Handz V1 PRD Bundle

CH10 — Moves: Canonical IDs, Aliases, Families, Variants

Document purpose: define the identity system for moves so default moves, user-created moves, flow nodes, sharing/imports, and analytics remain consistent even when names differ across arts, gyms, and personal style.

Doc ID:	HZ-V1-CH10_Moves_Canonical_IDs_Aliases_Families_Variants_R1
Revision:	R1 (2026-01-02)
Status:	Draft
Depends on:	CH00 (Manifest), CH03 (Core Concepts & Glossary), CH09 (Default Library System), CH08 (Entitlements & Plan States)
Related:	CH11 (Custom Moves + Editing + Revert), CH19 (Import Conflict Resolution), CH05 (Screen Inventory), CH13 (Flow Builder Node Types & Data Model), CH15 (Library Search/Sort)
Supersedes:	—
Owned Decisions:	Move identity model (canonical key, move_id), alias groups & user-preferred labels, move family taxonomy + variant-of relationship semantics.
Open Questions / Placeholders:	Finalize default move list (Owner CH09). Finalize which alias groups ship in V1 (Owner CH09/CH10). Decide if 'variant-of' is exposed during V1 onboarding or only in custom move creation (Owner CH10).

1. Scope

This chapter defines how Handz represents a “move” across different naming conventions, disciplines, and user customization, without breaking flows, practice history, or imports when names change.

In V1, this chapter is about WHAT the app does and HOW the move identity system behaves. Implementation choices (database vendor, etc.) belong to later build prompts and CH29 (storage limits) / CH28 (offline/sync).

In-scope

- Canonical identity for every move (default and user-created).
- Alias model: multiple names that can map to the same canonical concept (where appropriate).
- Family model: grouping related moves for browsing and onboarding.
- Variant model: optional parent-child relationship to represent “a specific version of a move.”
- Rules for display names, user preferences, and search behavior.
- Rules for imports and conflict resolution inputs owned by this chapter (the full flow is owned by CH19).

Out-of-scope (owned by other chapters)

- Exact default move list contents and counts (CH09).
- Custom move editor UX and revert UX beyond identity fields (CH11).
- Flow builder node UI and per-node annotations (CH12/CH13/CH14).
- Sharing link lifecycle, inbox behaviors, and caps (CH17/CH18).
- Scientific claims pages and copy (CH26).

2. Goals & non-goals

2.1 Goals

- Zero friction for the flow builder: users should be able to build flows even if they disagree on terminology.
- Flexibility: users can use default move concepts immediately, but can specialize later without breaking old flows.
- Consistency: renaming a move should not break search, practice logs, imports, or existing flows.
- Diversity across striking: default concepts should be broadly usable across major striking arts, with room for personal nuance.
- Import safety: imported flows should remain faithful to the sender’s intent, while giving the receiver control over how details are absorbed into their library.

2.2 Non-goals

- Handz does not claim a single “correct” technique description for any move in V1; default moves ship with tags and families but no technique cues/descriptions (see CH09 lock).
- Handz does not attempt to resolve all martial arts terminology disputes automatically. We provide structure so users can decide their own naming and nuance.

3. Core concepts (CH10-owned)

These concepts extend CH03’s glossary and are referenced across the bundle.

3.1 Move record types

- Default (Shipped) Move
  - Provided by the app in the default library.
  - Has a stable canonical identity (canonical_key) and a stable record id (move_id).
  - User may customize personal fields (notes, media, tags) only if allowed by entitlement; the canonical identity remains unchanged.
- User Move (Custom)
  - Created by a user (non-guest) via CH11.
  - Has a stable move_id and a user-owned identity; may optionally declare variant-of or family membership.
- Flow-local Move Snapshot (Import only)
  - Exists only inside an imported flow when the receiver chooses not to add it to their library.
  - Still carries enough identity metadata to display consistently inside that flow.
  - Owned by CH19, but the identity fields in §6.3 are owned by CH10.

