Guide contents

Understand the three routes

Set up a content mod

Asset domains and references

When C# is needed

Test and distribute

A complete first project: a simple wand

Beyond the first item

Sources / References

Documentation: In progress **Quick start: Complete**

Offline copy from ModDocs. Source links remain clickable.

Vintage Story modding

Vintage Story supports three broad kinds of mods: visual **theme packs**, data-based **content mods**, and C# **code mods**. Many blocks, items, entities, and recipes are represented by assets that can be authored without a compiled plugin. The Vintage Story modding introduction recommends learning the asset system before adding code.

> **Quick start:** create a tiny content mod with a `modinfo.json` and an `assets/<yourmodid>/` domain, then add one item or block using the current wiki tutorial. Load it from the game's Mod Manager, verify it appears in a test world, and only then package the mod.

Understand the three routes

Route	Best for	Main files
Theme pack	Changing the presentation of existing content	Textures and other visual assets

Route	Best for	Main files
Content mod	Adding or adjusting blocks, items, recipes, entities	JSON assets, textures, shapes, patches
Code mod	Behavior that data alone cannot express	C# project plus supporting assets

A code mod often uses content assets too. Begin with a content mod if you are unsure: it teaches namespaces, paths, JSON structure, and testing without needing a compiler. The modding overview explains the boundaries of each route.

Set up a content mod

Use the Mod Manager's **Open Mods Folder** command to locate the active user-data folder rather than guessing the installation path. Create a folder for your mod. At its root, put `modinfo.json`; under `assets`, create a domain directory matching your mod ID. The content-mod setup guide also offers a template and example project if you prefer to inspect working files first.

```text
Mods/
└── firstexample/
    ├── modinfo.json
    └── assets/
        └── firstexample/
            └── ...content files...
```

A minimal manifest uses `type`, `modid`, `name`, `authors`, `description`, and `version`. This is an illustrative manifest shape; use your own unique ID and check the current modinfo reference before release:

```json
{
  "type": "content",
  "modid": "firstexample",
  "name": "First Example",
  "authors": ["Your Name"],
  "description": "A small learning mod",
```

```
  "version": "1.0.0"
}
```

Follow a current content tutorial for the actual asset being added. Do not assume a block, item, and entity share one schema. Copy a small example, change one identifier and asset at a time, and check the game's log or Mod Manager when it does not load. A JSON validator or editor with JSON5 support helps catch punctuation mistakes, but a syntactically valid file can still contain a wrong game property.

Asset domains and references

The directory under `assets/` is an asset **domain**. References in the same domain can be short; references to another domain need a prefix such as `game:` for base-game assets. For example, `game:block/...` refers to the base game while `firstexample:block/...` refers to your mod. The content-mod domain explanation is worth reading before copying a texture path: a missing prefix can make the game look for a base texture inside your own mod.

Keep your mod's files under its own folder. The game's mod installation guide notes that editing installed game assets directly is not a durable way to distribute changes. For an adjustment to existing content, use the documented patch approach so the modification remains a separate mod.

When C# is needed

If data assets cannot implement the behavior you need, move to a code mod. The code-mod preparation guide describes the SDK, IDE, game-path setup, and mod project templates. Its page explicitly records the Vintage Story version it was verified against; check that label and the current template instructions before following version-specific commands. The code basics portal links from environment setup to creating and testing a project.

Prototype a visible behavior in isolation. Keep the manifest and assets working, then introduce one code feature. If the mod stops loading, distinguish a manifest/path error from a C# build error or a game API error before changing multiple things.

Test and distribute

Test in a separate world and save a backup before changing an existing world. Verify the mod is enabled, the content appears, names and textures resolve, and the game reloads it after a restart. If the mod is meant for multiplayer, test with matching mod versions on client and server. Then ZIP the **contents** of the mod folder so `modinfo.json` and `assets/` are at the archive root. The content-mod publication steps describe this layout and the game's ModDB destination.

A complete first project: a simple wand

The wiki's simple item tutorial supplies a complete learning sequence: manifest, item definition, shape, texture, and language file. Use its downloadable starter workspace or recreate its structure under your own domain. An item with only a code and creative inventory entry will register, but it may display as a black box until its shape and texture are added. That makes a useful staged test:

1. Add `assets/<domain>/itemtypes/simplewand.json` with a unique `code` and a `creativeinventory` entry. Enable the mod and look for the new item in Creative Mode.
2. Add a shape under `shapes/item/` and a PNG under `textures/item/`; reference both with the matching domain and relative asset paths from the tutorial. Restart and check its appearance in inventory and in the world.
3. Add `assets/<domain>/lang/en.json` with the item's language key. Confirm the displayed name changes, then test the ZIP without the loose development folder present.

If the item is absent, check `modinfo.json`, folder nesting, and JSON parsing first. If it appears as a black box, the registration worked and the remaining problem is its shape or texture reference. If the name is an internal key, inspect the language file. This separation is more useful than changing all five files at once. The tutorial records the game version it was checked against; compare that before relying on exact asset properties.

Beyond the first item

The simple wand, art, language file and clean ZIP test form a complete first content mod. This walkthrough is source-reviewed, not locally run in Vintage Story here. Blocks, entities, recipes and C# behavior need their own schemas and tests; the content tutorials and code portal remain the detailed references. Some wiki tutorials were verified against older game versions, so compare their stated version with the one you use.

Sources / References

External project and documentation links cited in this guide. Check the upstream release and version before following version-sensitive steps.

- Vintage Story modding introduction
- content-mod setup guide
- modinfo reference
- content tutorial
- game's mod installation guide
- code-mod preparation guide
- code basics portal
- simple item tutorial