Page contents

- Open the level and understand the three views
- Groups, hierarchy, and inherited properties
- Tags connect editor objects to scripts
- Layers make level variants
- Prefabs, instances, and spawnable assets
- What this page does not yet cover
- Sources and provenance

Offline copy of Editor: scenes, hierarchy and prefabs .

Teardown editor: scene organization and reusable assets

Documentation status: In progress. **Source checked:** 2026-10-06. **ModDocs page updated:** 2026-10-06. **Technical basis:** Tuxedo Labs' Teardown modding and editor manual. This page rewrites and explains selected editor workflows; it has not been independently tested in a running copy of Teardown. The quick start shows how to create a first playable local level.

Open the level and understand the three views

Create or select a **Content** mod in the in-game Mod Manager, then choose **Edit** from its Local Files entry. The editor opens that mod's `main.xml`. Use **File** → **Open** to edit a different level XML in the same project. `main.xml` is the entry level used when the player presses **Play** in Mod Manager; additional XML files need an explicit level transition before they appear in play.

The **Scene Explorer** is the level's object tree. Its order follows the XML hierarchy, and selection is synchronized with the 3D view. Right-click an Explorer item and use **New** to add an object. The **Property Window** shows fields for the selected object's type. The editor's hover help is useful when a field has little explanation elsewhere. Explicit property values appear differently from defaults and inherited values; check which

state you are changing before debugging an object that seems to ignore a parent setting.

For a first edit, select an object in the 3D view, find the highlighted Explorer entry, and inspect its properties. Save the level, test it from the editor, then return and change just one property. If the change does not survive a restart, confirm the edited XML belongs to the local mod you actually launched.

Groups, hierarchy, and inherited properties

Create a **Group** with **New** → **Group** in Scene Explorer and move related objects below it. Groups organize crowded scenes, but they also carry properties inherited by child nodes unless a child sets that property explicitly. A parent setting can therefore affect many objects at once. When a child behaves differently, inspect the child's own property value first; an explicit child value overrides what the group supplies.

Hierarchy also affects script searches. A call such as `FindShapes("target")` searches the script's scope below its place in the tree; `FindShapes("target", true)` searches the full scene. The second argument can make a missing tagged shape appear, but it can also match unrelated shapes elsewhere. Prefer placing the script near the objects it controls and choosing specific tags. Use a global search deliberately when the behavior truly spans the whole level. Teardown's API reference defines the function's return value and scope.

Tags connect editor objects to scripts

The `tags` property can attach a name or name/value pair to an object. A tag such as `target` can make a shape discoverable by `FindShape` or `FindShapes`; the value can be read with `GetTagValue`. Tags are attached to an object of a particular type. A shape lookup will not turn a group or body into a shape, even if the visible name looks right. Some tag names have built-in behavior, so inspect the editor's property help before reusing one.

To diagnose an empty lookup, check the tag spelling, the selected object's type, and the script's place in the hierarchy. Then try a global lookup to distinguish a scope problem from a missing tag. Treat the numeric value returned by a `Find*` function as

an object **handle**: it identifies an engine object for later API calls, rather than storing the object itself.

Layers make level variants

Set a group's `layer` property to give that group a named layer. The name appears in the editor's Layers Window. Enabled layers affect what the editor shows and what **File → Play** loads. Use layers to test variants of one level, such as a sandbox arrangement and a mission arrangement, without duplicating the whole XML file. A layer can exclude a group by prefixing the layer name with `-`, which is useful when a shared group should be absent from one variant.

When starting a later level from Lua, `StartLevel` accepts a space-separated layer selection, for example `StartLevel("level2", "MOD/level2.xml", "heist secrets")`. Verify that those exact layer names exist in the target XML. A layer hidden in the editor can look like lost content; check the Layers Window before recreating it. **Ctrl+F** opens the editor's Search Window to find objects by tag or property.

Prefabs, instances, and spawnable assets

A **prefab** is a reusable piece of scene content saved as a separate XML file. Right-click a group in Scene Explorer and convert it to a prefab. Add **instances** when the same asset is needed more than once. If you convert an instance back to a group, edit it, and commit the result, all instances that use that prefab can change. Test every placement after a shared prefab edit.

To put a prefab in the game's spawn menu, add a `spawn.txt` file in the mod root. Each line maps a prefab path relative to that root to a category/name pair:

```
prefabs/mycar.xml : Cars/My car
```

That path must point to an existing prefab XML. The official manual notes that a static object is converted to dynamic when spawned through the menu, so test collisions and motion after exporting. If an asset does not appear, check the relative path, spelling, and whether the group was actually converted to a prefab.

What this page does not yet cover

The editor manual also documents bodies, shapes, lights, triggers, locations, joints, ropes, environmental settings, voxel asset import, and multi-level missions. These need their own detailed ModDocs pages before this editor branch can be called comprehensive. The official editor manual remains the authoritative current source while those sections are written.

Sources and provenance

- Teardown modding and editor manual — Tuxedo Labs; checked 2026-10-06; page version is not separately stated; redistribution license not identified. ModDocs text is independently written from the documented workflows.
- Teardown scripting API — Tuxedo Labs; reference reports API version 2.1.0; checked 2026-10-06; used for `FindShapes`, handles, and `StartLevel` context.