment point.
- A rotor run type also creates <ALIAS>_SMRP at each rotor's own hub, beside MRP rather than instead of it.

Evidence

A name the package creates itself, such as MRP, cannot also be the name of a custom frame the reference declares: the file is refused at plan time, naming the collision.

Sources under src/pyflightstream/: workspace/inputs.py; cases/workflows.py.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Part 3

The study's own words

Sources under src/pyflightstream/: workspace/inputs.py; cases/__init__.py; cases/workflows.py; cases/matrix.py.

The aliases table

An alias is a name YOUR study gives to a set of boundaries, read everywhere a boundary is cited: a `MOVING_BC_ALIAS`, a `ROTATE` record, a `[groups]` member of the pproc, a `families` entry.

```
[aliases]
airframe = ["Body", "Base"]
rotors = ["PORT", "STARBOARD"]
```

- A member may be another alias, followed to the end: `rotors` above names two rotor blocks, and a row can write `rotors` to reach both.
- A member the opened mesh does not carry is **left out** rather than refused. This is what lets one reference file serve a wing-body mesh and an isolated-rotor mesh alike.
- Declare the set here once; every row, every group and every plot then says `airframe` instead of listing the boundaries again.

Sources under `src/pyflightstream/`: `cases/__init__.py`, the reference model; `cases/workflows.py`.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Custom coordinate systems: [[frames]]

One entry per frame, in the order written.

```
[[frames]]
name = "NACELLE"           # the name a row's ROTATE axis cites
origin = [4.5, 0.0, 0.0]   # in the geometry's own frame, metres
x_axis = [1.0, 0.0, 0.0]   # optional; default shown
y_axis = [0.0, 1.0, 0.0]   # optional; the third axis is their
                            # right-handed cross product
```

- Created right after the frames the package makes itself (`MRP`, then `<ALIAS>_SMRP` on a rotor run type), in the order written, and before any motion: a rotor whose axis is one of these turns about a frame that already exists.
- A row's rotation or translation cites a frame axis as `<frame>-X`, `<frame>-Y` or `<frame>-Z`.
- Refused at plan time: a name the package already creates, a name defined twice, or an origin that is not three numbers.

Sources under `src/pyflightstream/`: `cases/__init__.py`, the reference model; `cases/workflows.py`.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Part 4

One block per rotor

Sources under src/pyflightstream/: workspace/inputs.py; cases/__init__.py; cases/workflows.py; cases/matrix.py.

One block per rotor: `kind = "rotor"`

The block's NAME is an alias over everything that rotor owns.

```
[PORT]
kind = "rotor"
x_m = 4.5               # the hub, in the geometry's own frame
y_m = 2.5
z_m = 0.0
axis = "X"               # what it turns about
rpm_sign = 1              # which way
diameter_m = 3.6576       # what an advance ratio resolves against
families_general = []               # hub, spinner: anything not a blade
families_blades = ["Blade1"]        # the blades
blade1 = { azimuth_deg = 0.0, zero = "Y" }   # where blade 1 sits
```

- After this a row says `PORT` (`MOVING_BC_ALIAS` or a `MOTIONS` record) and states nothing else about that rotor: the hub, the axis, the sign, the families and the diameter are all here, once.

Sources under `src/pyflightstream/`: `cases/__init__.py`, `RotorBlock`; `cases/workflows.py`.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

rpm_sign: the hand lives in the reference

- rpm_sign is 1 or -1: which way the rotor turns. +1 is the right-hand rule about the block's axis.
- A row's own speed, whether stated as RPM or derived from ADVANCE_RATIO, is always a **magnitude**: it says how fast, never which way.
- A row that restates RPM_SIGN beside a rotor the reference declares is refused, whether or not the two agree: a row that agrees today says nothing when the reference is corrected tomorrow, so the refusal is about where the key goes rather than about its value.
- One exception: a row whose reference declares no rotor block at all, using the flat ROTOR_AXIS and MOVING_BOUNDARIES keys, has nowhere else to put the hand, so RPM_SIGN beside the speed is correct there.

Limitation

The hand is declared once, beside the axis, the hub and the blade count it already declares there; the row and the reference can never disagree because only one of them is allowed to say it.

Sources under src/pyflightstream/: cases/workflows.py, _refuse_a_row_restating_the_hand, _rpm_sign; cases/matrix.py.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

The blade count, and where `blade1` sits

- `families_general`: what turns with the rotor and is *not* a blade, such as the hub or a spinner.
- `families_blades`: the blades. **The blade count is the length of this list and nothing else**, so a row states no count of its own, and a sector mesh carrying one blade of four still reduces over four.
- `blade1`: `azimuth_deg` (where blade 1 sits, measured from `zero`) and `zero` (the axis that angle is measured from). It locates the first family of `families_blades` so a per-blade frame can be placed; it does not say which family is the first blade, only where that one sits.
- A blade family the opened geometry does not carry is skipped: it gets no blade frame, and the family the geometry does carry keeps its own. A rotor block listing no `families_blades` at all when a row needs the count is refused naming the block and what to add.

