Page contents

- Separate the mod, its entry file, and its objects
- Read an object registration table
- Choose the correct object class
- Move text into localization files when it grows
- Work with an existing object deliberately
- What still needs separate reference pages
- Sources and provenance

Offline copy of Mod identity and game objects .

Balatro with Steamodded: mod identity and game objects

Documentation status: In progress. **Source checked:** 2026-10-06. **ModDocs page updated:** 2026-10-06. This chapter follows the maintained Steamodded documentation and its JSON metadata model. It explains how one mod owns game objects; the first Joker quick start contains the runnable +4 Mult example. No Balatro, Lovely, or Steamodded version pair has been independently tested for these pages. Read the SMODS release notes when adapting code to another release.

Separate the mod, its entry file, and its objects

Steamodded finds a **mod** from metadata in its mod folder, then executes the named Lua entry file. The `id` identifies that mod for dependencies and loading. The `prefix` is used when the mod registers **game objects** such as Jokers or vouchers. Those objects are not separate mods: several objects can come from one entry file or files it loads.

The modern metadata route is a separate JSON file. The older `--- STEAMODDED HEADER` at the start of a Lua file is documented as outdated. Do not use both discovery methods for the same mod. JSON requires real JSON syntax, so examples with `//` comments in the upstream explanatory specification must have those comments removed before saving a working file.

Field	Role	Failure to check
`id`	Unique mod identity	A collision or invalid identity stops a reliable dependency target
`name`, `author`, `description`	Player-facing metadata	Missing required fields prevent the expected display/discovery path
`prefix`	Namespace for registered object keys	Reusing another mod's prefix risks key collisions
`main_file`	Lua entry point	A typo means the code is never executed
`version`	Release identifier	Dependency ranges and support reports become ambiguous without it
`dependencies`, `conflicts`	Loader-level compatibility	A required mod can be absent or an incompatible one can load

The metadata specification defines additional fields such as load `priority`, badges, icons and version constraints. Load priority orders mods from lower to higher values; it is not a substitute for declaring a true dependency. A metadata dependency prevents the whole mod from loading when its requirement is unmet. An object-level `dependencies` field instead controls whether one registered object loads when other mod IDs are present. Use the correct level for the behavior you want.

Read an object registration table

Steamodded's game-object API registers an object by calling its class with a Lua table: `SMODS.Joker { ... }`, `SMODS.Voucher { ... }`, and so on. Every class can define its own required fields and methods; shared fields live in the base game-object documentation. `key` is the local name for this object. Steamodded adds class and mod prefixes where that class uses them, so another part of your mod must refer to the **full registered key** when connecting objects.

`loc_txt` supplies text near the object when a small experiment does not yet have separate translation files. `unlocked` and `discovered` are different: the first controls initial availability, while the second controls initial collection visibility. `no_collection`

hides an object from the collection. These fields affect what a player can see and obtain; they do not prove a scoring or purchase callback worked.

For card-like objects, `config` defines initial data. A card instance receives data in its `ability` table, but the object definition itself is a shared prototype. Put custom starting values under `config.extra` when the API says only selected keys are copied. If a value must change on an individual card during play, do not store that run-specific value on a shared object prototype. The base object reference explains the common fields and `self` versus `card` parameters.

Choose the correct object class

The class matters because it determines where the object appears and which methods the game calls. A Joker scoring effect belongs under the Joker reference. A new voucher, consumable, blind, back/deck, edition, or enhancement uses its own class page and fields. Calling every card-like thing `SMODS.Joker` will not make a Tarot consumable or a deck. Some classes, including Seals and Challenges, sit outside the `SMODS.Center` family; navigate from the game-object index before copying parameters across classes.

The `calculate` function can respond to many **contexts**, which identify when the game is asking an object for an effect. `context.joker_main` is one scoring phase, not a generic all-events callback. The calculate reference documents scoring, round, shop, card and probability contexts. Check the requested context first, then return the effect table supported at that point. An unconditional return can unintentionally apply an effect at more than one stage.

Move text into localization files when it grows

The first Joker example keeps its name and description in `loc_txt` for approachability. For several objects or languages, Steamodded recommends files such as `localization/en-us.lua` or `localization/en-us.json`; `localization/default.lua` can provide fallback text. The localization key includes the object's **full prefixed key**, and its category must match the object's type. A Joker string placed under `Voucher` will not describe a Joker.

When text is missing, inspect the file path, language code, category, and full object key before changing gameplay code. Translation files can be split into subfolders in releases that support that feature, but older Steamodded versions need a version-specific check. The localization reference documents formatting, variables and programmatic functions beyond the simple name/description case.

Work with an existing object deliberately

If a feature needs to change a vanilla or another mod's object, registering a second object with the same visible name is not ownership. The base API offers `take_ownership` to replace specified fields while leaving the others in place. That operation has compatibility implications: another mod may also change the object, and the load order matters. Test the change with only its stated dependencies, then with the other mods it claims to support. Record the full object key, the Steamodded release, and which field you replaced.

What still needs separate reference pages

The upstream project documents many more game-object classes, the mod object and configuration methods, scoring contexts, Lovely patching, atlases, shader/sound/UI features, and distribution. These are tracked in ModDocs' internal Balatro coverage audit and must become on-site reference chapters before this game can be called usable as a primary technical reference. The authoritative SMODS documentation remains necessary for the missing classes and exact signatures.

Sources and provenance

- Steamodded mod metadata — SMODS Team; checked 2026-10-06; JSON schema and version constraints.
- Steamodded game-object API — SMODS Team; checked 2026-10-06; registration, common fields and ownership.
- The Mod Object — SMODS Team; checked 2026-10-06; runtime identity, config and loader state.

- Calculate functions and Localization — SMODS Team; checked 2026-10-06; reference scope and terminology.
- Steamodded source — project code is GPL-3.0; this page is original ModDocs prose rather than a copy of upstream documentation.