Page contents

- Input: watch the right event and respect keybindings
- Multiplayer: decide who owns state
- Mod integration: declare the real dependency
- When a code patch is necessary
- Compatibility and diagnostics matrix
- Sources and provenance

Offline copy of SMAPI: input, multiplayer and integrations .

Stardew Valley SMAPI: input, multiplayer and integrations

Documentation status: In progress. **Source checked:** 2026-10-12. **Target:** the current Stardew Valley modder wiki's SMAPI APIs. **Compiled or in-game validation:** not recorded. This chapter continues events and content and settings, save data and release for C# mods that interact with players or other mods.

Input: watch the right event and respect keybindings

SMAPI represents keyboard, mouse and controller buttons through `SButton`. `Helper.Input.IsDown` reads current state; `GetState` distinguishes pressed, held, released and idle states. For an action bound to a combination or alternate buttons, put a `KeybindList` in the mod's config model. A binding such as `F2, LeftShift + S` means **either** F2 or the two-button combination. Use `ButtonsChanged` and `KeybindList.JustPressed()` to trigger once. The input API warns against using `ButtonPressed` for a combination because it fires once for each constituent button.

```
using StardewModdingAPI;

public sealed class ModConfig
{
```

```
    public KeybindList ToggleKey { get; set; } = KeybindList.Parse("F2, Le
}
```

This is a config-model fragment, not a full mod. Read it with `Helper.ReadConfig<ModConfig>()` in `Entry`, subscribe to `helper.Events.Input.ButtonsChanged`, then check `config.ToggleKey.JustPressed()` in the handler. Also check whether the relevant menu or save state permits the action. Test keyboard, controller and split-screen claims individually. If your mod consumes an input before the game sees it, use the documented suppression API deliberately; suppression does not prevent another mod from responding to that same physical input. Do not suppress the player's normal action or menu key merely because your handler noticed it.

`GetCursorPosition()` exposes screen pixels, absolute map pixels, a tile, and the game's grab tile. A clickable overlay and a world tile edit need different coordinate systems. The wiki notes that pixel values are not automatically scaled for UI mode. Test at multiple zoom/UI scales before releasing any cursor-driven tool.

Multiplayer: decide who owns state

In a single-player save, there is one local game process. In multiplayer, the host and farmhands do not have identical state or save authority. The SMAPI data API limits save-data writing to the main player, while entity `modData` can synchronize as part of supported game objects. Keep a clear owner for each gameplay change. A farmhand action may need to send a request to the host, which validates it and performs the change; the host can then send a result or rely on game synchronization where appropriate.

The multiplayer API can enumerate connected peers and active synchronized locations, send typed messages and receive them through `Multiplayer.ModMessageReceived`. Give each message a type, check both `e.FromModID` and `e.Type` before deserializing it, and validate its payload and sender before applying a gameplay change. Limit destinations to the intended player/mod IDs when possible. The sender does not receive its own message back through that send call. A player may appear in connection information before the game has fully approved the connection, so test the timing of your chosen event.

Split-screen runs the same mod instance for multiple local screens. A field that appears player-specific may actually be shared between them. SMAPI's `PerScreen<T>`

exists for per-screen state. Test two local players with different settings and menus, and verify that an update event affecting one does not leak state to the other. Test reconnects and save/reload as well as an initial connection.

Mod integration: declare the real dependency

Use the manifest's `Dependencies` when another mod must load first. Mark an integration optional only when your mod can work without it. SMAPI then reports a missing required dependency to the player and orders installed dependencies before dependents. A dependency identifies another mod by stable `UniqueID`; a DLL filename or display name is not a substitute. The manifest reference documents minimum versions and `IsRequired`.

For runtime integration, `Helper.ModRegistry.IsLoaded("Other.AuthorMod")` checks presence. An API-providing mod can expose an object via `GetApi`; a consuming mod maps it to a **public interface** with `Helper.ModRegistry.GetApi<T>(...)`. The integration reference says to request another mod's API after all mods have initialized, commonly in `GameLoop.GameLaunched`, and to handle a missing optional API. Keep the interface version in mind: a mod may still be installed while its API has changed. Prefer a documented public API to reaching into another mod's private fields.

```
// In a GameLaunched handler, after all mods have initialized:
if (this.Helper.ModRegistry.IsLoaded("Other.AuthorMod"))
{
    IOtherModApi? api = this.Helper.ModRegistry.GetApi<IOtherModApi>("Othe
    if (api is not null)
    {
        // Call a method declared by the other mod's documented API.
    }
}
```

This is an integration pattern, not a copy-ready project: the interface and ID must come from the dependency's own documentation. Do not bundle a dependency's DLL inside your mod unless its license and integration instructions explicitly permit it. Test the no-dependency, supported-version and incompatible-version cases separately.

When a code patch is necessary

SMAPI provides events, content edits, input and mod APIs for many behaviors. If those cannot express the change, the Harmony guidance covers method patching. It warns that patches can conflict with other mods and break after game updates. The wiki recommends SMAPI's `EnableHarmony` build option and code API, careful exception handling, and postfixes where possible. A method's signature and behavior must be checked for the exact game version. Do not publish a Harmony patch based only on an example from another version; test normal flow, failure fallback and interactions with other patches.

Compatibility and diagnostics matrix

Scenario	Evidence to record
Keybinding	Keyboard/controller action fires once, doesn't trigger in the wrong menu, doesn't consume an unrelated game control
Save state	Title screen, new save, existing save, save/reload, return to title
Multiplayer	Host, farmhand, remote connection, split-screen, reconnect and ownership of changed state
Optional integration	Dependency absent, supported version present, unsupported version present
Method patch	Exact target version, error fallback, another mod patching the same method

Log enough context to reproduce a failure without writing private save content or large logs on every tick. Preserve the manifest, source revision, build output and tested version matrix with the release. This chapter maps common SMAPI API boundaries; a full code project, message schema, integration-specific interface, exhaustive input events and Harmony patch implementation remain to be separately verified.

Sources and provenance

- SMAPI input API — Stardew Valley modder wiki; read 2026-10-12; button state, bindings and cursor coordinates.
- SMAPI multiplayer API and data API — wiki references; read 2026-10-12; messages, peers and save ownership.
- SMAPI integrations, manifest, and Harmony — wiki references; read 2026-10-12. Original ModDocs prose; no dependency code redistributed.