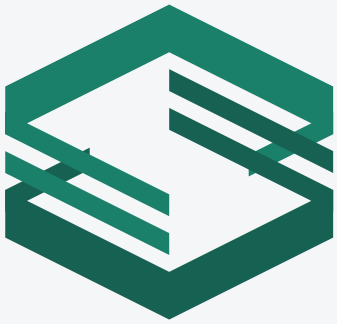
Manual 03 · from project to signed artifact

Build & Publish

How a project becomes an artifact the runner will accept — choosing what to freeze and what to leave configurable, managing your signing key, reading the manifest, and handing the artifact over to the people who run it.



CONTENTS

01 From project to artifact

02 Before you compile

03 Your signing key

- 3.1 Create it
- 3.2 Unlock, lock
- 3.3 Share the public half
- 3.4 Import an existing key
- 3.5 Rotate the key

04 The Compile panel

- 4.1 Log detail
- 4.2 Generate

05 Freeze or configure

06 Inside an artifact

- 6.1 Check before you ship

07 Handing over an artifact

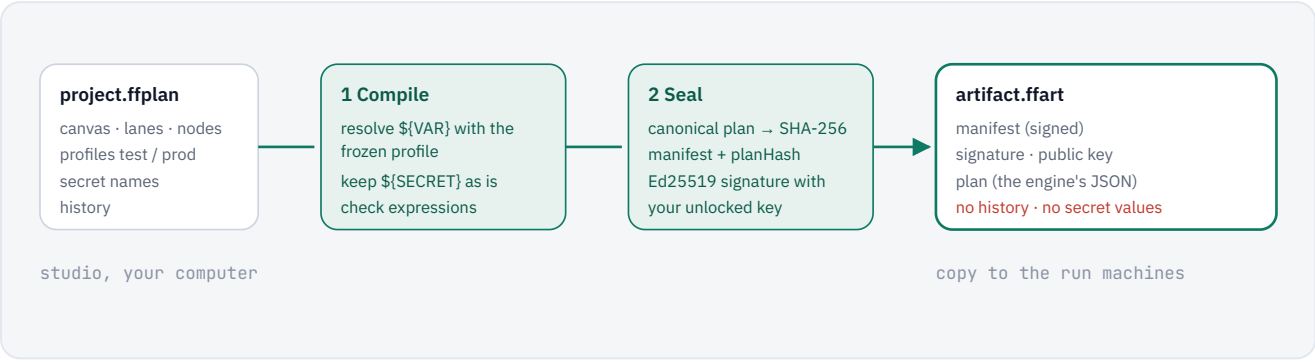
08 Versions and compatibility

09 Troubleshooting compilation

Copyright © 2025–2026 Marco Dongu. All rights reserved.

01 From project to artifact

A **project** (`.ffplan`) is what you design: canvas, lanes, profiles, history. An **artifact** (`.ffart`) is what you deploy: the plan compiled for the engine, one environment frozen into it, and a manifest signed with your key.



	Project `.ffplan`	Artifact `.ffart`
Who uses it	developers, in the studio	the runner, on the run machines
Environments	all profiles	**one** profile, frozen
Secrets	names only	names only (listed as *required secrets*)
History	yes, last 20 versions	no
Can be edited	yes	no — any change breaks the signature
Executed by	the studio (**Run**)	the runner, after verifying the signature

02 Before you compile

A short checklist saves a round trip with the people who run the artifact.

01 **Validation is clean.** The *Validation* tab shows no errors. Warnings are allowed but read them.

02 **The flow runs in the studio** with the profile you are going to freeze. Switch *Environments* → *Active profile* to that profile and press **Run** at least once on representative data.

03 **Nothing literal that belongs to an environment.** Hosts, paths, thresholds and dates are `${VARIABLES}`; passwords and tokens are `${SECRETS}`. Search the settings for host names and passwords you may have typed while testing.

04 **Every node has a clear name.** Logs and the monitor show names, not ids.

05 **Errors go somewhere.** Each lane's Error Handler sends `error_out` to a file, a table or a mail; critical nodes are marked.

06 **The project is saved.** The artifact records the project's name and version.

03 Your signing key

The runner executes only artifacts signed by a key it trusts. Your key is created once per computer and identifies you as the author of what you compile.

Create it

01 Press **Signing**. The panel says *No signing key on this computer*.

02 Under **Create the signing key**, type a passphrase and press **Generate from system**. The key is generated from the operating system's random source.

03 Keep the passphrase somewhere safe — a password manager. *If you forget it the key is unrecoverable.*

The panel then shows the key's fingerprint, its **keyId**, for example `ed25519:9f3c51a0…`. The same keyId appears in every artifact you sign and in the runners' trust stores.

File	Content	Protection
`~/.stilton/signing-key.json`	keyId and public key in clear; the private key **encrypted**	passphrase → Argon2id → XChaCha20-Poly1305; file readable only by you

Unlock, lock

The private key is decrypted only when you need it. **Unlock for this session** asks for the passphrase and keeps the key in memory until you press **Lock now** or close the studio; then it is wiped from memory. If the key is locked when you press **Generate**, nothing is written and the log says *signing key locked*: unlock it and generate again — compilation never produces an unsigned artifact.

