Page contents

- What the pack changes
- Pick an action by the size of the change
- Edit data without erasing neighbors
- Add conditions only after a static patch works
- Diagnose the loaded patch and the resulting asset
- Coverage still needed
- Sources and provenance

Offline copy of Content Patcher: assets, actions and conditions .

Stardew Valley Content Patcher: assets, actions and conditions

Documentation status: In progress. **Source checked:** 2026-10-06. **Target:** Stardew Valley 1.6 asset model and the Content Patcher author guide's current `2.9.0` format. **In-game validation:** not recorded. Start with the Parsnip price quick start for a complete two-file pack; this chapter explains the system behind it.

What the pack changes

A Content Patcher pack is a folder under `Mods` with `manifest.json` for SMAPI's identity and dependency information and `content.json` for the patches. An `assets` directory is conventional for your own images, maps, and included JSON. The pack describes changes to **assets loaded by the game**; it does not replace the installed `Content` files. The target is an asset name such as `Data/Objects` or `Portraits/Abigail`, without the `Content/` prefix, language suffix, or `.xnb` extension. Confirm an asset's name and data shape from the game data references before writing a patch.

The `Format` in `content.json` is the Content Patcher schema level the pack was written for, while the manifest's `Version` is **your pack's release**. The maintained author guide currently says `2.9.0`; pin that number only after checking the installed Content

Patcher release. When adapting an older pack, read the migration guide before changing `Format` blindly. JSON examples in upstream prose sometimes contain `//` annotations; remove them from files you actually ship.

Pick an action by the size of the change

Action	Use	Compatibility question
`EditData`	Add or change entries and fields in data assets	Can you touch only the needed field rather than replace an entry?
`EditImage`	Patch part of a texture or sprite sheet	Does the source area and target rectangle fit the intended sprite?
`EditMap`	Change selected map tiles or properties	Are the target coordinates and tilesheet references correct?
`Load`	Supply an entire asset from a pack file	Will replacing the asset overwrite another mod's contribution?
`Include`	Split patch definitions into more JSON files	Does every referenced relative path exist in the release ZIP?

The author guide's action index leads to the precise fields for each action. `Load` runs before edit actions on the same asset; subsequent patches see the result of earlier patches. Within one pack, patch order follows the list in `content.json`. Across packs, loader order and declared dependencies matter. A priority override exists, but it should be used for a real ordering need and tested with other mods.

Edit data without erasing neighbors

`EditData` sees a data asset as a dictionary or list. `Entries` can add, replace, or remove top-level entries. `Fields` changes individual fields of **existing** entries and is the safer choice for a small adjustment. In the quick start, `Fields` selects object ID `24` and sets its `Price`; it does not replace the object name, description, or other fields. For a new item, an `Entries` definition needs the data fields required by that game data model.

Deleting an entry uses a `null` value in `Entries`; doing so can affect saves and other mods that refer to the ID.

The EditData reference also documents `MoveEntries` for ordered lists, `TargetField` for an inner dictionary/list, and `TextOperations` for modifying an existing string. These are separate operations: an object field name is not interchangeable with a numeric index in an older slash-delimited value. Inspect the actual game asset and its version-specific data reference before choosing keys.

Add conditions only after a static patch works

Tokens such as `{{Season}}` substitute a value into a supported field, while `When` enables a patch only when its conditions match. That lets a pack choose an image by season or apply a data change only for a specified game state. A `ConfigSchema` can expose player choices as tokens. Translation files can provide text through the `i18n` token. These features do different jobs: a token provides a value, a condition decides whether a patch applies, and a config value comes from a player setting.

First test one unconditional patch. Then add one token or condition and check both a state where it should apply and one where it should not. An unavailable token can leave a patch inactive. The default field update rate is at day start; the `Update` field can request location or time changes where supported. That does not guarantee every game subsystem refreshes its visible content immediately; dialogue, for example, has its own day-start behavior. See the maintained token reference, config reference, and author guide for the exact current syntax.

Diagnose the loaded patch and the resulting asset

The SMAPI console can distinguish four failures that look identical in the game: pack not discovered, patch not loaded, condition false, and patch applied to the wrong or obscured asset. Work through them in order:

1. Confirm the pack appears in SMAPI's loaded-mod list. If absent, check folder depth, `manifest.json`, `ContentPackFor`, and JSON syntax.
2. Run `patch summary "Your.AuthorId"` with the pack's actual `UniqueID`. Read its loaded, conditions, and applied columns and any error text.

3. Run `patch summary asset Data/Objects` for a data target or the analogous asset name for art/maps. This shows other patches affecting it.
4. Run `patch export "Data/Objects"` to inspect the combined data after patches. For an image, inspect the exported PNG. An export shows the effective asset, not proof that a particular pack was solely responsible.
5. If you edit JSON while the game runs, `patch reload "Your.AuthorId"` reloads that pack's patches. The command does not reload every non-patch feature, so restart when testing schema, config, or dependency changes.

The Content Patcher troubleshooting guide documents these commands and `patch parse` for token text. Test with other packs disabled only to isolate a collision; restore the real dependency set for release testing. Keep a copy of the original save and record the game, SMAPI, and Content Patcher versions alongside the exported result.

Coverage still needed

This chapter maps the common action and diagnostic model. It does not reproduce the full token catalog, individual map/image field schemas, custom locations, text operations, translations, migrations, or Content Patcher's extension API. Each needs its own versioned reference and tested examples before this path is a primary technical manual.

Sources and provenance

- Content Patcher author guide — Pathoschild project; read 2026-10-06; pack structure, actions, update behavior and current format.
- EditData action reference — Pathoschild project; read 2026-10-06; entries, fields and patch options.
- Troubleshooting and console commands — Pathoschild project; read 2026-10-06.

- Stardew Valley modding index — community technical reference maintained on the game wiki; read 2026-10-06. This page is original ModDocs prose and includes no copied game assets or framework files.