Unfinished Full custom-content integration is not yet verified.

New balls, passives/items and characters: registration research

Updated September 30, 2026. Target: locally installed BALL x PIT 1.301, Unity 6000.0.62f1 IL2CPP, MelonLoader 0.7.3. This is an evidence ledger and experiment plan, **not yet an installation recipe for a new content type**. The modding guide remains usable independently of Ball x Editor.

What is verified in installed metadata

Content path	Declared model and registry	Other declared consumers	What remains unknown
Balls / special balls	`HeroInfo : UpgradeInfo` has `HeroType Type`, icon, colors, specials, hit effects and damage fields. `InfoDB` has `Heroes`, `BaseHeroes` arrays and `HeroOrder`. `BallObj.Init(HeroInst, bool, bool, Vector2, BallSourceType)` accepts a hero instance.	`LevelUpUI.GetHeroItem(HeroType)`, `GetHeroIdxFromSlug(string)`; `BattleSaveData.Heroes`; `MetaSaveData.HeroStats`.	Exact creation/registration timing, whether enums and registry arrays must expand together, offering/merge logic, object behavior, save compatibility and asset creation. Calling something a “ball” here is an inference from `BallObj`/`HeroInfo` relationships, not an official type name.
Passive items	`PassiveInfo : UpgradeInfo` has `PassiveType Type`. `InfoDB` has `Passives` array and `PassiveOrder` list. `UpgradeInfo` supplies slug/name/icon and level/merge data.	`LevelUpUI.GetPassiveItem(PassiveType)`; `BattleSaveData.Passives`; `MetaSaveData.PassiveStats`.	Whether an item is represented by a passive, another upgrade category or a pickup; selection odds, registration, effect implementation, save versioning. “Item” is not yet one verified registration category.
Characters	`CharInfo : SerializedScriptableObject` has `CharType Type`, `PrefabChar`, `PrefabCharSecond`, starter upgrade, stats, icon and slug. `InfoDB` has `Chars` array and `CharHousing`.	`CharSelectItem.Init(CharInfo)`; `CharSelectUI` has an item pool/grid; `MetaSaveData.Chars`, `LastChar`, unlock/worker order.	How the UI enumerates/filters the array, how `CharType` is allocated, prefab dependencies, stat/ability execution, unlock and housing dependencies, save serialization.

`HeroType`, `PassiveType` and `CharType` are declared enums, with 91, 73 and 24 names respectively in this interop export (including any sentinels and non-playable entries). Those counts are **not** verified amounts of usable content. `UpgradeType` declares `kHero`, `kPassive`, `kPet`, `kPetUpgrade`. A new enum number by itself does not prove the game will accept a new entry; code may assume fixed array sizes or switch over known values.

These declarations came from the installed game metadata and the existing `BallXEditor.Diagnostics` API export for identity `1.301:d447e578-124b-4133-a5c9-0aa521fb5621`. They verify that named members exist in generated interop, not the behavior of their native methods. The local Cpp2IL 2022.1.0-pre-release.21 run parsed metadata 31.1 and 117,869 method mappings. Its `dll_il_recovery` output gave `ldnull; throw` stubs for the key methods (`InfoDB.Awake`, `HeroInfo.GetIdx`, `CharSelectUI.Start`), so it cannot establish a registration algorithm. Its ISIL output contains native instruction listings, but most call targets remain raw addresses; no behavior claim is based on an unmapped call.

First live experiment: loaded registry inventory

The separate, read-only ContentInventory probe is installed as `Mods/BallXPIT.ContentInventory.dll`. It captured `InfoDB.I` successfully at the main menu on September 30, 2026, and wrote `ModdingResearch/content-inventory.json` in the installed game directory. It does not patch anything or copy images/prefabs. The export records array positions, type enum names, slugs, `IsInGame`/version flags, character starter-upgrade linkage, prefab/icon presence and registry/order counts.

Loaded registry	Entries	Enum names	Live observations
`Chars`	23	24	All 23 slots are populated; each `CharInfo.Type` matches its array index. The final enum name is `kNum`. All 23 report `IsInGame=true`, have a character prefab and icons, and their starter-upgrade slugs resolve in `Heroes`. `CharHousing` also has 23 positions aligned by index; two positions are null (default and influencer).
`Heroes`	90	91	All 90 slots are populated, have icons, report `IsInGame=true`, and match enum position. The final enum name is `kNum`. `BaseHeroes` has 21 entries, all also present in `Heroes`. `HeroOrder` contains each of the 90 hero slugs exactly once; its order differs from enum/array order.
`Passives`	72	73	All 72 slots are populated and match enum position. The final enum name is `kNum`. `PassiveOrder` contains 71 distinct loaded slugs. The omitted slot is 0 (`kAddHeroDamage`, Spinach), which reports `IsInGame=false` and no icon; the other 71 report `IsInGame=true`.