Share the public half

Under **Export the public key**, **Copy trust store entry** copies a small JSON document like this one:

entry.json — json

```
{
  "keyId": "ed25519:9f3c51a0e1b7c4d2…",
  "publicKey": "q3V0b1N0aWx0b24tZXhhbXBsZS1wdWJsaWMta2V5LTMyYg==",
  "owner": "",
  "status": "active"
}
```

Send it to whoever administers the run machines. They authorise it with `stilton-runner trust add entry.json --owner "Mark Rossi"` (see *Runtime Operations*). The entry contains no secret: it can travel by e-mail or chat. What matters is that the administrator **checks the keyId with you on a second channel** — a call, in person — before trusting it.

Warning. Never send the file `signing-key.json`, and never type your passphrase anywhere but in the studio. Whoever has both can sign artifacts in your name.

Import an existing key

To use the same identity on a second computer, **import** the key's seed (*Import and encrypt*, a base64 value of 32 bytes) and choose a passphrase for that computer. Prefer one key per computer instead: if a laptop is lost, you revoke only its key.

Rotate the key

Rotate when a computer is lost or replaced, when a passphrase may have been seen, or simply on a schedule.

01 In **Signing**, under *new identity*, choose **Generate new** and a new passphrase.

02 Send the new trust store entry to the administrators; they add it to every runner.

03 Recompile and redeploy the artifacts in use.

04 Ask the administrators to revoke the old keyId: `stilton-runner trust revoke ed25519:<old>`.

Until step 4, artifacts signed with either key are accepted. After it, artifacts signed with the old key are refused at the next run.

04 The Compile panel

Press **Compile**. The panel has five settings and shows the manifest as it will be signed.

Setting	What it decides
Profile to freeze	which environment's values are written into the artifact. *(none — default values)* uses the defaults of the shared variables
Compiled by	the label of this studio installation, shown by the monitor — for example `mark-laptop`. It is a label, not authentication
Monitor	where the runner sends the run events: a literal URL, `${MONITOR_URL}` to let each run machine decide, or empty for no monitor
Target platform	Linux or Windows; recorded in the manifest and used in the default file name. The plan is the same for both
Detail of the sent log	how much the runner sends to the monitor (below)

Log detail

Level	Sent to the monitor	Use it for
Essential	lifecycle of runs and nodes, errors and warnings, statistics	production
Normal (default)	+ progress and all node messages	most cases
Diagnostic	everything, including the **content of rows** and memory samples	troubleshooting on test data only

The level filters only what **leaves** the run machine. The complete log of every run is always written locally, in `~/.stilton/runs/` of the account that runs it.

> **Warning.** *Diagnostic* sends row content — names, e-mails, amounts — to the monitor and stores it there. Do not use it on personal or confidential data.

Generate

Press **Generate** → and choose where to save. The default name is `artifact-<platform>.ffart`; give it a name that says what it is and for which environment, such as `orders-daily-prod.ffart`. The log confirms the profile and lists the secrets the artifact requires.

05 Freeze or configure

By default every shared variable is **frozen**: its value comes from the profile you chose and is signed into the artifact. Change it, and you must recompile.

Some values are legitimately decided by the people who run the artifact — a threshold, a host name inside a known domain, a mode. Mark those variables **configurable at run time**:

01 In **Environments**, press ⚙ next to the shared variable.

02 Choose its type — text, number, true/false, JSON — and its constraints.

03 The value of the active profile becomes the default.

Constraint	Applies to	Example
Allowed values	text, number	`basic`, `full`
Minimum / maximum	number	`0` … `1000`
Format	text	`[a-z0-9.-]+\.internal` — the whole value must match

The constraints are signed into the manifest. On the run machine, an administrator can then change those values — and only those — with a configuration file:

prod.json — json

```
{ "profile": "prod", "values": { "THRESHOLD": "250", "DB_HOST": "db2.internal" } }
```

Linux — bash

```
stilton-runner --config prod.json orders-daily.ffart
```

The runner refuses the artifact (exit code 3) if the file sets a variable that is not configurable, or a value outside its constraints: `DB_HOST = db.example.com` would not match the format above. The run log records which names were changed and a fingerprint of the values, never the values themselves.

Make it configurable when…	Keep it frozen when…
operations must tune it without a developer (thresholds, batch sizes)	changing it changes what the flow *means* (a filter condition, a table name)
it differs per site within the same environment (host of the local database)	it is the same everywhere
the set of good values is small and known	you cannot express what a good value is

Note. An artifact with configurable variables requires a runner 0.2.0 or later. Secrets are never configurable variables: they stay environment variables or keychain entries on the run machine. A lane variable with the same name as a configurable variable is a compile error.

