OBSIDIAN & AI KNOWLEDGE SYSTEMS • FIELD GUIDE

Obsidian Bases & JSON Canvas with AI

Turning flat Markdown notes into live relational databases and spatial visual boards—having Claude Desktop and Antigravity generate native `.base` views, YAML schemas, and `.canvas` architecture maps.

- Native `.base` Databases
- Open JSON Canvas (`.canvas`)
- YAML Property Schemas
- Formulas & Dynamic Views
- Anti-Overlap Grid Math
- Zero Third-Party Plugins



Relational `.base` Views

Query thousands of vault notes into fast, two-way editable Tables and Gallery Cards powered by clean YAML filters and formulas.

Spatial `.canvas` Maps

Have AI agents calculate non-overlapping 2D grids, group boundaries, and labeled arrows directly in open JSON Canvas 1.0 files.

100% Plain-Text Native

Every note, database query, and whiteboard lives on your local disk as inspectable UTF-8 files with zero cloud or plugin lock-in.

The Native Tri-Format Stack: `.md` + `.base` + `.canvas`

Plain Markdown notes excel at linear prose, yet stumble when you need a sortable project tracker or a big-picture system diagram. Obsidian solves both natively—without proprietary cloud databases or fragile community plugins—using two plain-text companion formats that AI agents read and write fluently.

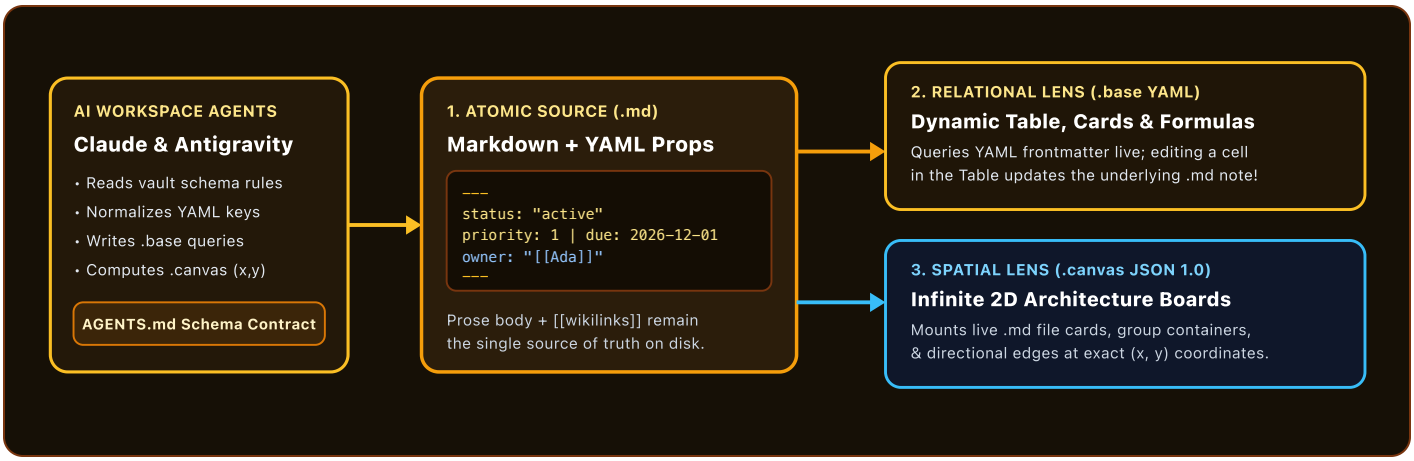
Quick Navigation

1. The Native Tri-Format Mental Model p. 2
2. Core Setup & Format Decision Matrix p. 3
3. Annotated Split-Studio UI Tour p. 4
4. Agent-Safe YAML Property Schemas p. 5
5. Authoring Native `.base` Views & Formulas p. 6
6. Open JSON Canvas (`.canvas`) & Grid Math p. 7
7. Bi-Directional Vault Workflows p. 8
8. 5 Copy-Paste Agent Recipes p. 9
9. Validation Scripts & `types.json` Polish p. 10
10. Cross-Tool Synergy Pipeline p. 11
11. Troubleshooting & Pre-Flight Check p. 12
Core Principle: One Vault, Three Lenses All

Why Obsidian's Tri-Format Stack Unlocks Structured Visual Thinking With AI

In cloud workspace tools, databases and whiteboards are trapped behind opaque APIs. In Obsidian, **all three layers sit side-by-side in your folder tree as human-readable text**: `.md` notes hold atomic knowledge and YAML frontmatter; `.base` YAML files define dynamic relational Table and Cards views over those properties; and `.canvas` JSON files arrange notes, groups, and arrows across an infinite 2D plane. When Claude Desktop (via Filesystem MCP) or Antigravity Desktop opens your vault, it can refactor all three layers in one coordinated pass.