3.2 Identity fields

Every move record contains:

- **move_id**: immutable UUID for this move record (primary key).
- **canonical_key** (default moves only): immutable stable string key for shipped concepts (e.g., STRIKE.PUNCH.JAB).
- **display_name**: what the user sees in their UI for this move (can be user-preferred label for default moves).
- **alias_group_id** (optional): points to a synonym/alias set (see §4).
- **family_ids**: list of families this move belongs to (see §5).
- **variant_of_move_id** (optional): parent move reference (see §5.4).
- **origin**: enum {default, user, imported_snapshot}.

Why we need canonical identity

- Names are unstable: “front kick,” “teep,” and “push kick” may be used differently by different gyms.
- Users rename moves over time as they learn more or cross-train.
- Imports must map consistently even when receiver uses different labels.
- Analytics needs stable concepts (e.g., “how many flows use some form of jab”) without depending on user spelling.

3.3 Display rules (what users see)

- In the UI, the primary label shown is always **display_name**.
- For default moves, display_name is initially the default label shipped by Handz, but may be overridden by the user’s preferred label settings (see §4.4).
- For custom moves, display_name is user-chosen at creation and can be edited later (CH11).
- When a move is shown in a context where ambiguity matters (imports, conflict resolution), show a secondary line: “Family: ” and “Also known as: ” where available.

4. Alias system

Aliases let different strings refer to a shared concept when they truly are synonyms in most contexts, while still allowing separate canonicals when nuance is meaningful.

4.1 Alias group definition

An **alias group** is a set of labels that map to a single canonical concept.

- Each alias group has: alias_group_id, canonical_key (or canonical move_id), labels[], and a recommended default_label.
- Default moves may participate in alias groups, but only when Handz decides they are synonyms for V1.
- Alias groups are not used to force agreement; they are used to reduce friction and improve search.

4.2 When NOT to use alias groups

- When users commonly treat two terms as different moves with different intent or execution.
- When the distinction matters to planning (e.g., a term implies a different tactic).
- When the community is split and Handz wants to stay neutral.

V1 lock example: **Teep** and **Push Kick** are separate canonical moves (not aliases) per product decision; they may still belong to the same family (see §5.3).

4.3 Search behavior with aliases

- Search matches both display_name and all alias labels for moves in the library.
- If user searches an alias label, results show the move using their preferred display_name.
- Search results show a subtle hint line: “Matched: ” only when the displayed name differs from the query.

4.4 User preferred labels (simple V1)

V1 should keep this simple: users can choose how certain common concepts are labeled in their UI, but we avoid complex prompts or “add both” options.

- A “Naming Preferences” screen exists (owned by CH07/CH08 onboarding flow; identity behavior owned here).
- Users select a preferred label for each eligible alias group (single choice).
- Users can change preferences later in Settings (owned by CH04/CH06/CH07; behavior here).
- If a user never touches naming preferences, defaults remain.

4.5 Guest behavior

- Guest users cannot personalize naming preferences; they see default shipped labels only.
- If a guest converts to an account, they are offered a one-time “Set your naming preferences” step (owned by CH07), without forcing repetition.

5. Families & variants

Families group related moves for onboarding, browsing, and search, without forcing synonym decisions. Variants allow a user to express that a custom move is a specific form of another move.

5.1 Family definition

- A **family** is a taxonomy node: e.g., Punches → Straight Punches → Jab.
- A move can belong to multiple families (e.g., “Jab” belongs to “Straight Punches” and “Boxing Essentials”).
- Families can be nested (tree), but moves can be tagged into multiple leaves.

5.2 Why families exist (V1)

- Onboarding: let users browse “essentials” by discipline and by body part, without needing to understand every move name.
- Library browsing: faster selection in the flow builder move picker.
- Search fallback: if user doesn’t know the exact name, they can drill down by category.

5.3 Families vs aliases: how they interact

- Aliases collapse names into one concept; families group multiple concepts together.
- Two separate canonicals can be in the same family if they’re closely related.
- Example: Teep and Push Kick are separate canonicals, but may both live under a family like “Front Linear Kicks” or “Distance Kicks.”

5.4 Variants (optional but supported)

Variants are a lightweight relationship: “this move is a specific version of that move.” They do not change core behavior; they improve organization and clarity.

