Page contents

- Split a pack at a patch boundary
- Conditions and file selection
- Verify the effective result
- Release and preservation record
- Sources and provenance

Offline copy of Content Patcher: included files, text edits and conflicts .

Stardew Valley Content Patcher: included files, text edits and patch conflicts

Documentation status: In progress. **Source checked:** 2026-10-12. **Target:** Stardew Valley 1.6 data assets and Content Patcher format `2.9.0`. **Verification:** the JSON examples were checked for syntax and against the maintainer's documented fields; no independent in-game test is recorded. Start with the two-file Parsnip pack if you have not made a pack yet.

This chapter is for an author whose `content.json` has grown beyond one change. It explains how to split patch lists into files, change a delimited text value without replacing the whole entry, and find which patch produced an unexpected result. Both features are part of **game content modding**; the pack stays inside the game's `Mods` folder and leaves installed game assets intact.

Split a pack at a patch boundary

An `Include` action inserts another file's patches at that point in the parent `Changes` list. The included file contains a **single top-level `Changes` list**. Do not repeat `Format`, `ConfigSchema`, `CustomLocations`, or `DynamicTokens` inside it. A `FromFile` path is relative to the main `content.json`, even if one included file includes another. An

inclusion loop is invalid. These rules come from the maintained Include action reference.

For a small data pack, use this layout:

```
Mods/
  [CP] MyGiftTasteEdit/
    manifest.json
    content.json
    patches/
      gifts.json
```

The manifest identifies a Content Patcher content pack. Replace the sample author namespace before distributing your own pack:

```
{
  "Name": "My Gift Taste Edit",
  "Author": "Your Name",
  "Version": "0.1.0",
  "Description": "Adds one item to the universal loved-gift list.",
  "UniqueID": "YourName.MyGiftTasteEdit",
  "UpdateKeys": [],
  "ContentPackFor": {
    "UniqueID": "Pathoschild.ContentPatcher"
  }
}
```

`content.json` owns the format version and the include position:

```
{
  "Format": "2.9.0",
  "Changes": [
    {
      "Action": "Include",
      "FromFile": "patches/gifts.json",
      "LogName": "Gift taste additions"
    }
  ]
}
```

`patches/gifts.json` owns only its patch list:

```
{
  "Changes": [
    {
      "Action": "EditData",
      "Target": "Data/NPCGiftTastes",
      "TextOperations": [
        {
          "Operation": "Append",
          "Target": ["Entries", "Universal_Love"],
          "Value": "128",
          "Delimiter": " "
        }
      ]
    }
  ]
}
```

The `TextOperations` target is a *path through the loaded data model*, not a filesystem path. Here it selects the `Universal_Love` entry in `Data/NPCGiftTastes`. The text operation appends object ID `128` with a space between it and an existing value. The maintainer's text operation reference uses this same asset and path shape. Check the gift taste data reference and the actual exported asset for the game version you target before choosing other keys or IDs. Personal tastes can override universal tastes, so test the actual villager and item combination; a successful parse alone does not establish the gameplay effect.

An `Entries` patch that assigns a full `Universal_Love` string would replace previous text, including additions by another pack. `Append` preserves the value it receives from patches that ran earlier. It can still add the same ID twice if another pack already added it; inspect the exported result and coordinate compatibility as needed. `Append` and `Prepend` accept an explicit delimiter; if you omit it, the default delimiter may be wrong for this space-separated value. `RemoveDelimited` and `ReplaceDelimited` work on **whole delimited values**, not arbitrary substrings. Specify their `Delimiter` and `Search` deliberately. See the text operation reference for those fields and their `First`, `Last`, and `All` modes.

Conditions and file selection

The `Include` action can have a `When` condition or a tokenized `FromFile` path. Use a static include first so missing paths and invalid JSON are easy to find. If you later select files by season or player config, every possible selected path must exist in the release package, or the condition must prevent selecting absent files. Keep shared configuration such as `ConfigSchema` in the main `content.json`; included files are patch lists. `LocalTokens` on an include can supply values to its child patches, but document the token's intended scope before using nested includes.

Patches from an included file behave as if pasted at the include's position. Inside one pack, order follows `Changes`; across packs, the loader's order and manifest dependencies affect which edit sees which prior value. The maintainer's author guide says `Load` runs before edit actions on the same asset. For compatibility, prefer a narrow edit to replacing an entire data asset, then test alongside likely overlapping packs. Do not assume an `Include` creates a separate mod or an isolated patch priority.

Verify the effective result

1. Validate all three JSON files with a JSON parser. Then use the Content Patcher validator for the manifest and main content file; a schema pass does not replace a game test.
2. Launch Stardew Valley through SMAPI with Content Patcher installed. Confirm this pack appears in the loaded list. Run `patch summary "YourName.MyGiftTasteEdit"` and inspect whether the include and child edit are loaded, their conditions match, and the target has been applied.
3. Run `patch summary asset Data/NPCGiftTastes` to see other patches affecting the same asset. Run `patch export "Data/NPCGiftTastes"` and inspect the exported `Universal_Love` value. Check that `128` appears once and that the prior values remain.
4. Test the intended gift interaction in a disposable save. Remove this pack, restart, and compare the result. This distinguishes a visible gameplay effect from a pack that merely loads.
5. If you edit `patches/gifts.json` while the game is running, `patch reload "YourName.MyGiftTasteEdit" "patches/gifts.json"` can reload that include's patches. Restart for configuration, manifest, dependency, or other non-patch changes; the reload command does not refresh every pack feature.

If `loaded` is false, check JSON, `ContentPackFor`, the include path, and the pack ID. If `conditions` is false, inspect the condition and tokens. If `applied` is false, confirm the game has loaded the asset and that `Target` is the correct asset name. If the exported value differs from your expectation, inspect every patch reported for that asset and test with a minimal mod set. A `patch export` is the *combined* asset, so it does not by itself identify which pack wrote a value. The maintained troubleshooting reference explains these three summary columns and the patch commands.

Release and preservation record

ZIP the whole pack folder with `manifest.json`, `content.json`, and `patches/gifts.json` at the paths shown above. Reinstall that ZIP in a clean test `Mods` directory with its dependencies. Record the game, SMAPI, and Content Patcher versions actually tested, the pack's manifest version, and whether the result was checked in a save. Keep the editable JSON alongside the archive and this chapter's PDF, HTML, and Markdown downloads. The example on this page remains **source-reviewed, not independently game-verified**.

Sources and provenance

- Content Patcher author guide — Pathoschild project; checked 2026-10-12; pack structure, patch order, actions and format.
- Include action reference — Pathoschild project; checked 2026-10-12; included file shape, paths, conditions and limits.
- Text operations reference — Pathoschild project; checked 2026-10-12; operation fields, delimiters and gift taste example.
- Content Patcher troubleshooting — Pathoschild project; checked 2026-10-12; validator and console checks.
- Stardew Valley gift taste data — game wiki; checked 2026-10-12; game asset context.

This page is original ModDocs prose. The sample pack is a new teaching example; no game assets or upstream project files are redistributed.