Architecture: How Plain-Text Notes Feed Live Databases & Spatial Boards



Zero Duplication: Edit Once in Any View, Sync Everywhere Instantly

Neither a `.base` file nor a `.canvas` file copies your note contents into a separate database. Editing a `status` cell inside a `.base` Table—or typing inside an embedded `file` card on a `.canvas` board—mutates the original `.md` file directly on disk.

Core Plugin Setup & Format Decision Matrix

Everything in this handbook runs on 100% official, built-in Obsidian features—meaning zero community plugin security audits, instant startup on macOS/Windows/Linux/iOS/Android, and guaranteed long-term format stability.

1. Enable the Three Native Core Engines (60-Second Setup)

1 Enable Bases

Open `Settings → Core plugins` and toggle **Bases** ON. Right-click any folder and select **New base** for a standalone `.base` file, or embed a fenced `base` block inside any note.

2 Enable Canvas

In `Settings → Core plugins`, toggle **Canvas** ON. Under `Settings → Canvas`, set new file location to `Same folder as current file` so agent boards stay co-located with notes.

3 Properties View

Toggle **Properties view** ON, and under `Settings → Editor → Properties in document` pick **Visible** to unlock the All Properties registry backed by `.obsidian/types.json`.

2. Point Claude Desktop or Antigravity Desktop at Your Vault

In **Antigravity Desktop**, open your vault root folder directly (`File → Open Folder`). In **Claude Desktop**, grant the official Filesystem MCP server access to your vault directory inside `claude_desktop_config.json`:

```
{ "mcpServers": {
    "obsidian-vault": {
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-filesystem", "/Users/you/Vaults/WorkVault"]
    } } }
```

