Page contents

- Select one branch before copying an example
- Read the generated project as a map
- Build a controlled baseline
- Coverage still needed
- Sources and provenance

Offline copy of tModLoader project and branch structure .

Terraria: tModLoader project and branch structure

Documentation status: In progress. **Source checked:** 2026-10-06. **In-game validation:** not recorded. This chapter expands the preserved generated-sword quick start. Its setup follows the maintained Basic tModLoader Modding Guide; the exact files and APIs must be checked against the installed branch.

Select one branch before copying an example

tModLoader has multiple Terraria compatibility branches. The current basic guide describes a **.NET 8** project setup, while the separate Terraria 1.4.5 migration draft describes **.NET 10** and breaking source changes. That draft calls itself work in progress. An example from the 1.4.5 branch is not a drop-in replacement for an example from `stable`, even if the class names look similar.

Write down four values before starting: installed Terraria branch, installed tModLoader build, .NET SDK target from the matching guide, and the Git/API documentation branch used. The stable ExampleMod is the default companion for the stable workflow. If working on an explicit preview branch, open that branch's ExampleMod and migration notes together. The `/docs/stable/` API reference reports its own build at the top of each class page; check it when a method signature conflicts with an example.

Read the generated project as a map

Use **Workshop** → **Develop Mods** → **Create Mod**, then inspect the generated `ModSources/<ModName>` folder before editing. The basic guide lists these roles:

File	Purpose	What to verify
`<ModName>.cs`	The single root `Mod` class	Does its namespace/name match the project?
`<ModName>.csproj` and `Properties/launchSettings.json`	IDE/build and debugging setup	Does the project target the installed toolchain?
`build.txt`	Mod version, author, display name and build options	Are dependencies and packaging settings deliberate?
`Content/Items/<ItemName>.cs` and `.png`	Generated sword class and sprite	Do class, asset path and localization agree?
`Localization/en-US_Mods.<ModName>.hjson`	English player-facing text	Are the item name and tooltip keyed to the content?
`description.txt`, `description_workshop.txt`, icons	In-game and Workshop presentation	Are release text and artwork present?

This is a **source project**, not the distributable. **Build + Reload** compiles it and loads a `.tmod` package. The basic guide locates the package in tModLoader's `Mods` folder. Keep C# source, original art, HJSON, project settings and release notes in version control or a backup; a `.tmod` alone is a poor preservation copy for future API migrations.

The build.txt reference documents `modReferences` for required mods, `weakReferences` for optional integrations, `buildIgnore`, `includeSource`, and the `side` setting. `weakReferences` does not make all code paths safe when the other mod is absent: guard optional integration at runtime and test without it. `buildIgnore` can

remove design files from a release, but inspect the packaged `.tmod` to ensure required sprites, sounds and localization were not excluded.

Build a controlled baseline

First build the unmodified generated project. The goal is to prove the installed SDK, tModLoader build and generated template work together. Enter a disposable world, craft or obtain the sword and read its displayed name, tooltip, damage and sprite. Then change one value and rebuild. If the compiler reports a missing override or type, compare the exact method signature to the **same branch's** API class page and ExampleMod source before changing the project target or adding packages.

When the feature grows, organize by content families (`Items`, `Projectiles`, `NPCs`, `Tiles`, `Systems`) without assuming directory names alone register behavior. C# class inheritance and the relevant tModLoader base type determine registration. Keep namespaces and content IDs clear; `ModContent.ItemType<T>()` and analogous helpers look up registered types rather than relying on a guessed numeric ID.

For release, build once from a clean source state, install the resulting `.tmod` as a player would, and test with only declared dependencies. The basic guide explains a direct `.tmod` share and Workshop publishing. Record the actual tModLoader and Terraria builds used for the test. A compile success proves only that the code built; it does not prove a multiplayer host, packaged asset, or world upgrade works.

Coverage still needed

The project is only one branch of a large modding ecosystem. Platform-specific setup, exact branch compatibility tables, debugging sessions, Workshop metadata and distribution policy, build options, legacy migration and repeatable example projects need dedicated versioned pages. Continue with content classes and hooks and save/multiplayer state for the API structure.

Sources and provenance

- Basic tModLoader Modding Guide — tModLoader project wiki; checked 2026-10-06; generated project and first build.
- Stable ExampleMod and build.txt reference — project-maintained examples and metadata; checked 2026-10-06.
- 1.4.5 migration draft — project-maintained, explicitly unfinished branch guide; checked 2026-10-06.
- tModLoader repository lists MIT for its code. This ModDocs page is original prose and does not include Terraria assets, project binaries or a copy of ExampleMod.