Page contents

- Save scope follows the object
- Follow ownership in multiplayer
- A reproducible test matrix
- Coverage still needed
- Sources and provenance

Offline copy of Save data and multiplayer state .

Terraria: save data and multiplayer state

Documentation status: In progress. **Source checked:** 2026-10-06 against the project Basic Netcode guide and stable API reference. **Host/client validation:** not recorded. This chapter follows content classes and hooks; it separates three jobs that are easy to confuse: storing a value, showing it on another machine, and deciding which machine is allowed to change it.

Save scope follows the object

Choose the owner of a value before adding a save hook. A player's permanent progression belongs with a `ModPlayer`; a world-wide flag belongs with a `ModSystem`; data unique to an item instance belongs with its `ModItem` or applicable `GlobalItem`. `SaveData`/`LoadData` for player and item data, and `SaveWorldData`/`LoadWorldData` for system state, use a `TagCompound`. Write only values that differ from defaults when the API recommends it, supply defaults for older saves, and test removing/reinstalling the mod on a **copy** of a save.

State	Example owner	Persistence API to inspect	Network question
Player's custom	`ModPlayer`	`SaveData` / `LoadData`	Must server and other clients know it?

State	Example owner	Persistence API to inspect	Network question
counter			
Boss defeated by world	`ModSystem`	`SaveWorldData` / `LoadWorldData`	Is the server the only writer, and when is world data sent?
Per-item mutable charge	`ModItem` or `GlobalItem`	Item save/load hooks	Does an inventory transfer or drop need a sync?

A save hook does **not** automatically make a value current on all connected clients. Conversely, a packet that updates a client does not automatically make a value survive save/reload. Test both properties separately.

Follow ownership in multiplayer

Clients communicate through a server; one client does not send directly to another. The netcode guide says the server owns NPC/world state, while a player-owned projectile can originate on its local client. A hook may run on the server, on one client, or on all of them. Its API documentation states this when relevant. Drawing hooks are client-side presentation; they are not a reliable place for a game rule that must exist on the server.

Many base properties synchronize automatically when used as intended. Extra NPC or projectile AI data may need `SendExtraAI`/`ReceiveExtraAI`; a `ModSystem` can use `NetSend`/`NetReceive` when world data is transmitted. A `ModPlayer` uses its `SyncPlayer`, `CopyClientState` and `SendClientChanges` pattern for data other machines need. Item-specific state has its own network hooks. The exact signatures and call timing belong to the stable API pages; use a small ExampleMod feature as a branch-matched implementation reference before writing a custom packet.

For world state, decide whether a change is server-authoritative. Apply it once on the server, then trigger the documented world-data sync when needed. For random NPC behavior, make the authoritative choice and synchronize the result; allowing each client to choose randomly can produce visible corrections or desync. For a player UI button that changes shared state, the client should request an action, the server should validate it, and the resulting state should be broadcast using the relevant

tModLoader mechanism. A custom `ModPacket` is useful when built-in sync hooks cannot carry the action or data, but the packet's message type, ordering, sender validation and length become part of your protocol.

A reproducible test matrix

1. In single-player, create the value, save, quit and reload. Confirm it persists and defaults correctly on an older test save.
2. Host a disposable world with the mod and connect a second client using the **same tModLoader branch**. Perform the action on the host, then on the joining client. Observe both screens and the server log.
3. Disconnect and rejoin the client. A value that only existed in one session should be distinguished from save-backed state.
4. Test an inventory transfer or dropped item if the value is per-item. Test world load after a mod version change if the schema changed.
5. Temporarily add focused logging for sender, net mode, entity ID and value at each change. Remove noisy logs before release, and test the packaged `.tmod` rather than only source builds.

The netcode guide describes a local two-client test and why `ModPacket` handling differs from automatic syncing. Claims of multiplayer compatibility need at least host and remote-client validation. A successful single-player save/reload alone does not cover it.

Coverage still needed

This chapter maps ownership, persistence and sync responsibilities. It does not provide a complete packet protocol, all entity sync hooks, compatibility with every content family, or a tested multiplayer sample. Those need branch-specific reference pages and runs against recorded tModLoader builds. Terraria remains **Documentation: In progress**.

Sources and provenance

- Basic Netcode — tModLoader project wiki; checked 2026-10-06; ownership, automatic sync, packets and testing.
- `ModPlayer`, `ModSystem`, `ModItem`, `GlobalItem` — stable API reference; checked 2026-10-06; save and network hook families.
- Stable ExampleMod — project-maintained examples; checked 2026-10-06. This page is original ModDocs prose and contains no copied game assets or project binaries.