3. Format Decision Matrix: When to Use Tables, `.base`, `.canvas`, or Dataview

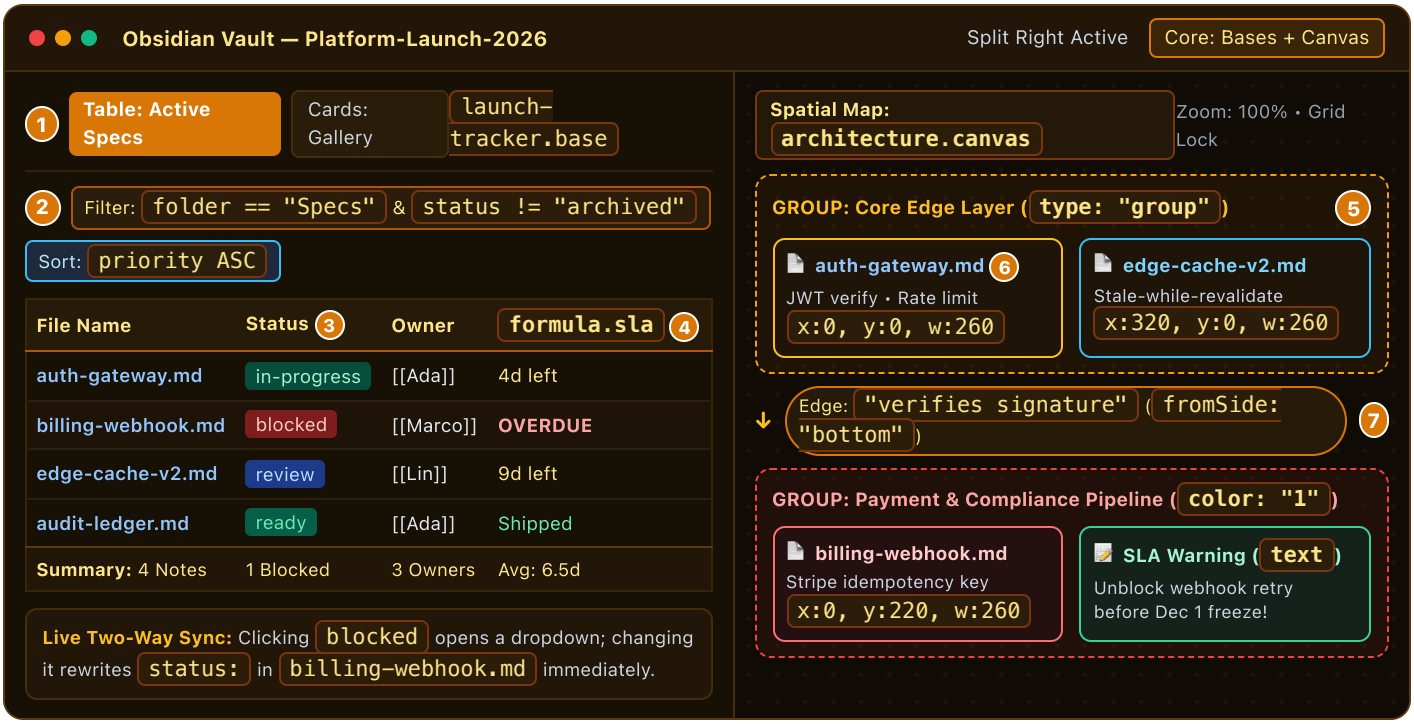
Vault Format	Best Use Case	Query Speed & Editing	AI Agent Reliability	Mobile & Longevity
Static Markdown Table (inside `.md`)	Small fixed reference matrices (5–15 rows) tied to one doc	Instant render; manual cells; zero live cross-note queries	**Very High**—simple pipe syntax, though wide tables wrap	Universal Markdown everywhere forever
Native Obsidian Base (`.base` YAML)	Project trackers, CRMs, reading queues, Table/Cards cockpits	**Native C++-grade speed** on 10k+ notes; two-way cell edits	**High**—declarative YAML schema with clean formulas	100% official core; full iOS/Android parity
JSON Canvas 1.0 (`.canvas` JSON)	System architectures, dependency flows, research clusters	Smooth 2D pan/zoom; live embedded file editing	**High with Grid Math**—open JSON spec + `(x,y)` slots	Open MIT spec (`jsoncanvas.org`); native mobile
Legacy Dataview (Community Plugin)	Procedural JS aggregations (`dataviewjs`) over inline fields	JS re-indexing; read-only cells without extra plugins	**Medium**—DQL/JS quirks often cause subtle query bugs	Third-party plugin; migrate tables to `.base`

Rule of Thumb: Store Facts in YAML, Query with `.base`, Map Relationships with `.canvas`

Whenever data evolves across multiple notes (tasks, contacts, literature, services), skip static Markdown tables that immediately go stale. Give each item an atomic `.md` note with clean YAML properties, then generate a `.base` view for sorting and a `.canvas` map for topology.

Annotated Tour: Split-Pane `.base` & `.canvas` Studio

Opening a relational `.base` view on the left and a spatial `.canvas` board on the right turns Obsidian into an interactive command center where tabular metadata and architectural topology stay linked in real time.



Workspace Callout Legend (Controls `1`–`7`)

1 Multi-View Switcher (`views[]`): One `.base` file stores multiple named layouts—switch instantaneously between a dense `table` view and a visual `cards` gallery.

2 Declarative Filter & Sort Pills: Visual pills reflect the YAML `filters` block (`and` / `or` / `not`) and sort priority without writing procedural scripts.

3 Two-Way Editable Property Cells: Every column maps to a YAML frontmatter key; editing a cell in-place updates the target Markdown file on disk.

4 Computed Formula Column (`formulas`): Calculates live countdowns, SLA flags, or ratios from note dates and numbers, plus bottom column summaries.