The exact ordinal alignment and list membership are stronger evidence than metadata declarations alone, but they are **not proof** that a new enum value or larger array will work. We have not verified selection filters, switch branches, save serialization, or the effect and prefab lifecycle. The omitted passive correlates with its unavailable flag, but the rule that produced the order list is still unverified. Do not claim `kNum` is safely extensible until a controlled runtime experiment confirms all consumers.

Second live experiment: upgrade choice to battle roster

The separate ContentFlow probe is installed for the inspected build. It polls `LevelUpUI.I._choices` only while the level-up object is active and records each choice's `UpgradeType` and `UpgradeInfo.Slug`. It also records in-memory `BattleSaveData.I.Heroes/Passives` type names when those rosters change. The output is `ModdingResearch/content-flow.jsonl` in the installed game directory. It uses no patches or writes to game state.

Verified normal level-up traces, September 30, 2026:

Before	Offered choices	After player selection
Balls: `kEarthquake`; passives: none	`kHero:hupg_freeze`, `kPassive:pass_stoneeffigy`, `kHero:hupg_eggsac`	Balls: `kEarthquake,kFreeze`; passives: none
Balls: `kEarthquake,kFreeze`; passives: none	`kHero:hupg_lightning`, `kHero:hupg_poison`, `kPassive:pass_eyeball`	Balls unchanged; passives: `kEyeball`

These verify an existing ball and passive moving from offered `UpgradeChoice.Info` slug to the corresponding in-memory `HeroInst.Type` or `PassiveInst.Type` in `BattleSaveData`. They do not prove which selection method executed, that a `BallObj` was created, that the record was written to disk, or that an added enum identity would work. Fusion remains untested.

Third experiment staged: temporary definition clones

The independent ContentClone probe ran successfully at the main menu on September 30, 2026. It cloned loaded `HeroInfo` (Freeze), `PassiveInfo` (Add Follower Damage) and `CharInfo` (Warrior), assigned test-only names/slugs and enum values, read them back, and destroyed the clones. The ball clone reported numeric type `90` and `GetIdx()=90`; the passive clone reported numeric type `72` and `GetIdx()=72`; the character clone reported numeric type `23`. All three values stringify as `kNum`, the existing trailing sentinel for their enum. Their copied icon/prefab references were present. The game registry counts remained 90/72/23 in the report. This verifies that cloned definitions can be constructed and edited in memory and that `GetIdx()` reflects the assigned type number for the ball/passive clones. It **does not** show that a sentinel or any new numeric ID is accepted by selection, behavior, or saves. No `InfoDB` arrays/order lists were expanded, and no save was touched by the probe.

The v0.3 read-only inventory ran during a normal battle on September 30, 2026. Its in-memory `MetaSaveData` arrays had 23 character entries, 90 hero-stat entries and 72 passive-stat entries, exactly matching the corresponding loaded definition registry lengths. The probe read only lengths,

not save contents, and did not write a save file. This is evidence that adding a new identity must account for a parallel progression-array slot; it does **not** prove the disk format, whether arrays are rebuilt on load, or a safe migration procedure.

The installed interop also declares `SaveMgr.LoadMetaAtPath`, `LoadMetaInSlot`, `SaveMeta`, `SaveBattle` and `SaveDataToPathSafe`, plus `MetaSaveData.HeroStats`, `PassiveStats` and `Chars`. These are method and field declarations only. We have not invoked save methods, intercepted serialization, or altered the player's profile. A real new-ID mod will need an isolated profile and a verified migration/recovery test before the guide can give save-safe installation instructions.

The local Cpp2IL ISIL dump of the inspected `MetaSaveData` constructor contains array-allocation calls with literal lengths `0x17` (23), `0x5A` (90), and `0x48` (72). These match the live character, hero-stat and passive-stat lengths. The raw native instructions support the inference that new progression objects are initialized with fixed build-specific sizes; they do not reveal a safe extension or migration method. The game-derived disassembly stays in a private research backup and is not bundled into this guide.

Fourth experiment staged: registry reference roundtrip

The separate RegistryRoundtrip probe ran successfully at the main menu on September 30, 2026. In one Unity update it replaced the ball/passive `InfoDB` arrays and order lists with expanded copies containing test clones, read the new slots, and restored the original references. Its report recorded `Heroes=91`, `HeroOrder=91`, a new ball slug at index 90 with `GetIdx()=90`; `Passives=73`, `PassiveOrder=72`, a new passive slug at index 72 with `GetIdx()=72`; and `restored=true` with the original sizes 90/90/72/71. The MelonLoader log reported completion without an error. These type numbers coincide with `kNum` sentinel values. This verifies that the registry fields accept and expose expanded copies in memory, but **not** that a new entry can be offered, executed, persisted or reloaded. The probe did not touch progression arrays, display the entries as choices, or save.

The local ISIL/native disassembly of `LevelUpUI.PopulateUpgrades()` contains separate loop comparisons `cmp r13d,0x5A` / `jl` and `cmp r13d,0x48` / `jl` (`0x5A=90`, `0x48=72`). This directly establishes fixed iteration bounds in that inspected method. It strongly suggests that merely expanding the ball/passive registries will not make new indices appear among ordinary level-up choices. We have not mapped every branch in the method or tested a patch that extends the bounds, so the exact registration remedy remains unknown. The disassembly itself stays in private research storage.

