Page contents

- Files and dependencies
- Define the texture, object and recipe
- Test each link in the chain
- Adapt and release
- Sources and provenance

Offline copy of Content Patcher: custom object and crafting recipe .

Stardew Valley: add an object, icon and crafting recipe

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 PNG were checked locally; no independent Content Patcher load or in-game test is recorded. Work in a copy of a save and record your own exact versions before publishing a derivative.

This project adds an object-type **Archive Token**, gives it a 16 × 16 icon, and adds a default crafting recipe that turns one Wood into one token. It demonstrates three linked assets: `Data/Objects`, a custom texture under `Mods/<pack ID>/Objects`, and `Data/CraftingRecipes`. The game's item reference describes unqualified versus qualified IDs and custom textures; the object reference defines the data fields; and the recipe reference defines the slash-delimited recipe string. This is a small game-content mod, not a change to installed game files.

Download the example Content Patcher pack. The ZIP contains the editable JSON and original icon. It is a **source-reviewed teaching pack** until tested in a recorded game build. Keep that status attached if you share the archive. You can instead make all three files by following the listings below.

Files and dependencies

Install compatible SMAPI and Content Patcher releases using their maintained instructions, and launch once to confirm both are loaded. Extract the example ZIP so the pack folder sits directly under `Mods`:

```
Stardew Valley/
└── Mods/
    └── [CP] ModDocs Example Token/
        ├── manifest.json
        ├── content.json
        └── assets/
            └── Objects.png
```

Do not put another folder level between `[CP] ModDocs Example Token` and `manifest.json`. Keep a backup of the test save. If you change the author namespace or pack identity, update the manifest `UniqueID`; the `{{ModId}}` token below then keeps the custom item ID and texture path tied to it. Preserve that ID across future releases: existing saves may contain the item.

`manifest.json` identifies the pack to SMAPI and Content Patcher:

```
{
  "Name": "ModDocs Example Token",
  "Author": "ModDocs",
  "Version": "0.1.0",
  "Description": "Teaching example: adds a craftable object-type token.",
  "UniqueID": "ModDocs.ExampleToken",
  "UpdateKeys": [],
  "ContentPackFor": {
    "UniqueID": "Pathoschild.ContentPatcher"
  }
}
```

The pack itself contains no DLL. `ContentPackFor` points to the mod that interprets `content.json`. The manifest `Version` identifies **this pack's release**; it is separate from the `Format` in `content.json`.

Define the texture, object and recipe

Use a 16 × 16 transparent PNG at `assets/Objects.png`. The example ZIP supplies an original gold token icon. If you draw your own, keep the canvas at 16 × 16, export a true PNG, and check transparent pixels around the design. `SpriteIndex: 0` selects the first 16 × 16 cell of this custom texture. A larger sheet needs a matching index and a grid aligned to 16-pixel cells.

Put this in `content.json`:

```
{
  "Format": "2.9.0",
  "Changes": [
    {
      "Action": "Load",
      "Target": "Mods/{{ModId}}/Objects",
      "FromFile": "assets/Objects.png"
    },
    {
      "Action": "EditData",
      "Target": "Data/Objects",
      "Entries": {
        "{{ModId}}_Token": {
          "Name": "{{ModId}}_Token",
          "DisplayName": "Archive Token",
          "Description": "A small token made from reclaimed wood.",
          "Type": "Basic",
          "Category": -16,
          "Price": 5,
          "Edibility": -300,
          "Texture": "Mods/{{ModId}}/Objects",
          "SpriteIndex": 0
        }
      }
    },
    {
      "Action": "EditData",
      "Target": "Data/CraftingRecipes",
      "Entries": {
        "{{ModId}}_Token": "388 1/Home/{{ModId}}_Token/false/default/Archi
      }
    }
  ]
}
```

The `Load` patch makes the icon a game content asset named `Mods/ModDocs.ExampleToken/Objects` for this manifest. `Texture` points the object data at that **asset name**, not the PNG's disk path. The pack's `FromFile` resolves to `assets/Objects.png` inside the pack. The Content Patcher Load and EditData references define these actions.