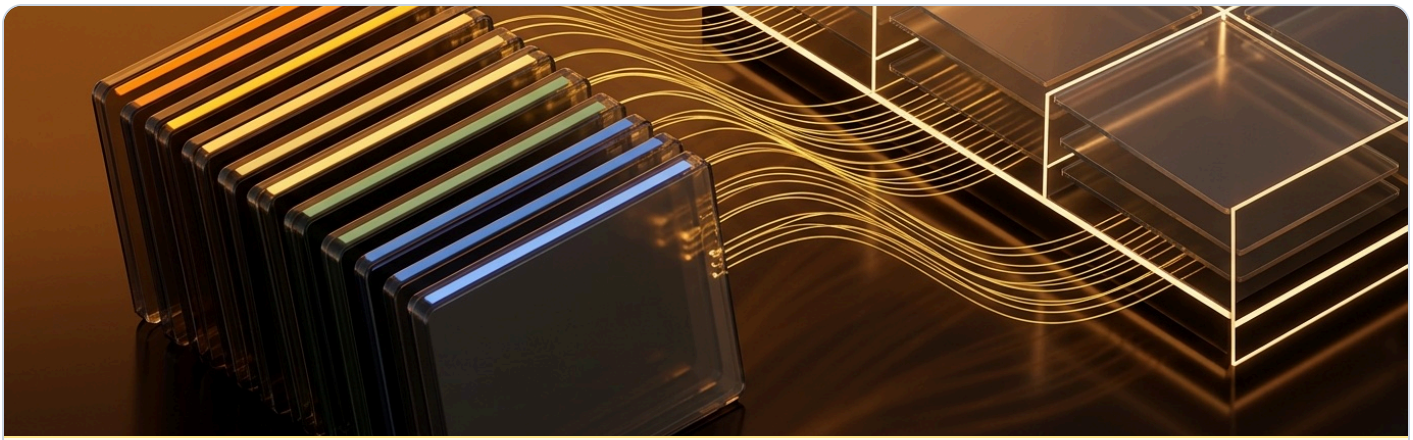
5 Spatial Group Frame (`type: "group"`): Bounding-box container listed first in `nodes[]` so child cards render cleanly inside its labeled border.

6 Live Mounted File Node (`type: "file"`): Embeds the actual `auth-gateway.md` note on the 2D plane—scroll or edit it right inside the canvas!

7 Labeled Directional Edge (`edges[]`): Connects exact anchor sides (`fromSide: "bottom"` → `toSide: "top"`) with arrow markers and semantic relationship labels so humans and agents can trace data flow visually.

Designing Agent-Safe YAML Property Schemas

A `.base` database is only as sharp as the YAML frontmatter underneath it. Without an explicit schema contract, AI agents drift across synonyms—writing `dueDate` on Monday and `due_date` on Tuesday—silently splitting one column into sparse fragments.



Schema-First Precision: Standardized YAML frontmatter headers act as color-coded keys that slot every atomic Markdown note cleanly into your relational `.base` Table and Cards layouts.

1. Obsidian's 6 Native Property Types (Backed by `.obsidian/types.json`)

UI Type	`types.json` ID	Exact Agent YAML Syntax	Agent Guardrail & Base Filter Behavior
Text	`"text"`	`status: "active"` | `owner: "[[Ada]]"`	Always quote wikilinks (`"[[Ada]]"`)! Unquoted `[[Ada]]` parses as a nested YAML array.
Number	`"number"`	`priority: 1` | `effort_hrs: 4.5`	Emit bare numbers without quotes so `<`, `>`, and `Average` summaries compute numerically.
Checkbox	`"checkbox"`	`shipped: false`	Use lowercase YAML booleans (`true` / `false`), never `"yes"` or `"True"` strings.
Date	`"date"`	`due: 2026-12-15`	Strict ISO-8601 `YYYY-MM-DD` only; powers `date(due) < today()` in `.base` formulas.
Date & time	`"datetime"`	`reviewed_at: 2026-11-12T14:30:00`	ISO-8601 timestamp (`YYYY-MM-DDTHH:mm:ss`); ideal for meeting logs and sync stamps.
List / Tags	`"multitext"` / `"tags"`	`tags: [project, q4]`	Always emit a YAML array even for single items (`[q4]`) so `.contains()` never fails.

2. Pin a Canonical Property Contract Inside Vault `AGENTS.md`

Place a concise `AGENTS.md` in your vault root so Antigravity Desktop and Claude Desktop enforce identical keys on every edit:

```
# Vault Metadata Schema Contract (AGENTS.md) — Use ONLY canonical snake_case keys:
- status: "inbox" | "active" | "blocked" | "review" | "done"  (Text)
- priority: 1 | 2 | 3                                        (Number: 1=Urgent, 3=Low)
- due: YYYY-MM-DD                                            (Date: strict ISO-8601)
- owner: "[[Full Name]]"  &  project: "[[Project Note]]"     (Text: ALWAYS quote wikilinks)
- tags: [lowercase-kebab]                                    (YAML List, no leading #)
```

Authoring Native `.base` Files & Formulas With AI

Every Obsidian `.base` file is a declarative YAML document with four top-level blocks: global `filters`, calculated `formulas`, column `properties` labels, and a `views` array defining Table and Cards tabs.