- A move may optionally set **variant_of_move_id** to another move.
- Variant-of can apply to default or user moves, but V1 primarily exposes it for custom moves.
- Variant-of is not required; users can ignore it and still build flows.
- Variant-of is visible in the move detail header: “Variant of: Hook” (tap to open parent).

  Example: “Check Hook” can be modeled as a variant-of “Hook.” This helps users find it even if they search for Hook.

5.5 Handling lead/rear and stance nuance (without bloating defaults)

Because lead vs rear execution differs and users have different preferences, V1 should not explode the default library into every permutation.

- Default moves stay neutral (e.g., “Front Kick”).
- Users can express specificity in three ways:
  - Create a custom move (e.g., “Lead-Leg Front Kick”) optionally marked as a variant-of “Front Kick.”
  - Annotate the move instance inside a flow via node metadata or sequence details (owned by CH13/CH14).

    - Use families/filters to pick the right concept quickly (e.g., “Kicks → Front Kicks”).
- When importing a flow, the sender’s exact move names and any variant-of relationships are preserved as metadata; the receiver chooses how to map them (CH19).

6. Data model requirements (behavioral spec)

This section defines the minimum data Handz must preserve to make identity robust. Exact storage tech is not locked here.

6.1 Default move schema (conceptual)

Default moves must include immutable canonical identity and a label set:

```
MoveDefault {
move_id: UUID (immutable)
canonical_key: string (immutable)
default_label: string
alias_group_id?: string
family_ids: string[]
created_by: 'system'
is_deletable: false
}
```

6.2 User move schema (conceptual)

```
MoveUser {
move_id: UUID (immutable)
display_name: string (editable)
origin: 'user'
variant_of_move_id?: UUID
family_ids: string[] (editable)
notes?: string
media?: (link or upload reference; governed by CH29)
}
```

6.3 Import snapshot schema (conceptual, identity fields only)

When a receiver views an imported flow without saving moves to their library, the flow still needs stable identity data so it renders consistently.

```
MoveSnapshot {
snapshot_id: UUID
suggested_canonical_key?: string
suggested_alias_group_id?: string
sender_display_name: string
sender_family_hints?: string[]
sender_variant_of_hint?: string
}
```

6.4 Renaming rules

- Renaming a move updates display_name only; move_id remains unchanged.
- Any flow nodes referencing that move_id automatically show the new display_name.
- Practice history records reference move_id (or path IDs that resolve to move_id), so history remains intact across renames.
- Default move canonical_key is never altered; user label preferences affect display only.

7. UX behaviors owned by CH10

This chapter does not own the full screen inventory (CH05), but it owns the behavior of identity-related UI elements wherever they appear.

7.1 Move picker behavior (flow builder)

- Move picker lists moves using display_name, grouped by recent, favorites (if any), and families.
- Search matches: display_name + alias labels + family names.
- A small secondary badge may show origin: “Default” vs “Custom” only in advanced view; default view stays minimal.

7.2 Move detail header (anywhere a move is inspected)

- Header shows display_name.
- If variant_of_move_id exists: show “Variant of: ”.
- If move is default and part of an alias group: show “Also known as: …” collapsed line.
- No technique description is shown by default in V1 (CH09 lock), but user notes are allowed for account users (CH11).

7.3 Naming preferences (Settings)

- List only the alias groups Handz chooses to expose in V1; avoid overwhelming users.
- Each item is a simple single-select radio list (no multi-add).
- A preview line shows how move labels will appear in the move picker and on nodes.
- Changes apply instantly across the UI.

8. Imports & conflict resolution inputs (CH10-owned fields)

CH19 owns the full import conflict resolution flow. CH10 defines the identity inputs and expected outcomes so imports remain consistent.

8.1 The three receiver actions

- **Map to existing**: receiver selects an existing move_id in their library (default or custom) that represents the sender’s move.
- **Add as new**: receiver creates a new MoveUser entry using sender_display_name and optional hints (family, variant-of).
- **Keep flow-local**: receiver keeps the sender move as a snapshot inside that imported flow only; it does not appear in global move picker.

8.2 What must be preserved from sender

- Sender display name for each move used in the flow.
- Any sender custom metadata that is allowed to travel with the share (owned by CH17/CH19).
- Family hints and variant-of hints when present, to help the receiver decide.
- If media is a link: link can be shared; if media is an upload: it is private-only and must not be shared (CH29 lock). The import UI must communicate this clearly.