`Data/Objects` is a string-keyed lookup. Its key is the **unqualified** item ID, here `ModDocs.ExampleToken_Token` after `{{ModId}}` resolves. The full qualified ID is `(O)ModDocs.ExampleToken_Token` because this entry is an object-type item. `Name` is an internal name; `DisplayName` and `Description` are player-facing text. `Type: Basic` and category `-16` put this example in the resource family; `Price: 5` is its base **sale** price, not a shop purchase price. `Edibility: -300` makes it inedible. The object field reference documents those fields and their defaults.

The crafting recipe is a **string**, unlike the object model. Read it by `/`-separated position:

Part	Example	Meaning
Ingredients	`388 1`	One Wood (`388`)
Legacy location label	`Home`	Required field; the modern game does not use it to restrict crafting location
Yield	`{{ModId}}_Token`	One object; a missing quantity defaults to one
Big craftable flag	`false`	The result is an object, not a big craftable
Unlock	`default`	Learned automatically by the game's normal default-recipe path
Display text	`Archive Token`	Name shown for the recipe

The recipe data reference documents this format and notes that missing default recipes are learned on day start. If the recipe does not appear immediately in an already-loaded save, sleep to the next day or start a fresh disposable save before treating the pack as broken. A recipe's key is its name in `Data/CraftingRecipes`; use a stable namespaced key so another pack is unlikely to overwrite it. If you later use the item in a shop, crop, machine, quest or gift taste, check that asset's own ID

convention. Some references accept qualified IDs while others store the unqualified ID.

Test each link in the chain

1. Parse both JSON files or run them through the SMAPI JSON validator. Confirm `assets/Objects.png` is really a PNG and not an image renamed from another format.
2. Start through SMAPI. Check that **ModDocs Example Token** appears among loaded content packs and that Content Patcher reports no error.
3. In the SMAPI console, run `patch summary "ModDocs.ExampleToken"`. Check the `Load` patch and both `EditData` patches. Then run `patch summary asset Data/Objects` and `patch summary asset Data/CraftingRecipes` to identify overlapping edits.
4. Run `patch export "Data/Objects"` and `patch export "Data/CraftingRecipes"`. Inspect the combined exported assets for the namespaced item entry and recipe entry. Run `patch export "Mods/ModDocs.ExampleToken/Objects"` and inspect the icon if the command can export that loaded image in your Content Patcher build.
5. In a disposable save, pass a day if needed, open the crafting menu, and craft one token using one Wood. Check its name, icon, inventory behavior and sale price. Save, exit and reload; check that the crafted token is still present and renders correctly.
6. Test uninstall **only on a copy of the save**. Removing a mod that defined a saved item may leave an Error Item until the pack is restored. Restore the pack before continuing a real save that uses it.

If the item entry exports but the icon is missing, compare `Texture` with the loaded asset name and check `SpriteIndex`. If the icon exports but the item does not, inspect the `Data/Objects` patch and JSON model. If the item exists but the recipe does not, check the recipe key, `Data/CraftingRecipes` export and `default` learning at day start. If the recipe appears but crafting gives the wrong item, compare the recipe's yield ID with the object's unqualified key. The Content Patcher troubleshooting guide explains `patch summary` and `patch export`; an exported asset shows the combined result from all active packs.

Adapt and release

Replace the teaching name, text, art, ingredient, sale price and author identity with your own work. If other players already have a release installed, treat its item ID and recipe key as persistent data. Changing either can break existing saved items or learned recipes; plan a migration before renaming. Add translations before publishing to multiple locales; see the Content Patcher translation chapter. Reinstall the final ZIP in a clean `Mods` folder with its dependencies, record the game/SMAPI/Content Patcher versions and test the crafting path on a new and an existing disposable save. Do not redistribute Stardew Valley game files or a copy of SMAPI/Content Patcher with the pack.

The shipped example is **not independently verified in-game**. Its ZIP, this page's PDF/HTML/Markdown downloads, the original icon, and the source links preserve enough material to repeat and correct the example when a tested environment is available.

Sources and provenance

- Stardew Valley item IDs and custom items — game wiki; checked 2026-10-12; ID forms, custom texture pattern and missing-item behavior.
- Stardew Valley object model — game wiki; checked 2026-10-12; fields for `Data/Objects`.
- Stardew Valley recipe data — game wiki; checked 2026-10-12; `Data/CraftingRecipes` layout and default learning.
- Content Patcher author guide, Load action, EditData action, and troubleshooting — Pathoschild project; checked 2026-10-12.

This article is original ModDocs prose. The icon and example files were created for ModDocs and contain no copied game art or project binaries.