**Page contents**


- What this project does
- Prerequisites and layout
- Project file
- Manifest
- Entry code
- Build, install and observe
- Release and remaining work
- Sources and provenance


Offline copy of SMAPI: a first C# gameplay mod .

# Stardew Valley SMAPI: a first C# gameplay mod

**Documentation status:** In progress. **Source checked:** 2026-10-12. **Target:** PC Stardew Valley 1.6.15, SMAPI 4.5.2, .NET 6 and the maintainer's ModBuildConfig 4.4.0. **Compiled/in-game validation:** not recorded. This is a complete *source layout* for a small mod, with build and test steps. It must be compiled against a locally installed game and SMAPI before use.

---

## What this project does

Pressing F2 while a save is loaded adds **one gold** to the current player's money and writes a confirmation to the SMAPI console. This deliberately tiny gameplay change shows the whole path from C# project to loaded DLL, event handler and visible result. Use only a disposable save: pressing F2 changes a saved value. Change the hotkey and behavior before publishing a derivative; F2 may already be used by another mod.

The project is different from a Content Patcher pack. It has an `EntryDll` and compiled code, while a Content Patcher pack has `ContentPackFor` and `content.json`. Do not

place both identity fields in the same manifest. The SMAPI manifest reference defines this distinction.

---

## Prerequisites and layout

Install the PC game, SMAPI and .NET 6 SDK, then create a **Class Library** project, not a .NET Framework project. The SMAPI getting-started guide explains why the target framework must be `net6.0` even if your development computer also has a newer SDK. Add the maintainer's ModBuildConfig package, which provides the game/SMAPI references and release packaging. The package version below is pinned to the maintainer repository's 4.4.0 project version as checked on 2026-10-12; recheck compatibility before changing it.

```
ModDocs.FirstCodeMod/
├── ModDocs.FirstCodeMod.csproj
├── ModEntry.cs
└── manifest.json
```

The project folder can live anywhere you keep source code. The **built mod folder** belongs under the game's `Mods` directory, with the manifest beside the DLL. Keep source files and game files out of a player release ZIP unless intentionally publishing source separately.

---

## Project file

Save this as `ModDocs.FirstCodeMod.csproj`:

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFramework>net6.0</TargetFramework>
    <Nullable>enable</Nullable>
    <Version>0.1.0</Version>
  </PropertyGroup>
  <ItemGroup>
    <PackageReference Include="Pathoschild.Stardew.ModBuildConfig" Version
```

```
  </ItemGroup>
</Project>
```

The target framework tells the compiler which runtime APIs the DLL may use; it is not the same as the SDK version installed on your development PC. The package reference tells the build where to find the game and SMAPI assemblies and how to prepare a mod release. If the package reports that it cannot find the game, configure its documented `GamePath` property to the actual game folder instead of copying random DLLs into this project.

---

## Manifest

Save this as `manifest.json` next to the project file:

```
{
  "Name": "ModDocs First Code Mod",
  "Author": "ModDocs Example",
  "Version": "%ProjectVersion%",
  "Description": "Adds one gold when F2 is pressed in a loaded save.",
  "UniqueID": "ModDocs.FirstCodeMod",
  "EntryDll": "ModDocs.FirstCodeMod.dll",
  "MinimumApiVersion": "4.1.0",
  "MinimumGameVersion": "1.6.14",
  "UpdateKeys": []
}
```

The build package's manifest token replaces `%ProjectVersion%` with the project `Version` in the packaged mod. Inspect the built manifest to confirm it contains `0.1.0`, not the literal token, before distributing the result. `EntryDll` must match the assembly filename. `UniqueID` must be unique and stable once released; change the `ModDocs` namespace to your own author ID before publishing a derivative. The minimum versions describe the supported floor, **not** an independent claim that every version above it was tested.

---

## Entry code

Save this as `ModEntry.cs`:

```
using StardewModdingAPI;
using StardewModdingAPI.Events;
using StardewValley;

namespace ModDocs.FirstCodeMod;

internal sealed class ModEntry : Mod
{
    public override void Entry(IModHelper helper)
    {
        helper.Events.Input.ButtonPressed += this.OnButtonPressed;
    }

    private void OnButtonPressed(object? sender, ButtonPressedEventArgs e)
    {
        if (!Context.IsWorldReady || e.Button != SButton.F2)
            return;

        Game1.player.Money += 1;
        this.Monitor.Log("F2 added one gold.", LogLevel.Info);
    }
}
```

`Entry` runs when SMAPI loads the mod, before a save necessarily exists. It registers a handler rather than changing the world immediately. `ButtonPressed` runs on a new button press; `Context.IsWorldReady` prevents access to `Game1.player` at the title screen. The `SButton` abstraction covers keyboard, mouse and controller codes. The SMAPI getting-started example documents this event, world-ready guard and console logging; the input reference explains button semantics. A production hotkey should be configurable with `KeybindList` and use `ButtonsChanged` for combinations, as described in input and integrations.

---

## Build, install and observe

1. Build the class library with the .NET 6 SDK. The ModBuildConfig package may copy the result into your local `Mods` directory during development. Confirm where it placed the files; do not assume an unfamiliar machine has the same game path.

2. Inspect the **built folder**. It needs `manifest.json` and `ModDocs.FirstCodeMod.dll` at its top level. Confirm the manifest token was expanded and that `EntryDll` matches the DLL filename.
3. Launch through SMAPI. Find the mod's name, ID and version in the loaded-mod list. If it is skipped, read the exact console error: missing DLL, JSON syntax, duplicate ID, incompatible game or SMAPI version, or missing game references are different failures.
4. At the title screen, press F2. No money change is possible because no save/player is loaded. The mod should not throw an exception.
5. Open a **disposable** save, note the current money, press F2 once, and verify exactly one gold was added. Check for the console confirmation. Repeat after saving, exiting and reloading.
6. Close the game, move only this mod's folder out of `Mods`, restart, and confirm F2 no longer changes money. Restore the original save from backup when finished testing.

If the DLL does not compile, first check the target framework, package restore, and game path. If the mod loads but nothing happens, check that the game window has focus, a save is loaded, and no other tool has intercepted F2. If the console line appears but the money display differs, inspect other mods and the game state before assuming the handler did not run.

---

## Release and remaining work

The ModBuildConfig release workflow can create a ZIP. Inspect and reinstall **that exact ZIP** in a clean `Mods` directory. Increment `<Version>` for each release, preserve the source revision, and record the game, SMAPI, package and operating-system versions actually tested. Do not include copies of the game or SMAPI in the mod ZIP. This example has no published update key; add one only when there is a real update endpoint.

This source layout is **not marked compile-tested or in-game verified**: this workspace has neither a .NET SDK nor a Stardew Valley installation available for that check. A production gameplay mod also needs configurable input, menu behavior, multiplayer ownership and broader compatibility testing. The SMAPI data/release and input/multiplayer chapters cover those decisions.

## Sources and provenance

- SMAPI getting started — Stardew Valley modder wiki; checked 2026-10-12; .NET target, project structure, event and world-ready pattern.
- SMAPI manifest and input API — wiki references; checked 2026-10-12.
- Mod build package and maintainer package project — checked 2026-10-12 for package behavior and pinned version. This page is original ModDocs prose and distributes no game assemblies.