Page contents

- Follow the data path
- Add one entry to an existing shop
- Choose where a condition belongs
- Check the patch, data and menu separately
- Extending beyond an existing shop
- Sources and provenance

Offline copy of Content Patcher: shop stock, prices and conditions .

Stardew Valley: stock a custom item in a shop

Documentation status: In progress. **Source checked:** 2026-10-12. **Target:** released PC Stardew Valley 1.6.15 data, SMAPI 4.5.2 and Content Patcher format `2.9.0`. **Validation:** the supplied JSON and icon were checked locally against the cited data references; no independent in-game or multiplayer test is recorded. Use a disposable save and record your actual versions before releasing a derivative.

The custom object and crafting recipe tutorial makes an Archive Token but does not give shops a way to sell it. This chapter adds it to Pierre's existing stock for **50g**, with **five purchases per player per day**. The object's base sale price remains **5g**. Those are separate fields in separate game assets, so changing `Data/Objects` `Price` alone does not set a shop's purchase price. The Stardew shop data reference defines these fields and uses `SeedShop` as Pierre's shop ID.

Download the standalone shop example pack. It includes the token, icon, crafting recipe and shop entry in one Content Patcher pack. You do **not** need to install the earlier token pack as a dependency. Remove that earlier teaching pack while testing this one so two similar-looking tokens do not confuse the result. The two packs have different IDs and therefore define different saved items.

Follow the data path

The pack's `manifest.json` identifies it as `ModDocs.ExampleTokenShop`. Its `content.json` has four patches:

Patch	Asset	Result
`Load`	`Mods/{{ModId}}/Objects`	Loads the original 16 × 16 token icon from the pack's PNG
`EditData`	`Data/Objects`	Creates `{{ModId}}_Token`, an object with base sale price 5g
`EditData`	`Data/CraftingRecipes`	Gives it a one-Wood default recipe
`EditData`	`Data/Shops`	Adds that object to Pierre's `Items` list at 50g

See the downloadable pack or the custom object tutorial for the object, recipe and icon pattern. The shop patch below is the addition. To use your own namespace, change the pack's manifest `UniqueID` **before** distributing it and keep that ID stable thereafter. The built-in `{{ModId}}` token then keeps the item ID, texture asset and shop entry names together.

Add one entry to an existing shop

This is the complete shop patch from the example's `content.json`:

```
{
  "Action": "EditData",
  "Target": "Data/Shops",
  "TargetField": ["SeedShop", "Items"],
  "Entries": {
    "{{ModId}}_PierreToken": {
      "Id": "{{ModId}}_PierreToken",
      "ItemId": "(O){{ModId}}_Token",
      "Price": 50,
      "AvailableStock": 5,
      "AvailableStockLimit": "Player"
    }
```

```
    }
  }
```

`Data/Shops` is a lookup of shops. `TargetField` selects the `Items` list **inside** the existing `SeedShop` entry; `Entries` then adds one list member. The Content Patcher EditData reference explains nested targets, and the Stardew shop page shows this same list-edit pattern. Replacing the entire `SeedShop` model or its full `Items` array would be much more likely to erase another pack's stock.

The list member `Id` is the shop-entry identity, distinct from the `ItemId` being sold. Keep `Id` unique within this shop so later edits can find it. `ItemId` is **qualified** with `(O)` because the token is an object-type item. The `Data/Objects` dictionary key remains unqualified: `ModDocs.ExampleTokenShop_Token` when this pack loads. The item ID reference explains those two forms. If `ItemId` points to an absent item, the shop entry cannot become the intended product merely because its own `Id` is valid.

`Price: 50` is the gold charged by the shop for this entry. The object's `Price: 5` is its base sale price when the player sells it. `AvailableStock: 5` limits purchases; `AvailableStockLimit: "Player"` gives each player a separate limit in multiplayer. The other documented limit modes are `Global` (shared by the world) and `None` (reset for each shop-menu instance). The shop reference also defines `TradeItemId` and `TradeItemAmount` for barter, price modifiers, and optional conditions. Add those only after this fixed-price example works and test the final effective price in the actual menu.