8.3 Mapping rules (identity)

- If sender references a default move canonical_key that exists in the receiver’s default library, the import flow should propose auto-mapping to that canonical move_id.
- If sender uses a label that matches an alias group label, propose mapping to the canonical move for that group (unless excluded by V1 decisions).
- Receiver always has final control and can override mapping.

9. Acceptance tests (Given/When/Then)

- Given a default move with canonical_key STRIKE.PUNCH.JAB, when the user changes their preferred label for its alias group (if applicable), then all instances of that move across the app update their display_name immediately without changing move_id.
- Given a custom move “Lead-Leg Front Kick” marked as variant-of “Front Kick,” when the user opens the move detail, then the header shows “Variant of: Front Kick” and tapping it opens the parent move.
- Given a flow node referencing move_id X, when the user renames the move record X, then the node label updates everywhere and practice history continues to reference the same move_id.
- Given the user searches for an alias label that differs from their display_name, when results are shown, then the move appears under their display_name and a subtle “Matched: ” hint appears.
- Given a free user receives an import with missing moves, when the import conflict screen loads, then it offers (Map / Add New / Keep Flow-Local) for each unmapped move and explains that uploads cannot transfer (links can).

Checklist

- Default moves have immutable canonical_key and move_id.
- Renaming never changes move_id.
- Search matches aliases and families.
- Variant-of relationship is optional and correctly displayed.
- Import mapping supports Map/Add/Flow-local outcomes.

10. Replit build prompt for CH10 (step-by-step)

Use this prompt together with CH00 and this CH10 PDF when implementing. Do not implement other chapters unless explicitly attached.

```
You are implementing Handz V1 PRD Bundle (HZ-V1). Follow CH00 rules strictly.
Implement ONLY CH10 (Moves: Canonical IDs, Aliases, Families, Variants). If you must assume
something, write it in a PRD Assumptions block and stop that feature.

Build tasks:
1) Create data models for MoveDefault and MoveUser with immutable move_id and (for defaults)
canonical_key.
2) Implement alias groups:
- alias_group table/object with labels[], default_label
- search matching against labels
- user naming preferences: per-alias-group preferred_label stored per-user
3) Implement families taxonomy:
- family nodes, nested structure
- moves can belong to multiple families
4) Implement variant-of:
- optional variant_of_move_id
- move detail header shows variant-of when present
5) Implement search rules:
- match display_name, alias labels, family names
- if query matches alias label not equal to display_name, show a subtle “Matched: …” hint
6) Implement import identity support interfaces (stubs if CH19 not implemented):
- mapping outcomes Map / Add New / Keep Flow-Local for a move snapshot
- preserve sender_display_name and hints
7) Add unit tests for identity invariants:
- rename does not change move_id
- preferred label changes update display across UI without changing identity
- search finds moves by alias and family

Do not add technique descriptions to default moves (V1 decision). Ensure teep and push kick
are separate default canonicals.
```

11. Troubleshooting notes for CH10

- Symptom: Flow nodes show wrong move names after a rename
  - Verify nodes store move_id, not display_name strings.
  - Verify UI resolves move_id → display_name at render time.
  - Check caching: if you memoize node labels, ensure invalidation on move updates.
- Symptom: Search misses obvious synonyms
  - Verify alias labels are included in the search index.
  - Verify case-folding and punctuation normalization (e.g., '1-2' vs 'one-two').
  - Confirm the term should be an alias group; if it’s a nuance distinction, keep separate canonicals.
- Symptom: Import mapping keeps creating duplicates
  - Ensure auto-suggest mapping uses canonical_key first when available.
  - Show receiver a clear mapping UI (Map/Add/Flow-local) and default to Map when high confidence.
  - Record mapping decisions so repeated imports learn the user’s preference.

- Symptom: Variant-of creates confusing hierarchy
  - Keep variant-of optional and primarily for user-created moves.
  - Do not auto-create variants; only user-initiated.
  - In UI, show variant-of as a small secondary line, not a primary navigation unless tapped.