Page contents

- Choose storage by lifetime
- Read a save entry safely
- Config and player-facing text
- Build and package the same files users will install
- Verification matrix
- Sources and provenance

Offline copy of SMAPI: settings, save data and release .

Stardew Valley SMAPI: settings, save data and release

Documentation status: In progress. **Source checked:** 2026-10-12. **Target:** the current Stardew Valley wiki SMAPI API and maintainer build-package documentation. **Compiled or in-game validation:** not recorded. Read SMAPI events and content first; this chapter covers where a code mod keeps state and how to ship a reproducible build.

Choose storage by lifetime

SMAPI gives a mod several places to put data. They are not interchangeable:

Data	Appropriate place	What to test
Player-selected options	`config.json` through the configuration API	Missing file, invalid values and upgrades
Progress tied to one save	`Helper.Data.ReadSaveData<T>` / `WriteSaveData`	Save, exit, reload and multiplayer host behavior

Data	Appropriate place	What to test
Data attached to a player, item, location or terrain feature	That entity's `modData` with a namespaced key	Clone/stack and network synchronization behavior
Static files supplied by the mod	Files in the mod folder or `Helper.ModContent`	File paths and case on each supported OS

The SMAPI data API says save data is only available after a save loads, is discarded if the player exits without saving, and is unavailable to a farmhand for writing in multiplayer. It identifies `GameLoop.Saving` as a useful point to prepare save data. `modData` persists on supported game entities and synchronizes in multiplayer; prefix each key with your mod's unique ID to avoid collisions. A `config.json` is a user preference for the mod, not per-save progress.

Read a save entry safely

This is a **focused API fragment**, not a standalone project. It belongs in a compiled `Mod` subclass after the getting-started project setup. Register for `SaveLoaded` in `Entry`; do not read save data at the title screen.

```
using StardewModdingAPI;
using StardewModdingAPI.Events;

internal sealed class ModEntry : Mod
{
    private const string DataKey = "progress";
    private ProgressData progress = new();

    public override void Entry(IModHelper helper)
    {
        helper.Events.GameLoop.SaveLoaded += this.OnSaveLoaded;
        helper.Events.GameLoop.Saving += this.OnSaving;
    }

    private void OnSaveLoaded(object? sender, SaveLoadedEventArgs e)
    {
        this.progress = this.Helper.Data.ReadSaveData<ProgressData>(DataKe
        this.Monitor.Log($"Loaded count {this.progress.Count}", LogLevel.D
```

```
    }

    private void OnSaving(object? sender, SavingEventArgs e)
    {
        this.Helper.Data.WriteSaveData(DataKey, this.progress);
    }
}

internal sealed class ProgressData
{
    public int Count { get; set; }
}
```

`ReadSaveData` can return `null` when there is no entry, so the example creates a default model. The key is scoped by SMAPI to this mod, but keep it stable across updates. Plan a migration if the shape of `ProgressData` changes; don't silently treat older saved fields as a new meaning. The example only loads and writes a value. Actual gameplay code must update `Count` at a deliberate event, check the host role if it controls shared state, and validate the resulting behavior in game. On returning to the title screen or loading another save, clear any state that must not carry over.

Config and player-facing text

The configuration API reads and writes a mod's own `config.json` through `Helper.ReadConfig<T>()` and `Helper.WriteConfig(...)`. Give the model useful defaults, reject impossible values in code, and test a newly installed mod as well as an older config with missing keys. Do not overwrite user selections as part of normal release packaging.

SMAPI's translation API reads `i18n/default.json` and locale files such as `es.json`; `helper.Translation.Get("key")` retrieves text, with fallback for missing locale-specific entries. Put text shown to players in translation files instead of C# literals. Test each release locale and keep keys stable. The SMAPI console's `reload_i18n` command can help during editing, but a mod that caches the returned string may still need a restart.

Build and package the same files users will install

The maintainer's mod build configuration package sets up game references, local deployment and a release ZIP for a .NET class library. Use the package version and target framework required by the current SMAPI getting-started guide; a newer installed SDK does not change the target runtime by itself. Your release folder needs `manifest.json`, the compiled entry DLL, any required dependencies, assets and `i18n` files. A source `.cs` file by itself cannot load as a mod.

Before publishing, inspect the ZIP instead of assuming the build step packaged correctly. After extraction, SMAPI should see one mod folder with the manifest and DLL at the expected relative paths. Keep development-only files, machine-specific paths, logs, saves and game-owned assemblies out. Preserve source and a source license separately from the player ZIP; record the commit or revision used to build it.

Verification matrix

1. Start with a clean `Mods` directory containing only the mod and declared dependencies. Confirm the manifest ID and version in the SMAPI console.
2. Launch without a save; confirm no save-data access or event handler assumes a player exists.
3. Load a disposable save with no prior mod data. Confirm default state, exercise the feature, save, exit and reload. Confirm persistence.
4. Test an older config/save shape before an update. If data migrated, preserve a backup and verify the migration can only run once.
5. If multiplayer is supported, test host and farmhand separately, including save/reload. Save-data writes should be host-owned where required.
6. Test on each claimed game/SMAPI version and operating system. Inspect logs for repeated errors, missing assets and dependency warnings.
7. Install from the actual release ZIP and repeat the smoke test. Document supported versions, dependencies, known incompatibilities and a removal path.

The manifest, test/troubleshoot and release references define version-specific metadata and distribution conventions. This chapter is source-reviewed; none of the matrix above has been performed for the illustrative code in this document. Input, multiplayer messaging, console commands, integration APIs, Harmony patches and full compile-tested projects still need separate coverage.

Sources and provenance

- SMAPI data API — Stardew Valley modder wiki; read 2026-10-12; save data and `modData` scope.
- SMAPI configuration API and translation API — wiki API references; read 2026-10-12.
- SMAPI events — wiki API reference; read 2026-10-12; lifecycle timing.
- SMAPI mod build package — framework maintainer documentation; read 2026-10-12; build and release ZIP behavior. Original ModDocs prose; no game files redistributed.