Choose where a condition belongs

The example pack stocks the token in every season. To show it only in spring, add a **game state query** to the shop item:

```
{
  "Id": "{{ModId}}_PierreToken",
  "ItemId": "(O){{ModId}}_Token",
  "Price": 50,
  "AvailableStock": 5,
  "AvailableStockLimit": "Player",
  "Condition": "SEASON Spring"
}
```

This `Condition` is part of the game's shop-item model and is evaluated by the game when it builds stock. The shop documentation uses `SEASON Summer` in an item example, and the game state query reference defines query syntax. Test in a spring save **and** a save in another season. Pierre's Missing Stocklist can override season conditions in Pierre's shop, so use a test save without it when checking the seasonal rule. Record that caveat in a released mod that depends on seasonal availability.

A Content Patcher `When` condition instead controls whether the **patch** applies to the loaded `Data/Shops` asset. It is written as a JSON object with Content Patcher token rules, not as the game's query string. For a shop entry that should remain in the data but appear only when the player opens a qualifying shop, the item's `Condition` is usually the clearer place. Do not paste `"Condition": "SEASON Spring"` into a `When` object or copy a Content Patcher token expression into a game state query. The Content Patcher token guide documents the separate patch-level syntax.

Check the patch, data and menu separately

1. Put `[CP] ModDocs Example Token Shop` directly inside `Mods`, with `manifest.json`, `content.json` and `assets/Objects.png` inside that folder. Confirm SMAPI and Content Patcher load it without errors. Keep the original save backed up.
2. Run `patch summary "ModDocs.ExampleTokenShop"` in the SMAPI console. The `Data/Objects`, `Data/CraftingRecipes` and `Data/Shops` edits should be loaded. If an entry is absent, check the manifest ID, folder depth and JSON syntax.
3. Run `patch summary asset Data/Shops` to find other packs editing shop data, then `patch export "Data/Shops"`. Inspect `SeedShop.Items` for `ModDocs.ExampleTokenShop_PierreToken`, its qualified `ItemId`, purchase price and limit. Export `Data/Objects` separately to verify the referenced object exists.
4. Open Pierre's shop in a disposable save. Confirm the token icon, name and **50g** purchase price. Buy up to five; close and reopen the shop to check that the per-player limit remains for that day. Pass a day and check the reset. Do the equivalent on a second player only if you can run a multiplayer test.
5. Craft one token and compare its appearance with the purchased one. Save, exit and reload to check the purchased/crafted item persists. If you try the optional season condition, test both a matching and a nonmatching season.
6. Remove the pack only on a copy of the save. A saved custom item may become an Error Item when its defining mod is missing. Restore the pack for saves that

use its item.

If the patch reports applied but the item is absent from the menu, compare the exported shop entry with its `Condition`, the referenced object and the shop you actually opened. `patch export` shows the **combined** asset after active edits; it does not prove that a shop menu used that entry or that this pack was the last writer. If another mod also edits `SeedShop.Items`, use the patch summary and a controlled test with the other pack disabled to isolate the conflict. The maintained Content Patcher troubleshooting guide documents these console commands.

Extending beyond an existing shop

`Data/Shops` can also define a new shop model, but a new ID alone creates **no place for the player to open it**. The game wiki documents an `Action OpenShop` map tile property that connects a map interaction to a shop ID. A full custom shop then requires map editing, owner/opening behavior, stock, art and save testing. See the shop reference and map/location chapter before attempting that path. Shop items can also use item queries, trades and modifiers; the supplied pack intentionally uses one fixed object ID and gold price so each data link is visible.

Before a public release, test the final pack ZIP in a clean `Mods` directory, capture the exact game/SMAPI/Content Patcher versions, and document effects of other shop mods and multiplayer stock. The downloadable example and this chapter are **source-reviewed, not independently game-verified**. Preserve the editable JSON, original icon, PDF/HTML/Markdown copies and upstream source URLs with the release record.

Sources and provenance

- Stardew Valley shop data — game wiki; checked 2026-10-12; `Data/Shops`, `SeedShop`, nested `Items`, purchase price, stock, conditions and custom-shop entry point.
- Stardew Valley item IDs and game state queries — game wiki; checked 2026-10-12; qualified object ID and condition context.
- Content Patcher EditData, tokens and conditions, and troubleshooting — Pathoschild project; checked 2026-10-12.

This page is original ModDocs prose. The example pack and icon were created for ModDocs; no game files or third-party mod binaries are redistributed.