Sources under `src/pyflightstream/`: `cases/__init__.py`, `RotorBlock`; `cases/workflows.py`.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

What else reads the rotor block

The tip and helical Mach numbers

R is half the block's `diameter_m`, else half the reference's `rotor_diameter_m`; a disc's is its `tip_radius_m`. With the point's speed V and speed of sound a: $M_{\mathrm{tip}} = \Omega R/a$ and $M_{\mathrm{hel}} = \sqrt{V^2 + (\Omega R)^2}/a$ [3, 7]. A rotor with no diameter anywhere gets [REF-35] no Mach numbers, and the plan and the run record say so, naming the row: nothing is guessed.

A `qsteady_rotor` row

solves the rotor steady, blades still and the free stream turning about the block's `axis` at the row's speed. It is valid only for an **isolated, axisymmetric** rotor. The sector case (periodic symmetry) meshes one blade; the wheel meshes all of them, and `families_blades` still gives the count N that spaces the row's `PASSAGE_POSITIONS` clockings inside one passage, $360°/N$.

Sources under `src/pyflightstream/`: `cases/workflows.py`; the page "Post-processing definitions", tip and helical Mach numbers.

Part 5

Actuator discs

Sources under src/pyflightstream/: workspace/inputs.py; cases/__init__.py; cases/workflows.py; cases/matrix.py.

One actuator disc: `kind = "actuator"`

A row may load a disc the reference declares: the solver's linearized propeller slipstream.

```
[PROP]
kind = "actuator"
frame = "HUB"           # a declared frame, MRP, or a rotor's own frame
axis = "X"               # X, Y or Z of that frame
offset_m = 0.0            # along the axis, from the frame's origin
tip_radius_m = 0.5
hub_radius_m = 0.1        # 0 <= hub < tip
rpm_sign = 1               # the right-hand rule about axis
blades = 3                  # needed by a row loading a profile
profile_units = "NEWTONS"    # unit the profile file is written in
```

- The block's NAME is what a row's `ACTUATOR` key cites, one disc per row.
- Refused: a block name shared with a rotor, an alias or a frame, and a block that forgets `kind = "actuator"`.

Sources under `src/pyflightstream/`: `cases/__init__.py`, the actuator model; `cases/workflows.py`.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

One row, one disc: geometry here, loading on the row

The disc's GEOMETRY belongs to the configuration, so it lives in the reference beside the rotors and the frames; its LOADING belongs to the condition, so the row states it.

```
ACTUATOR: PROP / ACTUATOR_RPM: 2400 / ACTUATOR_THRUST: 120
ACTUATOR: PROP / ACTUATOR_RPM: 2400 / PROFILE: prop_ct
```

- `ACTUATOR_RPM` is a magnitude, rev/min; its hand is the block's `rpm_sign`, exactly as a rotor's speed is.
- A row states exactly one loading: `ACTUATOR_THRUST` (net thrust, the elliptical model) or `PROFILE` (the stem of a file under `inputs/profiles/`, the custom model). Stating both, or neither, is refused.
- A reference declaring a disc changes nothing on a row that names no `ACTUATOR`: the disc simply is not loaded on that row.

Sources under `src/pyflightstream/`: `cases/workflows.py`; `workspace/matrix.py`.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Part 6

Reading the reference back

Sources under src/pyflightstream/: workspace/inputs.py; cases/__init__.py; cases/workflows.py; cases/matrix.py.

What post reads from the reference

A pproc `[groups]` entry names a product file and what is summed into it, and every value is one word the reference declares:

```
[groups]
PUSHER   = "PUSHER"      # a rotor of the reference: that rotor's own
                           # families
AIRFRAME = "airframe"    # an alias of the reference's [aliases] table
TOTAL    = "all"          # every family the geometry carries
```

- The alias resolves through the reference **as it stands when** `pyfs-matrix post` **runs**, not as it stood when the row was planned: renaming or extending an alias needs a re-run of post, never of the solver.
- A group whose alias selects no surface of any loads export writes no table: it is named under `skipped` in `products.json` with the surfaces the export does carry.
- Family resolution for surface-section integration reads the same live reference, including rotor names and their aliases, matched against the recorded boundary families [3].

