Guide contents

Pick the right stage

Make the smallest loadable mod

Add content in data.lua

Add behavior in control.lua

Settings, translation, and compatibility

Test and publish

A complete first project: greet new players

Sources / References

Documentation: In progress **Quick start: Complete**

Offline copy from ModDocs. Source links remain clickable.

Factorio modding

Factorio mods are small packages of Lua and assets. You can begin with a definition-only mod, then add runtime behavior when you need it. This page is a practical route through the official mod structure, data lifecycle, API reference, and the wiki's tutorial index.

> **Quick start:** decide whether you are adding a prototype, changing live gameplay, or both. Create a folder with `info.json`; add `data.lua` for prototypes or `control.lua` for runtime events. Start with one visible change, load the game, and read any startup error before adding more.

Pick the right stage

Goal	Start here	What it does
Add or change an item, recipe, entity, or technology	`data.lua`	Defines **prototypes** while the game starts.

Goal	Start here	What it does
Adjust another mod's prototypes	`data-updates.lua` or `data-final-fixes.lua`	Runs later in the prototype lifecycle. Declare dependencies and follow the documented load order.
React to events during a save	`control.lua`	Runs with the runtime API during gameplay.
Offer user options	`settings.lua`	Defines startup, map, or per-player settings.
Show readable names and descriptions	`locale/<language>/*.cfg`	Supplies localized strings.

The distinction matters: a prototype is a template loaded at startup; a runtime object is something the active map contains. The APIs and available data differ. See Data lifecycle before moving code between stages.

Make the smallest loadable mod

Factorio recognizes a mod folder or zip. A folder can use the mod's name or `name_version`; a zip must use `name_version.zip`. The mod name and version are declared in the required `info.json`. A simple working tree is:

```text
my-first-mod/
  info.json
  data.lua
  locale/
    en/
      names.cfg
```

`info.json` identifies the package; the other files are included only if you need them. The official mod-structure page lists optional files and exact packaging rules. Match its current schema when writing the manifest instead of copying an old mod's metadata unchanged.

Put the folder in the game's mods directory, enable it in the mod manager, and restart when requested. For a first test, add or alter one prototype using an example from the current prototype API and check that the content appears in a new test save. Do not treat a successfully loaded folder as proof that the content was registered; search for the object in game and review the log.

Add content in `data.lua`

Prototype definitions are Lua tables. The data stage reads them through the `data` interface, commonly using `data:extend(...)` to register new entries. Each prototype type has its own required fields, so copying an item example into an entity type will not work. Look up the exact type in the prototype reference and inspect a current base-game example for fields such as icons, recipes, or placement behavior.

Use your own unique prototype names. For changes to existing prototypes, first verify that the target exists and that your mod's dependencies put your change at the correct point in the lifecycle. `data-updates.lua` and `data-final-fixes.lua` are useful only when their later timing is actually needed; they are not general fixes for an incorrect definition.

Add behavior in `control.lua`

Runtime code subscribes to events and works with runtime objects, not raw prototype tables. Choose a narrow event, register a handler, and print or perform one small observable action. Then test a new save and a reloaded save. Runtime state must be handled with the persistence and migration rules in the current runtime API and data lifecycle; ordinary Lua globals are not a substitute for persistent state.

For a mod that adds a new machine, separate the work into: prototype data, locale and graphics, recipe and technology access, then optional runtime behavior. Test each stage before combining them. This keeps an error in the definition stage from being mistaken for an event-handler bug.

Settings, translation, and compatibility

Settings are their own startup stage. Decide whether a value should be changed only on restart, once per map, or per player, and use the appropriate setting type. Put player-visible text in locale files rather than hardcoding it in Lua. If another mod is required, declare that dependency in `info.json`; avoid silently assuming its prototypes exist. Read the current settings and mod-structure docs for the load order and recognized files.

Test and publish

1. Run with just your mod and its declared dependencies. Fix the first startup error before adding more content.
2. Create a fresh save to test prototype changes, then load an existing save to test upgrade behavior.
3. Verify the item or recipe is actually obtainable and localized, rather than merely present in data.
4. Check multiplayer and migration behavior if runtime state matters.
5. Package the exact folder or zip shape described by the official structure guide. Include a version and changelog; test the packaged copy before uploading.

A complete first project: greet new players

This small runtime-only mod is deliberately useful without requiring an icon, prototype, or migration. Make a folder named `hello-engineer` in the mods directory. For the documented **2.1** branch, create this complete `info.json`:

```json
{
  "name": "hello-engineer",
  "version": "1.0.0",
  "title": "Hello Engineer",
  "author": "Your Name",
  "factorio_version": "2.1",
  "description": "Welcomes a player who joins a new map.",
  "dependencies": ["base"]
}
```

Set `factorio_version` to the major game branch you actually use; a 2.1 manifest does not load on 2.0. The official mod-structure reference specifies the version format and required fields. Add a `control.lua` containing:

```lua
script.on_event(defines.events.on_player_created, function(event)
  local player = game.get_player(event.player_index)
  if player then
    player.print("Welcome, engineer!")
  end
end)
```

The event list defines `on_player_created`, and the runtime class reference documents `get_player`. The resulting project is just two files:

```text
hello-engineer/
  info.json
  control.lua
```

Enable the mod, create a **new** test map, and look for the message once when its first player is created. Reload that save: the message should not repeat merely because the map was loaded. Starting another new map should show it again. If the mod loads but nothing appears, check whether you tested with an existing player; `on_player_created` is not a generic startup event. Check the game log if the mod is rejected before a map opens.

To package it, make a ZIP named `hello-engineer_1.0.0.zip` containing the `hello-engineer` folder with those two files. The official mod structure allows a differently named internal folder, but keeping the names aligned makes review easier. Remove the development folder from the active mods directory, install the ZIP, enable the mod and repeat the new-map test. That checks the distributable rather than accidentally loading loose source files.

The next step is a content prototype in `data.lua`. Use a current prototype example for one specific type, then add its localization and a way to obtain it. This runtime example establishes the package and event lifecycle first; it does not pretend one generic item table fits every prototype.

This gives a complete runtime-only mod and paths to prototype, settings and localization work. The current official API remains the authority for individual prototype fields, migration rules, GUI events and multiplayer behavior, which vary by

feature and game version. The example above is source-reviewed against the linked 2.1 documentation; it has not been run in a local Factorio installation here.

Continue with the prototype stage and package chapter or the runtime lifecycle, storage and migrations chapter. They use the official API's 2.1.20 snapshot checked 2026-10-06. Factorio's overall documentation remains **In progress**; the greeting project has not been independently run in game.

Sources / References

External project and documentation links cited in this guide. Check the upstream release and version before following version-sensitive steps.

- mod structure
- data lifecycle
- API reference
- tutorial index
- prototype API
- mod-structure reference
- event list
- runtime class reference