Page contents

- Keep IDs, strings and resources aligned
- Save run state and persistent preferences separately
- Debug by boundary
- Package and distribute
- Remaining validation
- Sources and provenance

Offline copy of Localization, saves, debugging and release .

Slay the Spire: localization, saves, debugging and release

Documentation status: In progress. **Source checked:** 2026-10-06. **In-game validation:** not recorded. This chapter covers the remaining production paths for the original game and builds on content families.

Keep IDs, strings and resources aligned

BaseMod's Custom Strings and Hooks pages place string registration in `receiveEditStrings`. Store player-facing names and descriptions in the appropriate JSON family, including `CardStrings`, `RelicStrings`, `PotionStrings`, `EventStrings`, `PowerStrings` or character/UI strings as required by the object. Use the same namespaced ID in code and JSON. Establish one fallback language and test a second language or missing translation; avoid hard-coded English labels in Java where localization is expected.

Package image resources inside the mod JAR and reference their actual internal paths. Cards, relics, powers, potions and characters have different render surfaces and size expectations, so follow each BaseMod object page and inspect both normal and enlarged views. Test a clean JAR installation; a source-tree file visible to the IDE may be absent from the release. BaseMod's `AddAudioSubscriber` and `BaseMod.addAudio`

are listed in its hook table; register original audio there and test playback and missing-file behavior before shipping.

Save run state and persistent preferences separately

BaseMod's Custom Savable reference covers extra state on a card or relic via `CustomSavable<T>`, and describes `BaseMod.addSaveField` for state not owned by those objects. Save a serializable value, then restore it in `onLoad`; the reference example saves a deck index instead of trying to serialize an `AbstractCard`. Validate null, missing, old-version and out-of-range data as well as normal reload. A `counter` or `misc` field may be enough for some objects, but document its meaning and copy behavior.

Mod preferences are a different lifetime. The ModTheSpire SpireConfig reference and BaseMod's Mod Config and Panel describe saved options and settings UI. Choose defaults, read and write preferences, then restart the game to confirm persistence. Do not place run-specific combat state in a settings file or assume a settings option becomes a save-field automatically.

Debug by boundary

The BaseMod console can create content and inspect state without waiting for random rewards. Use a disposable run: the BaseMod README warns that some commands in the starting room can make saves unloadable, and invalid event IDs can crash the game. When a problem appears, work through these boundaries:

1. **Launcher:** Is the JAR listed, and are required dependency IDs selected?
2. **Initialization:** Did the `@SpireInitializer` entry point run once? Is the first stack trace from your mod?
3. **Resources:** Are metadata, JSON, image and audio paths present inside the built JAR?
4. **Registration:** Did the relevant subscriber execute, and is the object in the intended pool?
5. **Gameplay:** Does the effect fire at the intended timing and target, including upgraded/copy variants?

6. **Persistence:** Does state survive save/quit/reload and a clean restart?
7. **Interaction:** Does the behavior change when another mod patches the same method or pool?

Record the game, ModTheSpire and BaseMod versions, enabled mod order, steps, expected/actual result and first relevant exception. A compile success is only stage zero. For a patch conflict, reduce to the smallest mod set and check the SpirePatch ordering rules before changing unrelated content.

Package and distribute

Release the JAR containing `ModTheSpire.json`, classes, localization, images and audio owned by the mod. Declare `basemod` or other required mod IDs in the ModInfo dependency fields, state tested game/loader/API versions, and include a license or permission notice for your own assets. Install the release artifact in a clean `mods/` folder with only declared dependencies, then run a new game and a saved-game check. Keep source and a release manifest for future migration.

The ModTheSpire and BaseMod repositories expose release and Workshop-related notes, but current Steam upload screens and metadata can change. Treat Workshop publishing as a separate release verification step: check the current Steam tooling, draft title/description/dependencies, upload a private or test item where available, install it as a subscriber, and confirm its files and dependency list. This chapter does not claim a successful Workshop submission.

Remaining validation

The documented paths cover the requested ecosystem at an architectural and release-workflow level, but the site has not compiled a sample mod against named installed versions or run the game. Detailed code examples for every object family, cross-mod compatibility and Workshop install testing remain open. The status stays **In progress** until those are performed.

Sources and provenance

- BaseMod Hooks, Custom Strings, Custom Savable, Mod Config and Panel, and Console — project-maintained registration, state and diagnostic references; checked 2026-10-06.
- ModTheSpire ModInfo, SpireConfig, and SpirePatch — loader metadata, configuration and patch behavior; checked 2026-10-06.
- This chapter is original ModDocs prose and contains no game files, third-party code or Workshop package.