CH13 — Flow Builder: Node Types & Data Model

This chapter defines the canonical graph entities used by the Flow Builder (nodes, edges, transitions/sequence details, metadata, validation rules, and storage shapes). It is written to remove guesswork for implementation and to remain stable as other chapters evolve.

Key concept: the Flow Builder is a **directed decision graph**. A “move” node can branch into up to 10 responses. Multiple branches can later merge back into a shared continuation. “Sequence details” between two moves are optional and act as a transition payload.

Doc ID	HZ-V1-CH13_Flow_Builder_Node_Types_And_Data_Model_R1
Revision	R1 (2026-01-02)
Status	Draft
Depends on	CH03 (Core Concepts & Glossary), CH12 (Flow Builder Interaction Model), CH29 (Data Storage & Limits)
Related	CH14 (Sequence Detail Editor), CH19 (Import Conflict Resolution), CH20–CH22 (Practice Mode), CH16 (Flow Detail View)
Supersedes	—
Owned Decisions	Canonical graph entity shapes; validation rules; cycle policy (disallow cycles); “sequence details” stored on edges as a transition payload; placeholder branch endpoints allowed in Draft state.
Open Questions / Placeholders	PLACEHOLDER: Max nodes/edges per flow per plan • Owner: CH30/CH29. PLACEHOLDER: Whether to allow “soft cycles” for drill loops • Owner: CH20/CH21. PLACEHOLDER: Which edge-label presets ship by default • Owner: CH10/CH13.

Cross-reference rule: when a behavior is owned by another chapter, this chapter only references it. Interaction behaviors (pan/zoom, reorder, add branch, etc.) are owned by CH12. See: HZ-V1-CH12.

1. Scope and Vocabulary

This chapter defines data and types for:

- Nodes: Move Nodes (primary), plus optional Placeholder Nodes used only in Draft/incomplete graphs.
- Edges: Connections between nodes; can be linear or conditional (branch).
- Transition payload (“Sequence details”): Optional metadata attached to an edge, edited in CH14.
- Flow Graph container: A single flow’s graph, layout coordinates, and per-flow metadata.

Terms (owned by CH03): move, alias, family, variant, flow, path, branch, merge, sequence, gameplan, mastery, maintenance. See: HZ-V1-CH03.

2. Entity Overview

Entity	Purpose
Flow	Document/container for one user-created (or demo) flow. Owns graph, layout, metadata, and revision history.
Node	A vertex in the directed graph. In V1, the only content node type is a Move Node; Placeholder Nodes support drafts.
Edge	A directed connection from one node to another. Can include a branch condition label and an optional Transition payload.
Transition (Sequence Details)	Optional structured detail describing how to move from source move to target move. Stored on the edge; edited via a modal/editor (CH14).
Viewport/Layout	Graph canvas layout data: node positions, optional viewport transform, and local UI preferences.

3. Graph Rules and Validation (No Guesswork)

3.1 Directed graph characteristics

- **Directed:** every edge is from a source node to a target node.
- **Branching:** a Move Node may have 0–10 outgoing edges (max 10).
- **Merging:** a Move Node may have 0–N incoming edges (unlimited).
- **Dangling paths allowed:** leaf nodes (0 outgoing edges) are valid. Additionally, Draft flows may contain Placeholder Nodes representing “Add Response” endpoints.
- **Root is replaceable:** the first node is not special in data; a flow stores *rootNodeId* for convenience, but any node can become root by setting rootNodeId (CH12 owns the UI behavior).

3.2 Cycle policy

V1 disallows cycles (no node may be reachable from itself through directed edges). This prevents infinite practice loops and simplifies path enumeration for Practice Mode.

If a future version wants “drill loops,” that decision is owned by Practice chapters. Placeholder: allow “soft cycles” only inside Practice playlists, not inside the saved graph.

3.3 Edge uniqueness + self-loop rules

- Self-loops are forbidden: sourceNodeId != targetNodeId.
- Duplicate edges are allowed only if their *conditionLabel* differs. If both label and endpoints match, treat as duplicate and block creation.
- Edge IDs are globally unique within a flow. Implementations may use UUIDs.

3.4 Max outgoing branches

A Move Node can have at most 10 outgoing edges. Attempting to add an 11th triggers the CH30 Warning Ladder (soft caps/upgrade prompts). See: HZ-V1-CH30.

4. Canonical Schemas

All schemas below are normative. When importing/exporting, preserve unknown fields to support forward compatibility.

4.1 Flow (Graph Container)

Field	Type	Req	Description
id	string (uuid)	Y	Unique flow ID.
ownerUserId	string	Y	User who owns the flow.
title	string	Y	User-facing name. Can be empty during Draft; enforce non-empty on publish/share if needed.
status	enum	Y	draft | saved | archived. (Shareability owned by CH17.)
rootNodeId	string (uuid)	Y	Entry node ID for path enumeration and practice. UI allows replacing root (CH12).
nodeIds	array	Y	List of node IDs in this flow (for quick indexing).
edgeIds	array	Y	List of edge IDs in this flow.
layout	object	Y	Canvas layout and viewport info (see 4.4).
tags	array	N	User tags for search/filter (owned by CH15).
createdAt	timestamp	Y	Creation time.
updatedAt	timestamp	Y	Last modified time.
revision	int	Y	Monotonic increment when saved; helps conflict resolution (CH28).
isDemo	bool	N	True for built-in demo flows; does not count toward caps (CH08/CH25).