Sources under `src/pyflightstream/`: `workspace/matrix.py`; also repository `docs/post-processing-definitions.md`.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Common refusals

A row or file states	The refusal says
`RPM_SIGN` beside a rotor block	Remove it from the row; set `rpm_sign` on that rotor's block instead, naming which of the two is true today.
`ADVANCE_RATIO`, reference has no diameter	Names the missing field: the top-level `rotor_diameter_m`, or the cited block's `diameter_m`.
A rotor block with no `families_blades`	Names the block; add the families to the rotor reference block; do not restate its blade count on the row.
An actuator block missing `kind = "actuator"`	Names the block and that the key is missing.
A row loading an actuator: both loadings, or neither	Names the row and that exactly one of `ACTUATOR_THRUST` or `PROFILE` is wanted.
Two blocks (rotor, alias, frame, disc) sharing one name	Names the collision and both blocks.

Sources under `src/pyflightstream/`: `cases/workflows.py`; `cases/matrix.py`; `workspace/inputs.py`.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Part 7

A complete reference file

Sources under src/pyflightstream/: workspace/inputs.py; cases/__init__.py; cases/workflows.py; cases/matrix.py.

Complete reference example: lengths and vocabulary

`inputs/references/r101.toml`, selected by `REF = r101`. A two-rotor configuration, continued next frame.

```
area_m2 = 10.507   # SREF, m^2
chord_m = 4.0       # CREF, m
span_m = 3.6576      # BREF, m

[moment_point]
x_m = 2.0
y_m = 0.0
z_m = 0.0

[aliases]
airframe = ["Body", "Base"]
rotors = ["PORT", "STARBOARD"]

[[frames]]
name = "NACELLE"
origin = [4.5, 0.0, 0.0]
```

Sources under `src/pyflightstream/`: `cases/__init__.py`; `workspace/inputs.py`; also repository `docs/workspace-and-workflows.md`.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Complete reference example: the port rotor

Continue the same file. Two rotors follow, one block each.

```
[PORT]
kind = "rotor"
x_m = 4.5
y_m = 2.5
z_m = 0.0
axis = "X"
rpm_sign = 1
diameter_m = 3.6576
families_general = []
families_blades = ["Blade1"]
blade1 = { azimuth_deg = 0.0, zero = "Y" }
```

Sources under `src/pyflightstream/`: `cases/__init__.py`; `cases/workflows.py`; also repository `docs/workspace-and-workflows.md`.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

Complete reference example: the starboard rotor

Continue the same file. The two diameters differ on purpose: one advance ratio written once in a row's flight condition resolves against each rotor's own diameter, so unlike rotors turn at unlike speeds from one written number.

```
[STARBOARD]
kind = "rotor"
x_m = 4.5
y_m = -2.5
z_m = 0.0
axis = "X"
rpm_sign = -1         # counter-rotating
diameter_m = 1.8288
families_general = []
families_blades = ["Blade2"]
blade1 = { azimuth_deg = 0.0, zero = "Y" }
```

Sources under src/pyflightstream/: cases/__init__.py; cases/workflows.py; also repository docs/workspace-and-workflows.md.

Copyright (c) 2026 Geovana Neves, CC BY 4.0

The physics behind the file, and where to read it

- **Coefficients** divide a force by the dynamic pressure and the reference area, and a moment by the area and a reference length; the moment is taken about the moment point [4]. [REF-43]
- **The body axes** of the moment point and the rotor frames follow the right-hand rule of flight mechanics [4].
- **The advance ratio** $J = V/(nD)$ and the propeller coefficients are the classical airscrew quantities [5, 6]. [REF-36], [REF-42]
- **The actuator disc** is the momentum theory of a propeller: a pressure jump and a swirl across a disc of the rotor's radius [5].
- **The tip and helical Mach numbers** measure how far the blade tip is from the sonic limit of a linearised panel method [3, 7]. [REF-35]

Copyright (c) 2026 Geovana Neves, CC BY 4.0

References

[1] Geovana Neves, *pyflightstream documentation*, page "The workspace and the workflow". `https://nevesgeovana.github.io/pyflightstream/latest/workspace-and-workflows/`

[2] Geovana Neves, *pyflightstream documentation*, page "Getting started". `https://nevesgeovana.github.io/pyflightstream/latest/getting-started/`

[3] Geovana Neves, *pyflightstream documentation*, page "Post-processing definitions". `https://nevesgeovana.github.io/pyflightstream/latest/post-processing-definitions/`

REF-43 B. Etkin, L. D. Reid, *Dynamics of Flight: Stability and Control*, 3rd ed., Wiley, New York, 1996.

REF-42 H. Glauert, *The Elements of Aerofoil and Airscrew Theory*, 2nd ed., Cambridge University Press, Cambridge, 1947.

REF-36 B. W. McCormick, *Aerodynamics, Aeronautics, and Flight Mechanics*, 2nd ed., Wiley, New York, 1995.

REF-35 J. G. Leishman, *Principles of Helicopter Aerodynamics*, 2nd ed., Cambridge University Press, Cambridge, 2006.

Copyright (c) 2026 Geovana Neves, CC BY 4.0