1. Production `roadmap.base` Anatomy (Copy-Paste Reference)

```
filters:
  and:
    - or: [file.inFolder("Projects"), file.inFolder("Specs")]
    - 'status != "archived"'
formulas:
  days_left: 'if(due, (date(due) - today()).days, "")'
  sla_badge: 'if(status == "done", "Shipped", if(date(due) < today(), "OVERDUE", "On Track"))'
properties:
  file.name: { displayName: "Deliverable" }
  status: { displayName: "Stage" }
  formula.sla_badge: { displayName: "SLA Health" }
views:
  - type: table
    name: "Active Execution Table"
    limit: 50
    groupBy: { property: status, direction: ASC }
    order: [file.name, status, priority, owner, due, formula.sla_badge, formula.days_left]
  - type: cards
    name: "Blocked & Urgent Cards"
    filters:
      or: ['status == "blocked"', 'priority == 1']
    order: [file.name, owner, due, formula.sla_badge]
```

2. High-Value Built-In `.base` Functions Agents Should Use

File Context

`file.inFolder("Specs")`, `file.hasTag("q4")`, `file.hasLink("Roadmap")`, `file.name`, and `file.mtime` filter notes without redundant YAML keys.

Date & Math

Compare `date(due) < today()`, inspect `now()`, format decimals via `.toFixed(2)`, and branch cleanly with `if(cond, a, b)`.

List & Embeds

Test multi-value lists via `tags.contains("urgent")`. Embed any view inside Markdown using `![[roadmap.base#Blocked & Urgent Cards]]`!

3. Color-Coded Prompt Anatomy: Generating Multi-View `.base` Cockpits

[SCHEMA READ] First read `AGENTS.md` and `.obsidian/types.json` to verify exact property names and types. [DATASET SCOPE] Create `Dashboards/engineering-radar.base` filtering notes where `file.inFolder("Specs")` and `status != "archived"`. [FORMULAS] Add two `formulas`: `age_days` from `file.ctime` and `risk_flag` flagging items where `priority == 1` and `status == "blocked"`. [VIEWS & ORDER] Configure two `views`: a grouped `type: table` ordered by `priority`, plus a `type: cards` view limited to `limit: 12` top risks. [SYNTAX GUARDRAIL] Output valid standalone YAML only—quote expressions containing operators like `!=` or `<` so YAML parsers never choke on punctuation.

Standalone `.base` Files vs. Inline Embedded Base Blocks

Save reusable multi-tab trackers as standalone `.base` files so they appear in Quick Switcher and Graph View; reserve inline fenced `base` code blocks inside Markdown notes for one-off contextual mini-tables (such as listing child tasks inside a single project hub note).

Mastering Open JSON Canvas (`.canvas`) & Grid Math

Published under an open specification at `jsoncanvas.org`, every `.canvas` file is a standard JSON object with two arrays: `nodes[]` (ordered back-to-front by z-index) and `edges[]`. Deterministic grid math turns messy guesses into crisp blueprints.

1. Exact JSON Canvas 1.0 Schema Reference

```
{
  "nodes": [
    { "id": "grp-core-0000001", "type": "group",
      "label": "Core Services", "color": "4",
      "x": -40, "y": -48, "width": 720, "height": 268 },
    { "id": "nd-auth-00000001", "type": "file",
      "file": "Specs/auth-gateway.md",
      "subpath": "#Token Flow", "color": "5",
      "x": 0, "y": 0, "width": 300, "height": 180 },
    { "id": "nd-note-002", "type": "text", "color": "2",
      "text": "### Rate Limit\n1,200 req/min per key",
      "x": 380, "y": 0, "width": 260, "height": 180 }
  ],
  "edges": [
    { "id": "ed-auth-rate-001",
      "fromNode": "nd-auth-00000001", "fromSide": "right",
      "toNode": "nd-note-002", "toSide": "left",
      "toEnd": "arrow", "label": "enforces" }
  ]
}
```

4 Official Node Types (`nodes[]`)

- `"group"`: Visual container (`label`). **List groups FIRST in `nodes[]`** so they render behind cards!
- `"file"`: Relative vault path (`"Specs/auth.md"`) + optional heading `subpath`.
- `"text"`: Standalone Markdown card (`text` with `\n`).
- `"link"`: Live external URL card (`url`).

