CH11 — Moves: Custom Moves + Editing + Revert Model

This chapter defines how users create and manage their own moves; how edits apply to defaults vs user-defined moves; and how reverting works without breaking flows, practice sets, or imports.

Doc ID: HZ-V1-CH11_Moves_Custom_Moves_Editing_Revert_Model_R1

Revision: R1 (2026-01-02)

Status: Review

Depends on: See: HZ-V1-CH03 (Core Concepts & Glossary); HZ-V1-CH08 (Entitlements); HZ-V1-CH09 (Default Library); HZ-V1-CH10 (Canonical IDs/Aliases/Families/Variants); HZ-V1-CH29 (Data Storage & Media Rules); HZ-V1-CH30 (Safety/Abuse Limits).

Related: See: HZ-V1-CH12 (Flow Builder); HZ-V1-CH18 (Inbox); HZ-V1-CH19 (Import Conflict Resolution); HZ-V1-CH20–CH22 (Practice); HZ-V1-CH31 (Error States).

Supersedes: N/A (first publication)

Owned Decisions (Locked by CH11)

- Users can create Custom Moves that live in their personal library and can be used in flows.
- Default (shipped) moves are canonical; users cannot edit the canonical record. They can only create a User Overlay (personal customization layer) and can revert it at any time.
- Revert supports per-move revert (single move), per-flow scoped revert (flow-local overrides), and library-wide revert (advanced; gated behind a confirmation ladder).
- Guests cannot save flows and cannot persist customizations; creating/saving custom moves requires an account (see: CH07/CH08).
- No technique descriptions ship by default; user-entered notes are treated as personal content and are never assumed to be “correct” or “universal” (see: CH09/CH10).

Open Questions / Placeholders

- PLACEHOLDER: Custom move count cap per plan • Owner: CH08/CH30 • Options: None / Soft cap / Hard cap • Default: None
- PLACEHOLDER: Which custom move fields are shareable by default (privacy) • Owner: CH17/CH19/CH29 • Options: A) minimal, B) include notes, C) include media links only • Default: A
- PLACEHOLDER: Flow-local override support scope (which fields can be overridden within a flow only) • Owner: CH12/CH13/CH14 • Default: display name + notes only
- PLACEHOLDER: Variant creation UX (quick-create vs full editor) • Owner: CH10/CH11 • Default: quick-create with optional deep editor

1. Definitions and Objects

CH11 uses the move language established in CH03 and the canonical system in CH10. This chapter focuses on ownership, editing scope, and safe revert behavior.

1.1 Move types

- Canonical Move: a shipped move with a canonical ID, optional aliases/family membership, and baseline tags. Canonical records are immutable to users. (See: HZ-V1-CH10.)
- Custom Move: a user-created move record, owned by a user account. May optionally declare it is a variant of another move (canonical or custom).
- User Overlay: a user-owned customization layer applied to a canonical move. Overlays never alter the canonical definition; they only affect how that user sees/uses the move.
- Flow-local Override: optional, scoped customization that only affects one flow node/path. This avoids polluting the move library when a user only wants a special version for a single plan.

1.2 Editing scope hierarchy

When rendering a move inside the app, we resolve the “effective” view of the move using the following precedence:

- Flow-local Override (if present for this node)
- User Overlay (if present for this canonical move)
- Base Move (canonical or custom)

This hierarchy is the core safety net that prevents accidental global changes and makes “revert” deterministic.

2. Data Model (Implementation-Agnostic)

This section defines the fields and relationships without prescribing a backend. Replit should treat this as the product contract; storage can be Firebase (current direction) or equivalent. (See: CH29 for media/storage rules.)

```
Entity: Move (base record; canonical or custom)
- id: string (canonical IDs for shipped moves; UUID for user moves)
- owner_user_id: string | null // null means canonical
- is_canonical: boolean // true for shipped moves
- name: string // canonical display name or user-chosen name
- primary_art_tags: string[] // e.g., ["boxing","muay_thai"]
- families: string[] // family IDs; see CH10
- aliases: string[] // optional; see CH10
- variant_of_move_id: string | null // points to base move (canonical or custom)
- tags: string[] // generic tags (strike/defense/footwork/body-part/etc.)
- created_at, updated_at

Entity: MoveOverlay (user customization for a canonical move)
- user_id: string
- canonical_move_id: string
```

```
- custom_display_name: string | null
- hidden: boolean
- custom_tags: string[] | null
- custom_notes: string | null
- custom_media: (see CH29; likely links only) | null
- created_at, updated_at

Entity: FlowNodeMoveRef (how flows reference moves)
- flow_id: string
- node_id: string
- move_id: string // references Move.id (canonical or custom)
- local_override: { display_name?, notes?, tags?, media? } | null
- resolved_display_name_cache: string (optional performance cache; never
authoritative)
```

3. Create Custom Move (UX + Logic)

3.1 Entry points

- From Moves Library: “+ New Move”
- From Flow Builder move picker: “Create new move”
- From Import resolution (CH19): “Create missing move”

3.2 Minimum fields (fast path)

- Required: Move Name
- Optional on first save: Art tags, tags, family, variant-of, notes, media

  Rationale: remove friction; users can get to building flows immediately.