06 Inside an artifact

An artifact is a JSON file. You never edit it, but reading it answers most questions about what is about to run.

orders-daily-prod.ffart (shortened) json


```
{
  "formatVersion": 2,
  "kind": "flowpilot-artifact",
  "manifest": {
    "planName": "orders-daily",
    "planVersion": { "id": "…", "savedAt": "2026-11-06T17:42:10Z", "label": "after API v2" },
    "studio": { "id": "4b0e…", "label": "mark-laptop" },
    "studioVersion": "0.9.0",
    "profile": "prod",
    "platform": "linux",
    "logLevel": "normale",
    "monitor": "${MONITOR_URL}",
    "requiredSecrets": ["DB_PASSWORD", "API_TOKEN"],
    "engineVersionRange": "⩾0.9.0",
    "createdAt": "2026-11-07T09:15:02Z",
    "planHash": "sha256:5e1f…",
    "hashAlg": "sha256",
    "sigAlg": "ed25519",
    "keyId": "ed25519:9f3c51a0…"
  },
  "sig": "MEUCIQ…",
  "publicKey": "q3V0b1N0…",
  "plan": { "lanes": [ … ] }
}
```


Field	Meaning
`planHash`	SHA-256 of the plan in canonical form. The runner recomputes it: if one character of the plan changed, the artifact is refused
`keyId`	which key signed it; it must be active in the runner's trust store
`requiredSecrets`	the names the run machine must provide
`variables`	present only if some variables are configurable: their type, default and constraints
`monitor`, `logLevel`	where events go and how many; signed, so they cannot be changed in transit
`engineVersionRange`	which runner versions may run it: at least the studio's own *major.minor* (Studio 0.9.x writes `⩾0.9.0`)

Everything operational is **inside** the manifest, so everything is covered by the signature. The publicKey next to it is a convenience: the runner trusts only the public key in its own trust store, never the one inside the artifact.

Check before you ship

On a machine with the runner and your key authorised, ask whether an artifact would be accepted without running it:

Linux bash

```
stilton-runner --admit-only orders-daily-prod.ffart
```

output json

```
{"admitted":true,"keyId":"ed25519:9f3c…","owner":"mark","planHash":"sha256:5e1f…",
 "planName":"orders-daily","nodeTypes":["filter","sink_file","source_db","tmap"]}
```

`nodeTypes` lists every kind of node the artifact uses — exactly what a runtime profile (see *Runtime Operations*) will check.

---

07 Handing over an artifact

Give the people who run it a small, complete package:

Item	Why
the `.ffart` file	the job itself
the list of required secrets and what each one is	they must exist on the run machine
the configurable variables, with their meaning	so operations know what they may change
the trust store entry of your key, if the machine does not know it yet	first deployment only
how often it runs and what "success" looks like	rows expected, files produced, how long it takes

A handover note can be this short:

HANDOVER.txt text

```
orders-daily-prod.ffart   (Stilton 0.9.0, signed by mark-laptop, ed25519:9f3c51a0…)
Runs:          every day at 06:00, ~3 minutes, ~120 000 rows
Secrets:       DB_PASSWORD  read-only user etl_reader on db.internal
               API_TOKEN    token of the orders API, scope orders:read
Configurable:  THRESHOLD    minimum amount to export (0–1000, default 100)
Monitor:       set MONITOR_URL and MONITOR_TOKEN on the machine
Output:        /data/out/totals_<date>.csv  and  /data/out/rejected.csv
Success:       exit code 0; rejected.csv usually under 50 rows
```

---

08 Versions and compatibility

- The artifact records the project version it was compiled from (`planVersion`) and the studio version (`studioVersion`). Keep the `.ffplan` of every artifact you deploy — in Git, or at least as a named checkpoint in **History** — so you can always answer *"what exactly is running?"*.

- **Recompile** whenever you change the flow, a frozen value, the monitor or the log level. A runner never "updates" an artifact.
- Artifacts compiled by a newer studio may need a newer runner; the runner says so at admission (`engineVersionRange`) instead of failing halfway: *"the artifact needs a runner '>=0.10.0', this runner is '0.9.0' — update the runner, or recompile..."*.
- The other way round, a runner states from which studio version on it can read artifacts. If an old artifact is ever refused for that reason, the message says so: open its project in the current studio and generate the artifact again.
- **Projects always open in later versions** (see *Projects from other versions of Stilton* in the Studio Guide): that is why the `.ffplan` of every deployed artifact is worth keeping.

---

09 Troubleshooting compilation

Message	What to do
Signing failed: signing key locked	press **Signing → Unlock for this session**, then generate again
Signing failed **... no key**	create the signing key (above)
configurable variables: **default outside the constraints**	the active profile's value breaks a constraint you set with ⚙: change the value or the constraint
User function examples that are not true	an example written in a user function no longer holds: fix the function or the example (*FPEL Language Reference*)
Flow compilation failed	read the message: it names the node; the *Validation* tab usually shows the same problem
Export is available only in the desktop app	you are in a browser preview of the studio: compile from the installed application