Guide contents

Which path fits?

Content pack

C# mod

Testing and release

A complete first project: change a crop's sale price

Status

Sources / References

Documentation: In progress **Quick start: In progress**

Offline copy from ModDocs. Source links remain clickable.

Stardew Valley modding

Choose the smallest mod type that can make your change. The Stardew Valley wiki distinguishes content packs from C# mods; the former are usually easier to start with, while the latter can change behavior that a content pack cannot express.

> **Quick start:** make a small Content Patcher pack for an image or data edit, or a SMAPI C# mod for behavior. Load it in a test save and read the SMAPI console for errors before expanding it.

Which path fits?

Change	Good starting point
Replace or edit an image, dialogue, map, or game data	Content Patcher pack
Add behavior that reacts to game events	SMAPI C# mod

Change	Good starting point
Use features from another framework mod	A content pack for that framework, following its own schema

Both paths are loaded through SMAPI, but a Content Patcher pack does not need you to compile C#. Start with one visible change and a disposable save so it is easy to tell whether your pack works.

Content pack

For changes to images, dialogue, data, or maps, start with a Content Patcher pack. It works through SMAPI and describes edits in JSON files rather than replacing installed game files. Install the current supported SMAPI and Content Patcher versions, make one small change, then check SMAPI's launch output for errors before expanding it.

A content pack is a folder with a `manifest.json`, a `content.json`, and any files it needs. Content Patcher supports `Load` and edit actions for data, images, and maps. `Load` replaces the initial asset; edit actions are better suited to changing part of an asset and combining with other mods. Use the target asset name from the game's content system, not a guessed filename on disk. See the Content Patcher guide for the current format and patch actions.

```text
Mods/
  MyFirstPack/
    manifest.json
    content.json
    assets/
```

1. Install SMAPI and Content Patcher using their current instructions.
2. Create the pack folder and manifest. Give it a unique ID so it cannot collide with another author's pack.
3. Add one action to `content.json`, for one asset. Keep the first test small.
4. Launch through SMAPI and confirm the pack is reported as loaded. Enter a test save and look for the change.
5. If nothing changes, check the console for a JSON error, missing file, wrong asset name, or condition that is not true in that save.

Avoid editing the base game's installed files. Keep your original art or data in the pack folder so it is easy to distribute and remove.

- Modding index - choose a content or code path.
- Content Patcher guide - pack concepts and first test.

C# mod

Use the SMAPI mod guide when you need custom game logic. Follow its current setup and API references, build a minimal mod, and test it in a throwaway save before moving to complex behavior. Keep dependencies and target game version explicit.

The wiki's SMAPI getting-started guide currently describes a .NET class-library project, SMAPI's build configuration package, a class deriving from `Mod`, an `Entry` method, and a `manifest.json`. Use its current target framework and package versions; those requirements can change. A first C# mod can simply log when it loads. After that, subscribe to a documented SMAPI event, make one change, and verify that it fires only when the game state is ready.

```C#
using StardewModdingAPI;

namespace MyFirstMod;

internal sealed class ModEntry : Mod
{
    public override void Entry(IModHelper helper)
    {
        Monitor.Log("MyFirstMod loaded", LogLevel.Info);
    }
}
```

This is an illustrative entry class, not a complete project. The manifest, project file, package reference, and exact game target still come from the upstream guide. Build the DLL, run the game through SMAPI, and check its console output. Then install the built files in a clean `Mods` folder to verify release packaging.

- SMAPI mod getting started - C# project path.

Testing and release

Test in a throwaway save as well as a save with the game state your mod expects. Verify that the mod loads, the feature works, errors do not repeat in the SMAPI console, and the game still saves and reloads. Record required versions of Stardew Valley, SMAPI, Content Patcher, and any other mod. For a content pack, ship the manifest, `content.json`, and all referenced assets. For a C# mod, ship the manifest and the compiled DLL plus required supporting files. Follow the wiki's test and troubleshoot and release pages when preparing a public build.

A complete first project: change a crop's sale price

This **gameplay mod** changes the base sale price of the Parsnip object to 80g. It uses an `EditData` field patch, so it changes only one field in `Data/Objects` instead of replacing that whole game asset. The Content Patcher author guide demonstrates this field-edit pattern, and the Stardew object data reference defines `Price` as the player sale price. The item-query reference identifies `24` as the Parsnip object ID.

Guide revision: 0.3, source checked 2026-10-06. **Target:** Stardew Valley 1.6 data model and Content Patcher format 2.9.0 as documented by the maintained Content Patcher author guide. **Last independent in-game verification:** not recorded. This example is source-reviewed but should stay marked unfinished until tested against recorded game, SMAPI and Content Patcher versions. Back up your save folder before testing and use a disposable save; a sale-price edit changes the economics of a run. Keep your original game files intact.

