Page contents

- Choose the script context before writing a callback
- A small 2.1-style diagnostic script
- Know where state lives
- Find the right API family
- Troubleshoot by boundary
- Sources and provenance

Offline copy of Scripting: contexts, callbacks and handles .

Teardown scripting: contexts, callbacks and scene handles

Documentation status: In progress. **API version represented:** 2.1.0, as reported by the official Teardown scripting API on **2026-10-06**. **ModDocs page updated:** 2026-10-06. These explanations have been checked against the reference but have not been run in-game. The older quick start describes the original single-player route; use the callback form documented for the game version and mod mode you actually target.

Choose the script context before writing a callback

The 2.1.0 reference describes a multiplayer-aware **server/client** model. The host runs the server and is also a client; other players run clients. Each script has its own Lua context, so a Lua global in one script is not a shared variable in another. The engine supplies API calls and a hierarchical **registry** for interaction beyond that local context.

Server callbacks include `server.init()`, `server.tick(dt)`, `server.update(dt)`, `server.postUpdate()`, and `server.destroy()` for game-mode scripts. Client callbacks include `client.init()`, `client.tick(dt)`, `client.update(dt)`, `client.postUpdate()`, `client.draw()`, and `client.render(dt)`. These callbacks are optional. Put state-changing game logic on the server when the result must be authoritative for all

players; use client drawing for a player's overlay. The official callback table is the source for the exact timing and availability of each callback.

`tick(dt)` runs once per rendered frame; `update(dt)` runs at a fixed physics rate and can run zero, one, or two times in a frame. Read edge-triggered input in `tick`, not `update`, or a press may be missed or processed twice. `client.draw()` is the UI drawing callback. If the same script runs on both sides, keep server-only and client-only behavior in their respective callback tables rather than assuming a function is safe everywhere.

A small 2.1-style diagnostic script

Attach a script to a local test level and give a **shape** the tag `moddocs_target` in the editor. In the script, check for that shape once, then display whether the handle remains valid. This deliberately changes no gameplay state:

```
local target = 0

function client.init()
    target = FindShape("moddocs_target", true)
    DebugPrint("Target handle: " .. tostring(target))
end

function client.tick(dt)
    if target ~= 0 and IsHandleValid(target) then
        DebugCross(GetShapeWorldTransform(target).pos)
    end
end
```

The returned number is a **handle**, an identifier for an engine object, rather than a Lua object. Zero means the lookup found nothing. `IsHandleValid` guards against using a handle after its object has been removed. `FindShape` takes a tag and an optional global-search flag; `true` searches the full scene. This client-side example draws only a diagnostic marker. If the marker is missing, verify that the tag is on a shape, the script loaded, and the reference's callback style matches your installed release. The editor chapter explains why hierarchy affects a non-global search. The function names and handle behavior are described in the Entity and Shape sections of the official API.

Know where state lives

The `server` table exists only on the server, and `client` exists on clients. The `shared` table synchronizes data from server to client code in the same script; clients treat it as read-only. Avoid copying large frequently changing tables into `shared`, since those changes consume bandwidth. The registry is a separate hierarchical key/value store. The API reserves namespaces such as `game` and `options`; use the documented mod persistence area `savegame.mod` for saved mod data, and a deliberate `level` key for level-specific coordination.

Use an event when another script should react to a discrete occurrence, and the registry when it should read a value. The official reference documents `PostEvent`, `GetEvent`, and `GetEventCount`, plus built-in events such as player damage and explosions. Before wiring an event, note whether it is delivered to server, client, or both; applying the same gameplay change on each receiver can duplicate an effect.

Find the right API family

Task	Start with	What the category gives you
Locate objects and inspect tags	Entity, Body, Shape	Search functions, relationships, transforms, properties and handle checks
Move or affect physical things	Body, Shape, Scene queries	Velocity, impulses, bounds, ray and overlap queries
Make 3D positions	Vector math	Vectors, quaternions and transforms represented as Lua tables
React to a player	Input, Player, Events	Input state, player information and game events
Draw an overlay	UI	Drawing and interactive widgets from the client drawing context
Play a sound	Sound	Load, play, stop and inspect sound playback handles

Task	Start with	What the category gives you
Diagnose a scene	Debug and drawing functions	Text, markers and lines visible while testing

The official API reference provides signatures, parameter types, return values and examples. This ModDocs chapter covers the context model and a working search pattern; it is **not yet** a replacement for the full function reference. Separate, navigable API category pages are required before Teardown's scripting coverage is comprehensive.

Troubleshoot by boundary

Symptom	First check
Callback never runs	Script path, active mod/level, callback spelling, and API version
Shape handle is zero	Object type, exact tag, and scoped versus global search
Handle worked and later fails	Whether the object was deleted or replaced; call `IsHandleValid`
UI drawing fails	Whether code runs inside `client.draw()` and uses a client-supported function
Multiplayer state differs by player	Which side changed it, whether `shared` was updated by the server, and whether the action belongs on the server
Input acts twice or is missed	Whether edge-triggered input was read in `update` instead of `tick`

Sources and provenance

- Teardown scripting API — Tuxedo Labs; reference reports version 2.1.0; checked 2026-10-06. Original documentation is linked for exact signatures and availability; ModDocs prose and the diagnostic example are original.
- Teardown modding manual — Tuxedo Labs; checked 2026-10-06. Its older single-player scripting discussion remains useful but requires version context for newer multiplayer-aware releases. A redistribution license was not identified.