The same method's ISIL call map includes `UpgradeMgr.GetHero(HeroType)` followed by `HeroInfo.CanBeUsed()`, and `UpgradeMgr.GetPassive(PassiveType)` followed by `PassiveInfo.CanBeUsed()`. Those signatures are also present in the interop index. This narrows the candidate-enumeration path, but we have not established all of the availability conditions or how final choices are drawn from candidates.

For characters, the separate CharacterFlow probe ran during a normal selection-to-battle transition on September 30, 2026. All 23 `MetaSaveData.Chars` entries had a `CharType` matching `InfoDB.Chars` at the same index. The parallel `InfoDB.CharHousing` array also had 23 positions, but two were null: `kDefault` / `char_default` at index 0 and `kInfluencer` / `char_influencer` at index 22. Housing is therefore not universally present. In this profile 13 of 23 progression entries reported unlocked; that is a player-specific observation, not an intrinsic content count. The visible selection item carried `kShade` / `char_shade`, and the following `BattleSaveData.CurChar.Type` was `kShade`. The installed interop declares `CharSelectItem.TgtInfo`, `CharSelectUI._selectedItem`, `BattleSaveData.CurChar` and `CharBattleInst.Type`; these were the read points. The trace verifies one existing character selection path, not a new-character registration or save migration.

The local native dump of `CharSelectUI.Activate()` also contains a `cmp eax,0x17` / `jge` boundary check (`0x17=23`) before it indexes a character-related array. The surrounding native call targets are not fully mapped, so this is evidence of a fixed bound in the selection path, not proof of exactly which rule rejects a new character. It reinforces the need to test UI creation and selection separately from appending `InfoDB.Chars`.

The ContentFlow v0.2 probe captured normal level-ups on September 30, 2026. In one screen, with `kDark` as the only ball and no passives in the in-memory battle roster, the visible UI showed `pass_thorns`, `pass_irononesie` and `pass_hourglass`. At that instant `_availNewHeroes` held 18 distinct loaded slugs (all belonging to the 21-entry `BaseHeroes` subset), and `_availNewPassives` held 37 distinct loaded slugs. Across 21 distinct captured upgrade screens and 57 displayed choices, **zero displayed slugs remained in their corresponding `_availNewHeroes` or `_availNewPassives` list** at capture time. This repeated result is consistent with the draw removing chosen entries before the UI becomes visible, but the exact mutation sequence is still an inference. Do not describe these lists as complete pre-draw candidate pools. The probe does not inject choices.

What “item” means here

The game metadata distinguishes passive upgrades from world pickups. `UpgradeType` lists `kHero`, `kPassive`, `kPet` and `kPetUpgrade`, while `PickupType` separately enumerates XP, health, bombs, resources, magnet, blueprints and other pickups. `PassiveInst`/`PassiveInfo` and

`PickupInst`/`PickupObj` are different declared paths; `PickupMgr` has methods such as `CreatePickup` and `DropPickup`. This is a declaration-level classification, **not** a verified recipe for a new pickup. A request for a new "item" should first choose whether it means a selectable passive, a pet upgrade, or a collectible dropped in the arena.

Path to a real add-new-content recipe

1. **Confirm the inventory.** Done for the inspected 1.301 main-menu build. Keep the JSON as local research output and remove the probe once further read-only captures are complete. Do not turn failed/null reads into zero-sized-registry claims.
2. **Trace one existing entry end to end.** For a ball, identify one base game's `HeroInfo` in `InfoDB`, then observe its level-up offer, equip/`HeroInst`, `BallObj`, and save/reload. Repeat for one passive, one character, and one dropped pickup if that item category matters. Record exact methods and whether an observed link came from declarations, native call tracing or live gameplay.
3. **Prototype an existing-slot replacement.** In an isolated test copy/profile, change only a reversible display/behavior value and confirm all lifecycle stages. This tests cloning and UI refresh without pretending a new ID is registered. Do not redistribute game assets.
4. **Test a genuinely new identity.** Determine whether the game accepts a new type number and expanded registry/order arrays, or requires patches to consumers. Check selection, unlock, merge/evolution, encyclopedia, save/load, run restart and uninstall. Reject the experiment if it merely overwrites an existing slot.
5. **Publish only a verified recipe.** Include exact game/loader versions, build steps, code, asset loading, install/remove instructions and native test results. Keep hooks and IDs version-gated.

Current status: no independently selectable new ball, passive item or character has been created or verified. The existing runtime's enemy spawn substitution and arena captures do not prove content registration.

Sources / References

Primary evidence: locally inspected BALL x PIT 1.301 interop metadata and research observations recorded in this project in September 2026. This page is a ModDocs interpretation of that evidence; inline external links identify loader and project documentation where used. Unverified behavior is labeled in the text.

- Content registration research and evidence limits
- BALL x PIT modding guide