Page contents

- Three kinds of extension
- Read a hook contract before overriding it
- Type IDs are not instance indexes
- Build features in testable slices
- Sources and provenance

Offline copy of Content classes, hooks and API navigation .

Terraria: content classes, hooks and API navigation

Documentation status: In progress. **Source checked:** 2026-10-06 using the tModLoader stable API pages and matching ExampleMod. **Compiled or in-game validation:** not recorded. Begin with the generated-sword quick start if this is your first mod.

Three kinds of extension

tModLoader's API is organized around the type of game content or behavior being extended. A class derived from `ModItem` defines **one modded item**. A `GlobalItem` can run hooks for **many items**, including vanilla and other mods' items. A `ModPlayer` attaches behavior/state to a player; a `ModSystem` owns broader world or game systems. The central `Mod` class is the mod's load/integration root, not a place to put every item, world save field and player effect. This separation is important when a feature crosses content boundaries.

Need	Start with	Then inspect
New sword, accessory, consumable or weapon	`ModItem`	`Item` defaults, recipes, texture, use/shoot hooks

Need	Start with	Then inspect
New projectile, NPC, tile, wall or buff	Matching `ModProjectile`, `ModNPC`, `ModTile`, `ModWall`, `ModBuff`	Lifecycle, spawn/placement, sprite and sync hooks
Modify behavior of existing items	`GlobalItem`	Applicability and per-entity state
Player-specific effect or save value	`ModPlayer`	Reset/update, save and network hooks
World generation or world-wide state	`ModSystem`	World lifecycle, save and net hooks

The API reference contains many more classes, including prefixes, biomes, UI layers, dust and sound helpers. Choose a concrete feature, find the closest ExampleMod example **on the same branch**, then inspect the corresponding class reference. A class page lists hooks and their call location; that matters in multiplayer and in menus where no world is loaded.

Read a hook contract before overriding it

A tModLoader "hook" is typically a virtual method that a derived class overrides. Its return value often affects whether vanilla logic or other hooks continue. The same method name may change parameters across branches. Copy the exact signature from the branch's API page into the correct class; an override in `ModItem` cannot simply be moved into `GlobalItem` because the latter usually receives an `Item` parameter for the item being processed.

`SetStaticDefaults` is for type-level setup that is shared across instances. `SetDefaults` configures the item or entity instance. `ModItem.AddRecipes` registers recipes during the recipe phase; the Recipe reference distinguishes creating/registering recipes from editing existing ones. `ModItem.Shoot`, for example, controls weapon projectile creation; its client/server call behavior must be checked before putting persistent world changes inside it. A texture path and localization key complete the feature's player-visible side.

In `GlobalItem`, default behavior applies broadly. Narrow it using the documented target checks or `AppliesToEntity` where appropriate. If the extension must hold

different mutable values for separate item instances, inspect `InstancePerEntity` and cloning/saving hooks; a shared global object field can otherwise leak one item's state into another's. The GlobalItem reference states which hooks are called on local client, server, or remote clients.

Type IDs are not instance indexes

The basic guide warns that both a content **type** and an active instance's array **index** are integers. `ModContent.ProjectileType<MyProjectile>()` retrieves a registered content type; a projectile's `whoAmI` identifies an active instance's slot. Using a type ID as an array index can compile and silently address the wrong object. Prefer the API's typed lookup and use instance indexes only when a hook or event explicitly supplies one. This distinction applies to items, NPCs and other content families.

Build features in testable slices

For a new weapon, add the item and sprite first, then its recipe, then a projectile, then any on-hit effects. Verify each layer: the item loads and localizes; the recipe makes it obtainable; the projectile spawns and behaves; server and clients see the same effect. Use the ExampleMod content tree as an example index, not as a single file to copy wholesale. For a global behavior change, test with both vanilla and another mod's item to ensure the applicability rule is narrow enough.

The class references are large API documents. This chapter establishes how to navigate them and why the extension families differ. It does not reproduce every method for NPC AI, bosses, world generation, tiles, UI, sound, biomes or equipment. Those need separate versioned technical pages and tested examples before Terraria is a complete reference.

Sources and provenance

- tModLoader basic guide — hooks, type/index distinction and ExampleMod usage; checked 2026-10-06.

- `ModItem`, `GlobalItem`, `ModPlayer`, `ModSystem`, Recipe — project API reference; checked 2026-10-06.
- Stable ExampleMod — project source/examples; checked 2026-10-06. This chapter is original ModDocs prose and does not redistribute ExampleMod or game files.