3.3 Progressive disclosure (power-user depth)

- Section A — Identity: name, aliases (optional), language/labeling helpers
- Section B — Classification: art tags, strike/defense/footwork categories, body part tags
- Section C — Relationships: family membership, variant-of linkage
- Section D — Personal notes/media: user notes; media handling per CH29 (links shareable; uploads private-only)

3.4 Variant creation (quick-create)

Users asked for fast creation of variants (e.g., check hook as a variant of hook). V1 supports a quick-create pattern:

- User selects base move (canonical or custom) -> taps “Create Variant”
- Prompt: Variant name (required) + “What’s different?” (optional short note)
- System stores: variant_of_move_id = base move id
- UI: variant badge shows parent: “Variant of: Hook” and allows jump-to-parent

If the user wants deeper customization, they can open the full edit screen later.

4. Editing Moves

Editing must be safe, predictable, and reversible. V1 separates editing into three distinct actions depending on what the user is editing.

4.1 Editing a custom move (owned record)

- Edits update the custom move record directly.
- All flows referencing the custom move reflect updated display name/tags automatically (unless a node has a flow-local override).
- If user changes the move name, the app must preserve searchability by keeping prior names in the move's alias list (optional suggestion; placeholder if you want).

4.2 Customizing a canonical move (overlay)

Canonical moves cannot be edited directly. Instead, the user creates/edits a MoveOverlay.

- Entry: move detail screen shows "Customize" for canonical moves.
- Overlay fields supported in V1: custom display name, hidden toggle, custom tags, custom notes, and optional custom media links (per CH29).
- Overlay never alters the canonical move's ID, family placement, or canonical aliases.

Important: This preserves global consistency for imports and sharing while allowing personal nuance.

4.3 Flow-local overrides (node-scoped)

Flow-local overrides exist for cases like: "Jab" but in this plan it means "pawing jab" with special notes. The override lives on the node, not in the move library.

- Entry: on a node, user taps "Node Details" -> "Override Move Details for this node only."
- Supported override fields in V1: display name override + node notes override (and optionally tags).
- UI must clearly indicate that an override is active (badge: "Local").
- User can remove the override ("Revert node to move defaults").

5. Revert Model

Revert is required to maintain trust. Users should be able to undo customizations without fear of breaking flows or imports.

5.1 Revert actions

- Revert Overlay: deletes/clears the user overlay for a canonical move; the user returns to canonical defaults immediately.
- Revert Local Override: clears the node's local_override field; node returns to overlay/base resolution.
- Revert Custom Move Edit: not automatic; provide "Undo" toast for recent edits and optionally a version history placeholder for later.

5.2 Revert UX requirements

- Every customization screen has a visible “Revert” action.
- Revert requires confirmation only if it would remove substantial user-entered content (notes/media). Use warning ladder style (see: CH30).
- Revert copy must be explicit about scope: “Revert this move in your library” vs “Revert only this node.”

5.3 Safety rails to prevent trust breaks

- Provide an easy way to downgrade mastery-related labeling if the user feels the app overestimates them (owned by CH23, but surface entry points here).
- Any time the app applies an overlay or local override in a view, show an info affordance: “Why does this move look different?” -> explains resolution order (Section 1.2).

6. Deletion & Tombstones

Users may delete custom moves. This must not corrupt flows that reference them.

- If a custom move is referenced by at least one flow node, deletion must show options: A) Cancel, B) Replace references, C) Delete anyway (creates tombstone).
- Tombstone behavior: the node continues to render with label “Deleted Move” and retains any flow-local override text if present.
- Replace references flow: user selects a replacement move; all nodes referencing deleted move are updated.

7. Guest & Account Rules (V1)

- Guests can browse demo content but cannot save flows or persist custom moves (global lock).
- If a guest attempts to create a move or save a flow, show a conversion gate: “Create an account to save your library and access it on any device.” (See: CH07/CH08 for exact placements.)
- If guest begins editing, store a temporary in-memory draft only until app close; no offline persistence.

8. Imports & Sharing Interactions (CH17–CH19)

CH11 defines how move ownership behaves during imports; detailed conflict rules live in CH19.

