Page contents

- A save has more than one start
- Persist data through <code>storage</code>
- Register for the event you mean
- Upgrade an existing save safely
- Reference coverage still missing
- Sources and provenance

Offline copy of Runtime lifecycle, storage and migrations .

Factorio: runtime lifecycle, storage and migrations

Documentation status: In progress. **Source checked:** 2026-10-06 against Wube's 2.1.20 runtime and auxiliary documentation. **In-game validation:** not recorded. The greeting quick start shows one `on_player_created` event; this chapter explains how that event fits a save's lifetime.

A save has more than one start

`control.lua` runs when a new or existing save starts. It registers handlers and interfaces for that mod's own runtime Lua state. At this first load step, `game` is unavailable and `storage` has not yet been restored; do not alter a player or initialize persistent state in the top level of `control.lua`. The official data lifecycle separates later setup points:

Hook or phase	Runs	Appropriate work
`script.on_init`	When the mod is new to a save	Create initial `storage` values; inspect or modify game state.
Migration files	Once per migration filename per save	Transform renamed prototypes or saved data deliberately.

Hook or phase	Runs	Appropriate work
`script.on_load`	When the mod was already in that save	Rebuild local references or conditional registrations; **do not write `storage` or use `game`**.
`script.on_configuration_changed`	When the save's mod/game configuration changes	Adapt stored data and game state after version, setting or prototype changes.
Ordinary event handler	After setup, when its event occurs	React to a player action or game state.

The `on_init` and `on_load` paths are mutually exclusive for one mod at one load. A mod newly added to an existing save gets `on_init`, even though the map itself is old. In multiplayer, a joining client does not rerun server-side migrations; its event registrations must still match the server's. The data lifecycle is the source for these ordering and synchronization rules.

Persist data through `storage`

The 2.1 API provides each mod its own `storage` table, serialized into the save. Ordinary Lua globals can cache data for the current session but do not persist across save/load. Factorio before 2.0 called this table `global`, so old examples using `global` need a version branch before copying into a 2.1 mod. Keep persistent values small and meaningful: IDs, counters, configuration state, or supported LuaObject references. Functions cannot be stored, and unregistered metatables are not preserved as metatables.

Initialize fields in `on_init`, read or update them from events, and evolve their shape in `on_configuration_changed` or a migration. `on_load` may read storage and set local references, but a write there is prohibited because it can desynchronize multiplayer and replays. The storage reference lists serializable values and metatable behavior.

Register for the event you mean

`script.on_event(defines.events.on_player_created, handler)` runs when a **player is created**, not every time an existing save loads. This explains why the quick-start greeting appears in a new-map test and not on every reload. Each event's reference entry lists its arguments, such as `player_index`, and when it fires. Use events to select a specific trigger, then runtime classes to inspect the objects returned by `game`, surfaces and players. A prototype field reference is not a runtime object method reference.

Avoid registering an expensive every-tick handler for a rare change when a specific event exists. Work that depends on deterministic multiplayer behavior should be tested with a hosted save and a joining client, including save/reload. Do not assume a callback ran because the mod loaded; log a single event payload or visible effect in a disposable save.

Upgrade an existing save safely

A mod version change, game version change, added/removed mod, changed startup setting, changed prototype set, or applied migration can trigger `on_configuration_changed`. It is a place to reconcile persistent data with a new schema, but it is not a substitute for every migration. The migrations guide distinguishes JSON migrations, used for prototype rename/type mappings, from Lua migrations that adjust game state. JSON files run before Lua files, and a save remembers migration filenames it has already applied. Renaming or removing an entity may invalidate references or ghosts, so inspect the consequences rather than assuming all saved objects survive.

Use this upgrade check: copy a save made with the old mod version; install only the new packaged version; load it once; inspect the log and changed entities/state; save and reload; then repeat in multiplayer if supported. Keep migration filenames stable once released. Preserve both old and new ZIPs and the test save so a compatibility claim can be reproduced.

Reference coverage still missing

The runtime reference contains event arguments, classes, commands, remote interfaces, GUI, rendering, surfaces, logistics, forces and many more systems. This chapter gives the lifecycle and persistence contract, not all signatures. Full runtime API

pages and tested migration examples remain open work. Factorio's documentation status remains **In progress**.

Sources and provenance

- Wube Data lifecycle, Storage, Migrations, Events and runtime class index — official 2.1.20 API; checked 2026-10-06.
- Factorio API documentation license — Wube's documentation license permits redistribution with its notice. This page is original ModDocs prose and bundles no game files or copied API reference.