Obsidian Preset Palette (`color`)

Strings `"1"` Red (blockers), `"2"` Orange (warnings), `"3"` Yellow, `"4"` Green (done), `"5"` Cyan (APIs), `"6"` Purple (specs)—or hex `"#d97706"`.

Clean Orthogonal Edges (`edges[]`)

Always set `fromSide` & `toSide`: left-to-right flows wire `"right" → "left"`; top-down trees wire `"bottom" → "top"`.

2. Anti-Overlap Coordinate Grid Math (Never Stack Nodes at `0, 0`!)

When prompted vaguely to *"make a canvas,"* LLMs often guess overlapping coordinates. Hand your agent these four deterministic layout equations:

Layout Element	Deterministic Integer Formula	Why It Guarantees Readability
Card Slot `(x, y)`	`x = col * 380` | `y = row * 240`	For `300×180` cards, leaves an `80px` horizontal gutter for edge labels and a `60px` vertical gutter.
Height by Lines	`height = max(160, 90 + lines * 24)`	Prevents vertical scrollbars on multi-bullet `text` cards at overview zoom.
Group Bounds	`gx = min_x - 40`, `gy = min_y - 48` `gw = (max_r - min_x) + 80`	Wraps child cards with `40px` side padding and `48px` top clearance for the group title bar.
Integer Pixels	`int(round(coord))` (no floats!)	JSON Canvas 1.0 requires strict integer pixel values for `x`, `y`, `width`, and `height`.

Strict JSON Hygiene: Zero Comments & Zero Trailing Commas

Obsidian's `.canvas` parser enforces strict RFC 8259 JSON. A single `// comment` or trailing comma renders an empty canvas—always validate with `python3 -m json.tool` before saving.

Bi-Directional Workflows: Notes ↔ Bases ↔ Canvas

Because all three formats live as plain text in your workspace, information flows effortlessly in **both directions**: synthesize scattered notes upward into dashboards, or sketch rough boxes on a canvas and let an AI agent scaffold your vault downward.



Spatial-to-Relational Roundtrip: Sketch high-level group frames and rough concept cards visually on an infinite `.canvas` table, then instruct Antigravity or Claude Desktop to turn every card into a structured Markdown spec and live `.base` tracker.

A Bottom-Up Synthesis (Notes → Views)

When to use: You have 30+ messy meeting notes, ADRs, or research clippings in a folder and need operational clarity fast.

1. **Audit & Normalize:** Ask Claude Desktop or Antigravity to scan `Projects/Q4/*.md` and backfill missing `status`, `owner`, `priority`, and ISO `due` YAML frontmatter per `AGENTS.md`.
2. **Generate Operational Base:** Have the agent author `Projects/Q4/roadmap.base` with an execution Table tab and a blocked-items Cards tab.
3. **Map Dependencies:** Have the agent inspect `[[wikilinks]]` between those 30 notes and emit `Projects/Q4/architecture.canvas` grouping upstream vs. downstream services on a `380×240` grid.

B Top-Down Scaffolding (Canvas → Vault)

When to use: You prefer brainstorming visually—dragging out 12 rough `text` cards and arrows on `brainstorm.canvas` during a whiteboard session.

1. **Read Visual Topology:** Point Antigravity at `brainstorm.canvas`. It parses `nodes[]`, parent `group` bounds, and `edges[]` to infer hierarchy and dependencies!
2. **Scaffold Linked Specs:** For every `text` card, the agent creates a dedicated `Specs/<slug>.md` note populated with YAML properties (setting `category` from the enclosing Canvas group and `depends_on` from incoming arrows).
3. **Upgrade Cards In-Place:** The agent converts those temporary `"type": "text"` nodes in `brainstorm.canvas` into live `"type": "file"` nodes and generates `Specs/tracker.base`!

Pro Technique: Non-Destructive Canvas Upgrades With Backup Snapshots

Before letting an agent rewrite an existing hand-arranged `.canvas` file (such as converting `text` cards to `file` cards), instruct it to preserve your exact `x`, `y`, `width`, `height`, and `color` attributes verbatim—modifying only `"type": "file"` and `"file": "Specs/..."` so your spatial muscle memory stays untouched.

