

Respawning a Destroyed Vehicle in Hammer

A step-by-step guide to using `point_template` so a vehicle rebuilds itself after it's destroyed

Why This Is a Hammer Job, Not a Plugin Job

Each vehicle isn't one entity — it's a whole cluster: the `game_ui`, the `func_physbox` chassis, half a dozen `phys_thruster` entities, a `phys_keepupright`, plus doors, sounds, and lasers, all wired together by exact name. To "respawn" one from a SourceMod plugin, the plugin would have to recreate every single one of those entities from scratch with the exact right keyvalues and I/O wiring — essentially reimplementing, in fragile code, something the engine already has a purpose-built tool for.

That tool is **`point_template`**. It snapshots a group of entities (connections and all) and can stamp out a fresh, identically-wired copy on command. This is entirely Hammer work — no plugin code involved — and it's robust because the engine itself is doing the entity recreation, not anyone's guess at it.

Try This First: A 5-Minute Test

Before building the full respawn system, it's worth checking whether a much simpler swap already gets you most of the way there. **`func_physbox_multiplayer`** (a variant already used elsewhere on this map, just not for these vehicles) has native reset/respawn behavior built in specifically to stop multiplayer physics props from staying destroyed forever.

1. Open the vehicle's chassis entity in Hammer and change its class from `func_physbox` to `func_physbox_multiplayer`.
2. Since the `phys_thruster` entities reference the chassis by *name* (not classname), this swap alone shouldn't break anything else in the vehicle's wiring.
3. Compile and test: destroy the vehicle and see whether it comes back on its own.

If that gets you the behavior you want, you're done — skip the rest of this guide. If it doesn't respawn the way you need (e.g. you want a controlled delay, an announcement, or a limit on how many times it can respawn per round), continue on to the full `point_template` approach below.

Step 0: Confirm How the Vehicle Actually "Dies"

This matters more than it sounds like it should. `point_template` respawn logic needs to hook onto *something* that fires when the vehicle is destroyed — and there's more than one way a `func_physbox` can go away:

- **Health-based breaking:** if the chassis has a Health value set and takes enough damage, it fires an OnBreak output and gibs. This is the easiest case — hook OnBreak directly.

- **Outright removal:** if something (a plugin, a trigger, a kill command) just deletes the entity outright rather than damaging it to zero health, OnBreak never fires at all, since there was no "break" event — just a straight removal.

Open the chassis entity's properties and check its **Health** field. If it's a real number above zero, OnBreak is almost certainly your hook (Step 3 below uses it). If Health is 0/blank, or you're not sure how "destroyed" is currently being triggered, track that down first — the rest of this guide assumes OnBreak is firing.

Step 1: List Every Entity in the Vehicle's Cluster

Write down the exact name of every entity that belongs to one vehicle. Using the speeder as the example we've already mapped out:

```
speeder5_ui • speeder5_physbox • speeder5_boost • speeder5_boost_up (and  
every other thruster) • speeder5_X • speeder5_y • speeder5_door •  
speeder5_laser1 • speeder5_spark • speeder5_start_sound •  
speeder5_move_sound • speeder5_stop_sound • speeder5_lasersoundfire
```

You need every one of these listed in the point_template in the next step, or the respawned copy will be missing pieces (e.g. it moves but has no sound, or the door never opens).

Step 2: Create the point_template

1. Place a new entity anywhere on the map (position doesn't matter, it has no physical presence) and set its class to point_template.
2. Open its properties. You'll see a set of numbered fields like **Template 01**, **Template 02**, and so on — one per entity. Set each one to the name of an entity from your Step 1 list, using the name picker (eyedropper) rather than typing it by hand to avoid typos.
3. Repeat until every entity from the cluster has its own Template slot filled in.
4. Under the **Flags** tab, enable "**Preserve entity, if it's already spawned**" (wording varies slightly by FGD version, but it's the flag that keeps the referenced entities alive as the normal, first live copy of the vehicle — without it, they'd vanish the moment the map loads, since point_template's default behavior is to treat its references as templates only, not live entities).

Give the point_template itself a clear name, e.g. speeder5_template.

Step 3: Wire Up the Respawn Sequence

When the vehicle breaks, three things need to happen, roughly in this order: clean up whatever's left, wait a bit, then spawn a fresh copy.

3a — Clean up the old entities

Breaking usually gibs the chassis itself, but the `game_ui`, thrusters, and `phys_keepupright` generally *don't* get removed just because the chassis broke — they'll still be sitting on the map. If you spawn a new copy while those are still alive, you'll get duplicate names fighting each other again — the exact "fly one, they all fly" bug, just self-inflicted this time.

On the chassis's **OnBreak** output, add entries that fire a **Kill** input on every other entity in the cluster that might still be alive (`game_ui`, each thruster, `keepupright`, door, lasers, sound entities). This can get to be a lot of output rows — a tidy alternative is to route them all through one `logic_relay` instead, so **OnBreak** only needs one output (firing that relay), and the relay itself holds all the individual Kill inputs.

3b — Add a respawn delay

Every output in Hammer has a **Delay** field (in seconds). Set the delay on whichever output triggers the respawn (Step 3c) to however long you want the vehicle to stay gone — e.g. 30 or 60 seconds. This is the simplest way to add a cooldown; no separate timer entity needed unless you want something more dynamic (like an announcement partway through the wait).

3c — Fire the respawn

Still on that same **OnBreak** (or the relay it fires), add one more output targeting your `point_template` with input **ForceSpawn**, with the delay from 3b applied. This is what actually recreates every templated entity, fully wired, at its original position.

Example wiring on the chassis's OnBreak output:

```
OnBreak → speeder5_cleanup_relay → Trigger    (delay 0.0)
OnBreak → speeder5_template → ForceSpawn    (delay 30.0)
```

Step 4: Guard Against Double-Triggering

If more than one part of the vehicle can independently fire a break/damage event (for example, several breakable pieces), you risk two respawn sequences overlapping and creating duplicate entities. A simple guard: route the respawn trigger through a `logic_case` or a one-shot toggle (a `math_counter` checked before firing, reset after the new copy spawns) so only the first break event actually triggers a respawn until the cycle completes.

Step 5: Test It

- Compile and load the map. Confirm the vehicle still works completely normally as the first live copy (this confirms the "Preserve" flag is set correctly).
- Destroy it, and confirm the leftover pieces (`game_ui`, thrusters, etc.) actually disappear, not just the chassis.

- Wait out the delay and confirm a fresh copy appears, fully functional — mount it, move it, fire its weapon, everything.
- Destroy the new copy and confirm it can respawn again (not just once).
- If you're running the SourceMod vehicle-attach plugin, also confirm mounting still works correctly on the respawned copy — it should, since the new entities have the exact same names as before, but worth a quick check.

Common Pitfalls

- **Vehicle is missing at map start:** almost always means the "Preserve entity" flag wasn't set on the point_template.
- **OnBreak never fires:** check the chassis's Health field (see Step 0) — if it's not set to break on damage, you'll need a different trigger for "destroyed," not OnBreak.
- **New copy appears, but old broken pieces are still there too:** the Step 3a cleanup didn't kill everything — double check every entity from your Step 1 list has a corresponding Kill input.
- **Respawned vehicle doesn't move/fire/mount correctly:** almost certainly a Template slot was left empty or pointed at the wrong entity in Step 2 — recheck the full list against what's actually filled in.
- **Two vehicles spawn instead of one:** a double-trigger issue — see Step 4.

This is entirely native Hammer/engine functionality — no SourceMod plugin involved anywhere in this process.