Storage note: in Firestore, Flow is typically a document under *flows/{flowId}* with nodes/edges as subcollections (recommended) or embedded arrays (only for small graphs). See 6.1.

4.2 Node

Field	Type	Req	Description
id	string (uuid)	Y	Unique node ID.

Field	Type	Req	Description
type	enum	Y	move | placeholder. (V1 ships only these two.)
moveRef	object	Cond	Required when type=move. See 4.2.1.
placeholderKind	enum	Cond	Required when type=placeholder: add_response | add_branch_stub.
ui	object	Y	UI/layout fields (position, size). See 4.4.
meta	object	N	Optional metadata: notes, createdAt/updatedAt overrides, provenance (import).

4.2.1 MoveRef (inside Move Node)

Field	Type	Req	Description
canonicalMoveId	string	Y	Stable canonical ID for the move concept (owned by CH10).
displayName	string	Y	Rendered name in this flow node (can differ from canonical name).
userMoveId	string	N	If user has a customized move entry, reference it here; otherwise null to use canonical defaults.
stanceContext	enum	N	orthodox | southpaw | switch | any | unset (optional; used for display/filtering only in V1).
tags	array	N	Node-local tags (rare; defaults to none).

4.3 Edge

Field	Type	Req	Description
id	string (uuid)	Y	Unique edge ID.
sourceNodeId	string (uuid)	Y	From node.
targetNodeId	string (uuid)	Y	To node.
kind	enum	Y	linear | conditional. Conditional edges represent opponent-response branches.
conditionLabel	string	Cond	Required when kind=conditional. Example: “leans back”, “shells up”.
order	int	N	Optional ordering hint for rendering when multiple outgoing edges exist (CH12 owns layout).
transition	object	N	Optional Sequence Details payload (see 4.3.1).

Field	Type	Req	Description
ui	object	N	Optional UI style hints (label position, curvature).
meta	object	N	Optional metadata/provenance for imports/exports.

4.3.1 Transition (Sequence Details payload)

Field	Type	Req	Description
id	string	Y	Unique within the edge (can reuse edgeId or nested uuid).
isEnabled	bool	Y	True if transition details exist and should be shown in the Sequence Editor.
fighter	object	N	Fighter transition fields (footwork, upper-body cues, etc.). V1 stores structured lists + freeform notes.
opponent	object	N	Opponent position/behavior notes relevant to this transition.
notes	string	N	Freeform transition notes.
attachments	object	N	Video link refs if supported (CH29 determines link vs upload).
updatedAt	timestamp	N	Last edit time (for conflict resolution).

Editing UI for Transition is owned by CH14. This chapter only defines the data shape.

4.4 Layout / Viewport

Field	Type	Req	Description
nodes	map	Y	Map from nodeId to UI fields; used to persist canvas layout.
viewport	object	N	Optional: {x, y, zoom}. Persist for “resume where you left off.”
layoutVersion	int	Y	Bump if layout algorithm changes.

NodeUI (referenced above):

Field	Type	Req	Description
x	number	Y	Node x coordinate in canvas space.
y	number	Y	Node y coordinate in canvas space.
w	number	N	Optional width (if resizable nodes ship later).
h	number	N	Optional height.

Field	Type	Req	Description
collapsed	bool	N	Optional for Summary/Collapse features (owned by CH12/CH13).

5. Optional Node Types and Draft States

5.1 Placeholder Nodes (Draft-only)

Placeholder Nodes exist only to support incomplete graphs while the user is building. They are valid in **draft** status flows. When a flow is saved (or shared), the app should prompt to resolve placeholders or keep them as intentional endpoints based on CH12 rules.

- **add_response**: Represents an unfinished branch where the condition exists but the response move is not yet chosen.
- **add_branch_stub**: Represents a “tap to add branch” affordance anchored near a move node, if the UI uses explicit stubs instead of a toolbar action.

5.2 Summary Nodes (Optional, future-friendly)

If the product uses a “single summary node” to collapse a subtree for readability, implement it as a UI state on nodes (*NodeUI.collapsed=true*) rather than a separate node type. That keeps the graph stable for practice/path enumeration.

If a later version requires a real Summary Node type (e.g., to represent a saved subgraph reference), create it as CH13-R2 and update CH00 first.

6. Firebase (Firestore) Storage Shape (Recommended)

This section provides a buildable Firestore model that scales to large graphs without document-size limits.

6.1 Collections and documents

- **flows/{flowId}** — Flow document: metadata, rootNodeId, revision, layoutVersion, lightweight counters.
- **flows/{flowId}/nodes/{nodeId}** — Node documents: type, moveRef, ui, meta.
- **flows/{flowId}/edges/{edgeId}** — Edge documents: source, target, kind, conditionLabel, transition payload.

Rationale: Subcollections avoid large embedded arrays and allow partial updates (e.g., moving one node doesn’t rewrite the whole flow).