5 Real-World Copy-Paste `.base` & `.canvas` Recipes

Drop any of these five battle-tested prompts directly into Claude Desktop (with Filesystem MCP) or Antigravity Desktop to build complete, zero-plugin relational and spatial systems inside your vault in under sixty seconds.

Recipe & Target Files	YAML Schema & Layout Spec	Copy-Paste Prompt for Claude Desktop / Antigravity
1. Personal CRM & Follow-Up Radar `People/*.md` + `crm.base`	Props: `role`, `org`, `cadence_days` (Number), `last_met` (Date). Views: Overdue Table + VIP Cards.	*"Scan `People/*.md` and ensure every note has YAML keys `org`, `role`, `cadence_days` (number, default 30), and ISO `last_met`. Then create `People/crm.base` with formula `days_since: '(today() - date(last_met)).days'` and `needs_ping: 'if((today() - date(last_met)).days > cadence_days, \"Ping Now\", \"Warm\")'`. Add a Table sorted by `days_since` DESC and a Cards view filtered to `Ping Now`."*
2. Literature & Evidence Matrix `Papers/*.md` + `evidence.base`	Props: `year` (Number), `methodology`, `verdict`, `topics` (List). Views: Synthesis Table + Topic Cards.	*"Read all notes in `Papers/`, standardize frontmatter to `authors` (list), `year` (number), `methodology`, `sample_size` (number), and `verdict` (supports/mixed/refutes). Generate `Papers/evidence.base` with a Table grouped by `verdict` showing `Average` on `sample_size`, plus a Cards view for papers where `year >= 2025`."*
3. Microservice Architecture Mapper `arch-map.canvas`	4 layered `group` lanes (Clients, Gateway, Services, Stores) on `col*380, row*240` grid.	*"Inspect our repo's `services/` folders and `Docs/ADR/*.md`. Generate `Docs/arch-map.canvas` following JSON Canvas 1.0: place 4 horizontal `group` containers first in `nodes[]`, mount matching ADR notes as `file` nodes and external APIs as `text` nodes using `x = col * 380`, `y = row * 240` (zero overlaps), and connect calls via labeled orthogonal `edges[]`."*
4. Product Launch Cockpit + Dependency Map `launch.base` + `deps.canvas`	Synchronized pair: `.base` tracks owner/SLA; `.canvas` maps critical-path blockers in red (`color: "1"`).	*"Across `Launch/*.md`, normalize `status`, `owner`, `due`, and `blocked_by` (quoted wikilink list). Create `Launch/launch.base` with Table and Cards views, AND generate `Launch/deps.canvas` mounting every launch note as a `file` node colored `\"1\"` if blocked or `\"4\"` if done, wiring `blocked_by` links as left-to-right arrows."*
5. Legacy Dataview-to-Native-`.base` Migrator Vault-wide `*.base`	Converts read-only community `dataview` DQL blocks into fast two-way native `.base` files + embeds.	*"Find every note in my vault containing legacy Dataview TABLE or LIST queries. For each query, translate its `FROM`, `WHERE`, `FLATTEN`/calculated columns, and `SORT` clauses into a standalone native `.base` file in `Bases/`, then replace the old block with a clean `![[Bases/name.base]]` embed after showing me a diff preview."*

Embed Multiple Views Inside One Executive Summary Note

Once your agent creates `Launch/launch.base` and `Launch/deps.canvas`, create a single `Launch/README.md` hub note containing `![[launch.base#Blocked & Urgent Cards]]` right above `![[deps.canvas]]`—giving stakeholders a live relational table and spatial map on one scrollable page.

Validation Scripts & Layout Polish in Antigravity

Professional agent workflows never trust raw LLM JSON blindly. Adding a tiny 25-line Python check inside Antigravity Desktop catches broken file paths, duplicate IDs, float coordinates, and overlapping cards in milliseconds before you even switch windows to Obsidian.

