Page contents

- Initialize once, then subscribe
- Follow the content registration path
- Choose a patch only when a hook does not fit
- Observe lifecycle and save boundaries
- Coverage still needed
- Sources and provenance

Offline copy of Subscribers, content registration and patches .

Slay the Spire: subscribers, content registration and patches

Documentation status: In progress. **Source checked:** 2026-10-06. **In-game validation:** not recorded. Work through the loader and project chapter first. This chapter distinguishes BaseMod's hook subscriptions from ModTheSpire's code patches.

Initialize once, then subscribe

ModTheSpire calls a public static, no-argument `initialize` method on a class marked `@SpireInitializer` after patching and before game load, according to its initializer reference. Make that entry point create one main mod instance and subscribe it to BaseMod. The BaseMod hook guide recommends keeping subscriptions on this main instance, since repeatedly subscribing transient objects can leave duplicate callbacks.

```
@SpireInitializer
public class YourMod implements EditStringsSubscriber, EditCardsSubscriber
    public YourMod() {
        BaseMod.subscribe(this);
    }
```

```
    public static void initialize() {
        new YourMod();
    }

    @Override
    public void receiveEditStrings() {
        // Load this mod's CardStrings JSON from its packaged resource pat
    }

    @Override
    public void receiveEditCards() {
        // BaseMod.addCard(new YourCard());
    }
}
```

This is a lifecycle sketch: add the matching imports and real resource paths from the project version you build against. The hook index assigns `EditStrings` to string-file registration and `EditCards` to card registration with `BaseMod.addCard`. Keep those responsibilities apart so a missing translation can be diagnosed before the card pool is involved.

Follow the content registration path

The BaseMod hook index groups registration callbacks by content: `EditCards`, `EditRelics`, `EditCharacters`, `EditKeywords`, `EditStrings`, and `AddAudio`. These are registration points, not a promise that a compiled class appears automatically in rewards, shops, saves or all character pools. For a first card, use a unique namespaced ID, package its image and `CardStrings` entry, register the card in `receiveEditCards`, then inspect it with the BaseMod console. Check its displayed name/description, use effect, upgrade, copy and save/reload behavior on a disposable run.

Before adding a second object family, write down its dependency chain. A relic may require localization and art; a potion or character may require additional registration, visual assets and pool behavior. The BasicMod example and BaseMod source show concrete paths for these object types, but each feature should be checked against the installed API. This chapter does not claim a complete implementation of powers, potions, events, characters or monsters.

Choose a patch only when a hook does not fit

BaseMod subscribers expose named lifecycle and content events. A `@SpirePatch` targets a particular game class and method and therefore depends more tightly on the game's binary shape. The ModTheSpire patch reference documents Prefix, Postfix, Insert, Instrument, Replace and Raw forms, as well as parameter matching and patch order. For a narrowly scoped change, first check whether a subscriber callback expresses the behavior. If it does not, identify the exact target method and its installed-game signature before writing a patch.

For patch debugging, begin with a patch that only logs entry; verify that it fires exactly where expected. Then add the change and test normal, upgraded and modded interactions as relevant. Avoid a wholesale game-file replacement: the ModTheSpire wiki warns that replacing the same game path conflicts with other mods. Patch order among multiple mods and overloaded methods can also affect the result; record the mod list and first stack trace when a conflict appears.

Observe lifecycle and save boundaries

The BaseMod hook table distinguishes callbacks before starting or continuing a game, after game start, at battle start, and during combat. Choose a callback based on what data exists at that time. For one-off subscriptions, use `BaseMod.unsubscribeLater` as the hook guide instructs; direct unsubscribe during callback iteration can cause `ConcurrentModificationException`. Persisting custom run state needs the documented save API and separate load tests; a field that survives one combat in memory is not proof that it survives a restarted run.

Coverage still needed

This is a framework map, not a complete card implementation. Dedicated, version-tested chapters are still needed for each object family, localization and asset pipelines, custom save fields, configuration, audio, compatibility tests, and Workshop distribution. Keep the guide status **In progress** until those paths are written and exercised.

Sources and provenance

- BaseMod Hooks and Console — subscription model, registration callbacks and testing tools; checked 2026-10-06.
- ModTheSpire SpireInitializer and SpirePatch — initialization and patch forms; checked 2026-10-06.
- BasicMod example — example structure; checked 2026-10-06. This chapter is original prose and contains only a short illustrative lifecycle sketch.