Page contents

- What is released and what is a migration draft?
- Content Patcher edits game assets, not installed files
- Common 1.6 data families
- Follow a change across data and presentation
- Why game conditions and Content Patcher conditions differ
- Versioned verification workflow
- Sources and provenance

Offline copy of Game data: asset families and version boundaries .

Stardew Valley game data: asset families and version boundaries

Documentation status: In progress. **Source checked:** 2026-10-12. **Target:** released PC Stardew Valley 1.6.15, SMAPI 4.5.2, and Content Patcher's documented `2.9.0` pack format. **In-game validation:** not recorded. Game data and the Content Patcher schema have separate version numbers. Check both before adapting an example.

What is released and what is a migration draft?

As checked on 2026-10-12, the game wiki's current-version table lists PC 1.6.15; the SMAPI release page lists 4.5.2, requiring game 1.6.14 or later. The wiki also has 1.6.16 and 1.7 migration pages. The 1.7 page explicitly says that version is announced without a release date. These migration pages are useful for preparation, but do not use an unreleased schema as the default for a public 1.6.15 pack.

Version to record	What it controls	Where to confirm
Stardew Valley build	Game data models and runtime behavior	Game title/version and official release notes

Version to record	What it controls	Where to confirm
SMAPI version	Loader, C# API and supported game range	SMAPI console and maintainer release notes
Content Patcher version	Available patch features and pack format	Loaded-mod list and maintainer author guide
Pack `Format`	Content Patcher schema level used by `content.json`	Pack file and installed Content Patcher support
Pack manifest `Version`	Your mod release and update checks	`manifest.json` and release artifact

Record the **exact tested tuple** rather than only "works on 1.6." A data field may appear in a later 1.6 patch; an old wiki screenshot or raw data dump may describe 1.6.8, while the user's game has a newer build. If a migration page describes future changes, label any preview-only branch explicitly and keep a working released branch available.

Content Patcher edits game assets, not installed files

An asset name such as `Data/Objects` points to content loaded by the game. It is not `Content/Data/Objects.xnb` on disk, and it does not include a locale suffix. Content Patcher's `EditData` combines edits with the loaded asset. `Fields` adjusts parts of an existing entry; `Entries` adds, replaces or removes a whole entry. A full replacement is easier to conflict with another pack. The EditData maintainer reference distinguishes dictionary, list and model assets, and documents `TargetField` for an inner list/dictionary and `MoveEntries` for ordered lists. Determine the target's actual data shape before choosing one.

Common 1.6 data families

You want to change	Asset and key model	Related content to check
Object-type item	`Data/Objects`, string item ID → model	Icon texture and `SpriteIndex`; sale price versus shop price
Placeable big craftable	`Data/BigCraftables`, string ID → model	Texture, placement rules, crafting recipe
Crop growth and harvest	`Data/Crops`, seed item ID → model	Seed and harvest object IDs, crop sprite row, seasons
Processing machine	`Data/Machines`, qualified machine item ID → model	Input/output rules, produced items, ready time, multiplayer behavior
Shop stock	`Data/Shops`, shop ID → model	Item entries, conditions, per-player limits, availability
Cooking or crafting recipe	`Data/CookingRecipes` / `Data/CraftingRecipes`	Ingredients, result ID, unlock condition, localization

The `(O)` prefix identifies an **object-type** item and `(BC)` a **big craftable** in a qualified item ID. Some assets use the unqualified ID as their dictionary key, while references between assets use qualified IDs. Do not paste an `(O)` prefix into a key that expects only the object's ID. The item overview and object, big craftable, crop, machine, shop, and recipe references describe the game-side models represented here. This table is a map for choosing the right asset, not a substitute for each field definition.

Follow a change across data and presentation

The Parsnip price pack edits only `Data/Objects` entry `24`'s `Price`. That field controls the base player sale price, not every observed sale after quality/profession modifiers and other mods. A new crop is broader: the seed and harvested object IDs, `Data/Crops` growth model, crop texture, acquisition path (shop or other source), possible recipe/machine uses and translation text must all agree. A machine likewise needs a placeable item and processing rules in `Data/Machines`; adding only an icon does not create behavior.

Before writing a new entry, inspect the **live** target with Content Patcher's `patch export`, compare a similar existing entry from the same game build, and read the

asset-specific field reference. Use a globally unique ID based on your manifest identity for new content. Prefer adding an entry or targeting a narrow field over replacing the entire data asset. For lists nested within models, use `TargetField` and stable entry IDs when documented, so another mod's list members survive.

Why game conditions and Content Patcher conditions differ

`When` is a **Content Patcher condition** that decides whether that patch runs. A `Condition` field *inside* a game data model is interpreted later by the game's own data system, often as a game state query. They have different syntax and timing. For example, a shop item condition belongs in `Data/Shops` and can be evaluated when the shop opens; a Content Patcher `When` decides whether the shop-data edit exists in the loaded asset at all. Do not copy a `When` object into a game field expecting a query string. The shop reference and Content Patcher token guide define the two systems.

Versioned verification workflow

1. Record the exact game, SMAPI and Content Patcher versions and copy the save to a safe location.
2. Confirm the target asset and data shape on that build; check whether a wiki example is marked for an older or future version.
3. Apply one narrow patch, launch through SMAPI, and inspect `patch summary` for active/error state.
4. Use `patch export` to inspect the combined asset after all installed patches. Record relevant keys and field values.
5. Exercise the change in a disposable save. Test loading, saving, reloading, multiplayer if supported, and removal on a copy of the save.
6. Repeat with a likely overlapping pack. If behavior differs, compare the exported combined asset and patch order before adding a priority override.
7. Package the actual files tested, preserve the source revision and generated PDF/HTML documentation, and list the version tuple and known interactions in the release notes.

This chapter does not claim a complete field-by-field reproduction of every Stardew data model. Advanced item queries, game state queries, machine output rules, NPCs, events, quests, dialogue, mail, buildings and later migration branches still need deeper standalone coverage. Keep the guide **In progress** while those and in-game tests remain open.

Sources and provenance

- Stardew Valley game version table, SMAPI releases, and Content Patcher author guide — checked 2026-10-12 for released targets and pack format.
- Stardew Valley 1.6.16 migration and 1.7 migration — checked 2026-10-12 as future-branch documentation.
- Content Patcher EditData reference and Stardew game data index — original ModDocs synthesis; no game assets or copied raw data distributed.