First install SMAPI and Content Patcher from the original projects using the SMAPI player guide and Content Patcher project page. Launch once and confirm both load in the SMAPI console. The pack then consists of **exactly two JSON files** in its own folder under the game's `Mods` directory:

```text
Stardew Valley/
└── Mods/
    ├── ContentPatcher/              installed dependency; exact folder name may va
    └── [CP] ModDocsParsnipPrice/
        ├── manifest.json
        └── content.json
```

Create `manifest.json` with the documented content-pack manifest:

```json
{
  "Name": "ModDocs Parsnip Price",
  "Author": "Your Name",
  "Version": "0.1.0",
  "Description": "Sets the base sale price of Parsnip to 80g.",
  "UniqueID": "ModDocs.ParsnipPriceExample",
  "UpdateKeys": [],
  "ContentPackFor": {
    "UniqueID": "Pathoschild.ContentPatcher"
  }
}
```

`UniqueID` must be unique among installed mods. Change `ModDocs` to your own author namespace before distributing a derivative, and keep the chosen ID stable across updates. `ContentPackFor` tells SMAPI which mod reads this pack. This pack has no DLL and must not contain `EntryDll`.

Create `content.json` next to it:

```json
{
  "Format": "2.9.0",
  "Changes": [
    {
      "Action": "EditData",
      "Target": "Data/Objects",
      "Fields": {
        "24": {
          "Price": 80
        }
      }
    }
  ]
}
```

`Format` is the Content Patcher content-pack schema version, **not** your pack version. `Target` names the game's data asset. `Fields` selects an existing entry (`24`, Parsnip) and changes only its `Price`; it leaves name, description, sprite and other fields alone. If the installed Content Patcher rejects `Format`, consult its current author guide and update the schema version deliberately, rather than guessing. The JSON files cannot contain comments or trailing commas. A text editor that saves UTF-8 without adding `.txt` is sufficient.

Verify the pack and the effect

1. Launch through SMAPI. Confirm **ModDocs Parsnip Price** appears in the loaded content packs list. If it is absent, check the folder depth, both true `.json` extensions, the manifest and the Content Patcher dependency.
2. In the SMAPI console enter `patch summary`. Find the pack's `EditData Data/Objects` patch and check whether it is applied or reports an error. Content Patcher documents this command in its troubleshooting guide.
3. Enter `patch export "Data/Objects"` in the SMAPI console. Open the exported data file reported by that command and inspect object `24`: its `Price` should be `80`. This separates a successful data edit from other gameplay factors such as item quality or professions.
4. In a disposable save, sell a normal-quality Parsnip and compare the sale result with the unmodified game under the same conditions. Do not use a higher-quality crop or a sale-price profession for the first comparison; those can change the observed amount. Save, exit, reload and check again.
5. Close the game, move this pack folder out of `Mods`, restart and confirm the patched field is gone. Do not delete a user's save or the Content Patcher dependency to remove this one pack.

If `patch summary` says the target entry is absent, verify the game version and object ID rather than replacing the entire `Data/Objects` asset. If the exported `Price` is `80` but your sale result differs, look for quality, profession bonuses, or another mod changing sale calculations. If the export differs, test with other packs disabled and check patch order. A green “loaded” line only proves the pack was discovered; it does not prove the edit applied.

To release it, ZIP the **whole `[CP] ModDocsParsnipPrice` folder** so that `manifest.json` and `content.json` are directly inside that folder after extraction. Reinstall the ZIP in a clean test `Mods` directory with SMAPI and Content Patcher, verify the patch again, increment the manifest `Version`, and record the game/SMAPI/Content Patcher versions you actually tested. Do not include game files or a copy of SMAPI/Content Patcher in the pack. Preserve the two JSON source files, this page's offline PDF/HTML, and upstream URLs with the release.

Status

Unfinished: the complete Content Patcher files above are source-reviewed but have not been independently tested in-game. The separate C# first project has a full source layout, but no compiled or in-game validation is recorded. The Stardew Valley modding index remains the original source for version-specific schemas and APIs.

Continue with Content Patcher structure and patch behavior, tokens and player configuration, included files, text edits and conflicts, a custom object and crafting recipe, shop stock, prices and conditions, images, maps and locations, game data and version boundaries, a complete SMAPI source project, SMAPI events and game content, SMAPI save data and release, or SMAPI input, multiplayer and integrations for the separate authoring paths.

Sources / References

- SMAPI and Stardew Valley modding index
- Content Patcher project and source
- Content Patcher author guide
- EditData action reference
- Content Patcher troubleshooting and patch commands
- SMAPI content-pack manifest
- Stardew Valley object data