- When importing a flow that references a move the receiver lacks, the receiver must be able to create a missing move quickly (Section 3).
- If the imported flow includes custom overlays/notes/media on a canonical move, the receiver chooses scope: keep only inside that imported flow (node local overrides) vs add as overlay in their library (MoveOverlay).
- Uploads are private-only and not shareable; if a sender used uploads, receiver sees a badge: “Media not shared (private upload).” (See: CH29.)

9. Edge Cases

- Two moves that look similar but are canonically separate (e.g., Teep vs Push Kick): both remain available. Users can hide one via overlay if desired (hidden=true). (See: CH10.)
- If user renames a canonical move via overlay to an alias that already exists as a separate canonical move, show a warning: “This name is also a separate move in your library.” Do not block; just inform.
- If a user creates a variant chain (A -> B -> C), UI must show the parent chain in the move detail (breadcrumb). Cap display depth to avoid clutter (placeholder).
- If a move is hidden but used in an existing flow, it still renders in that flow; hidden only affects pickers/search lists.

10. Accessibility & Localization Readiness

- All move names and tags must be screen-reader readable; avoid icon-only buttons without labels.
- Revert confirmations must be fully navigable with VoiceOver.
- Do not bake hard-coded martial arts terminology into button labels; keep copy in strings for future localization (see CH32).

Acceptance Test Checklist (CH11)

Write tests in Given/When/Then format. These are the minimum for V1 correctness.

- Given a signed-in user, when they create a custom move with only a name and save, then the move appears in their move library and is selectable in the flow builder picker.
- Given a canonical move, when the user taps Customize and changes display name, then the canonical record is unchanged and only that user sees the customized name in pickers and nodes (unless a node override exists).
- Given a node with a local override, when the user removes the override, then the node immediately resolves to the overlay/base move display using the hierarchy in §1.2.
- Given a custom move referenced by a flow, when the user attempts to delete it, then the app blocks silent deletion and offers Replace / Cancel / Delete anyway (tombstone).
- Given an imported flow containing custom notes on a canonical move, when the receiver chooses 'Flow-only', then notes are stored as node local overrides and do not create/modify overlays in the receiver's library.
- Given a hidden move, when the user opens an existing flow that references it, then the move still renders in the flow and practice selections still include the path.
- Given a guest user, when they attempt to save a custom move or save a flow, then the app shows a create-account gate and does not persist the content after app close.

Checklist

- Custom move creation works from all entry points (library, picker, import).
- Canonical customization uses overlays; no canonical edits exist in UI.
- Flow-local overrides are visibly indicated and reversible.
- Delete/replace/tombstone flow works and never breaks rendering.
- Import handoff to CH19 is implemented (scope selection).
- Guest restrictions enforced (no saving).

Replit Build Prompt (CH11 only)

```
You are implementing Handz V1, Chapter CH11 (Moves: Custom Moves + Editing +
Revert Model) for iOS portrait.
Inputs attached: CH00 (Manifest) and CH11 (this document). Implement ONLY CH11.
Treat cross-references as dependencies.
If you must assume anything, write it in a 'PRD Assumptions' block comment and
stop before shipping that part.

Build the following, end-to-end:
1) Data structures for Move (canonical + custom), MoveOverlay (user customization
for canonical), and FlowNodeMoveRef local overrides.
2) UI:
- Moves Library: create custom move, open move detail.
- Move Detail: if canonical show 'Customize'; if custom show 'Edit'.
```

```
- Customize canonical: edit overlay fields (display name, hidden, tags, notes,
media links if supported).
- Node Details: add/remove local override (display name + notes at minimum).
- Delete custom move: if referenced, show Replace/Cancel/Delete (tombstone).
3) Resolver logic: effective move display is local override -> overlay -> base.
4) Guest rules: disable saving moves/flows; show conversion gate.

Acceptance criteria:
- All Given/When/Then tests in CH11 pass.
- No canonical move record is mutated by user actions.
- Revert actions are deterministic and clearly scoped in UI.
```

Troubleshooting Notes (CH11)

- If a move name appears inconsistent across screens, verify resolver order (§1.2) and confirm node overrides are not being mistakenly applied globally.
- If imports are overwriting user libraries unexpectedly, ensure 'Flow-only' scope stores data in node local overrides, not overlays (handoff to CH19).
- If deleting custom moves breaks flow rendering, implement tombstones: keep node references but render a safe placeholder label.
- If hidden moves disappear from existing flows, fix picker filtering vs renderer: hidden affects search/pickers only, not flow rendering.
- If guests can still persist data, audit storage writes: block writes at the domain/service layer, not only the UI layer.