6.2 Example documents (JSON)

```
{
"id": "flow_123",
"ownerUserId": "user_abc",
"title": "Pressure Defense A",
"status": "saved",
"rootNodeId": "node_root",
"revision": 7,
"layout": { "layoutVersion": 1, "viewport": {"x": 0, "y": 0, "zoom": 1.0} },
```

```
"createdAt": "2026-01-02T18:10:00Z",
"updatedAt": "2026-01-02T19:05:00Z"
}

{
"id": "node_root",
"type": "move",
"moveRef": {
"canonicalMoveId": "move_jab",
"displayName": "Jab",
"userMoveId": null,
"stanceContext": "any"
},
"ui": { "x": 120, "y": 240 },
"meta": { "createdAt": "2026-01-02T18:10:05Z" }
}

{
"id": "edge_001",
"sourceNodeId": "node_root",
"targetNodeId": "node_cross",
"kind": "conditional",
"conditionLabel": "leans back",
"transition": {
"id": "edge_001",
"isEnabled": true,
"notes": "Opponent weight shifts rearward; head stays high."
}
}
```


7. Derived Computations (Used by other chapters)

This chapter defines how to derive commonly needed structures from the raw graph. These are used by Practice Mode and Sharing but owned by those chapters for behavior.

7.1 Path enumeration

A **path** is a sequence of nodes starting at rootNodeId and following edges until a leaf (node with 0 outgoing edges) or until a Placeholder Node is reached.

For V1, practice selection operates on enumerated paths. See: HZ-V1-CH20 (Practice Setup) for UI and credits rules.

Implementation guidance (non-binding): use DFS from root; stop at leaves; enforce cycle check; store path IDs as deterministic hashes of ordered node IDs + edge IDs.

7.2 Merge handling

Merges do not create ambiguity for path enumeration: different upstream paths may converge into the same downstream segment. Treat each full traversal from root to leaf as a distinct path, even if suffixes are shared.

8. Acceptance Test Checklist (Given/When/Then)

- Given a move node with 10 outgoing edges, when the user attempts to add an 11th edge, then the app blocks creation and triggers the CH30 warning/upgrade flow.
- Given a draft flow containing a placeholder add_response node, when the user saves the flow, then the flow remains saveable and the placeholder remains visible as an endpoint (unless CH12 requires resolution).
- Given two nodes A and B, when an edge is created from A to B, then the edge stores sourceNodeId=A and targetNodeId=B and can optionally store conditionLabel and transition payload.
- Given a user attempts to connect a node to itself, when saving the edge, then the system rejects the edge (self-loop forbidden).
- Given a graph with a potential cycle, when the edge that would complete the cycle is created, then the system rejects it and displays an error explaining cycles aren’t allowed in V1.
- Given a node is moved on the canvas, when the move is committed, then only NodeUI x/y are updated and no other node/edge fields change.
- Given an imported flow containing unknown future fields, when saving locally, then the app preserves the unknown fields (forward compatibility).

9. Replit Build Prompt (Implement CH13 Only)

```
You are implementing Handz V1 PRD Bundle (HZ-V1). Use CH00 rules.

Implement CH13: Flow Builder Node Types & Data Model ONLY. Do not implement UI gestures
(owned by CH12) or Practice behavior (CH20–CH22) beyond what’s necessary for data
structures.

1) Create Firestore collections:
- flows/{flowId}
- flows/{flowId}/nodes/{nodeId}
- flows/{flowId}/edges/{edgeId}

2) Define TypeScript types for Flow, Node, MoveRef, Edge, Transition, NodeUI, Layout.

3) Implement validation utilities:
- maxOutEdges=10
- no self-loops
- no cycles (reject edge if it introduces a cycle)
- allow leaf nodes
- allow placeholder nodes in draft flows

4) Implement CRUD:
- createFlow(title)
- addMoveNode(flowId, moveRef, position)
- addEdge(flowId, sourceId, targetId, kind, conditionLabel?)
- updateNodeUI(flowId, nodeId, x, y)
- upsertTransition(flowId, edgeId, transitionPayload)
- deleteNode(flowId, nodeId) (and cascade delete edges touching it)

5) Write minimal unit tests for validation rules and CRUD.

If any assumption is required, write it into a PRD-Assumptions comment block and STOP that
```

```
feature until confirmed.
```

10. Troubleshooting Notes (CH13)

- Symptom: adding an edge sometimes deletes another edge. Likely cause: using non-unique edge IDs (e.g., Date.now collisions). Fix: UUIDs.
- Symptom: canvas re-layout resets positions. Likely cause: overwriting NodeUI during node updates. Fix: partial update of ui.x/ui.y only.
- Symptom: practice path list infinite or app hangs. Likely cause: cycle introduced; cycle check missing. Fix: enforce cycle policy on edge creation.
- Symptom: imported flows lose metadata. Likely cause: strict schema parsing that drops unknown fields. Fix: preserve unknown fields during import.
- Symptom: edge labels not showing. Likely cause: conditionLabel stored for linear edges or kind mismatched. Fix: ensure kind=conditional when label exists.