1. The 25-Line `validate_canvas.py` Pre-Flight Script

Save this helper in a local `_scripts/validate_canvas.py` file (or let Antigravity run it automatically after generating any `.canvas` board):

```
import json, sys, pathlib

vault = pathlib.Path(sys.argv[1]).resolve()
canvas_path = pathlib.Path(sys.argv[2]).resolve()
data = json.loads(canvas_path.read_text(encoding="utf-8"))  # Fails fast on trailing commas!

nodes, edges = data.get("nodes", []), data.get("edges", [])
ids = [n["id"] for n in nodes] + [e["id"] for e in edges]
assert len(ids) == len(set(ids)), "Duplicate node/edge ID detected!"

cards = [n for n in nodes if n["type"] != "group"]
for n in nodes:
    for k in ("x", "y", "width", "height"):
        assert isinstance(n[k], int), f"Node {n['id']} {k}={n[k]} must be integer!"
    if n["type"] == "file":
        assert (vault / n["file"]).exists(), f"Missing vault file: {n['file']}"

for i, a in enumerate(cards):
    for b in cards[i + 1:]:
        overlap = (a["x"] < b["x"] + b["width"] and a["x"] + a["width"] > b["x"] and
                   a["y"] < b["y"] + b["height"] and a["y"] + a["height"] > b["y"])
        assert not overlap, f"Bounding-box collision between {a['id']} and {b['id']}!"
print(f"OK: {len(nodes)} nodes, {len(edges)} edges, 0 broken paths, 0 card overlaps.")
```

2. Syncing `.obsidian/types.json` So Property Pickers Just Work

When an AI agent introduces a brand-new YAML property across twenty Markdown files (such as `due` or `cadence_days`), Obsidian infers its type on first scan—unless an older note accidentally stored a string in that field, triggering a type-warning icon in Properties view and breaking numeric or date sorting inside `.base` tables.

The Fix: Have Antigravity inspect and update `.obsidian/types.json` whenever you add a new canonical field to `AGENTS.md`. Locking `"due": "date"` and `"priority": "number"` ensures native calendar pickers, checkboxes, and `.base` column summaries behave deterministically across your entire vault.

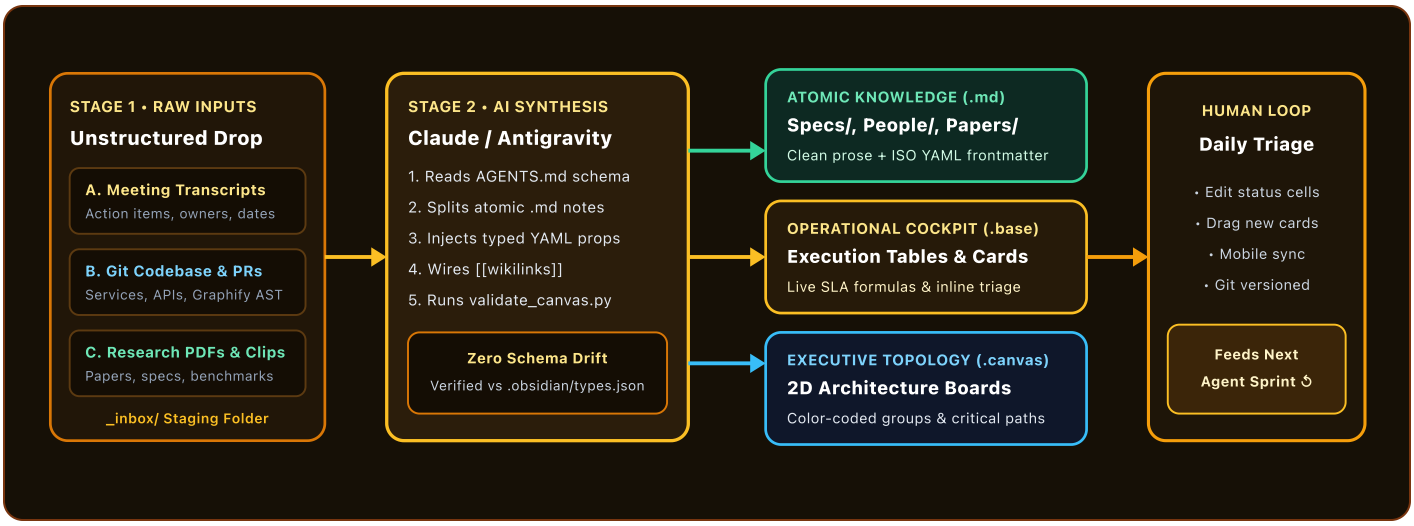
```
// .obsidian/types.json (valid strict JSON)
{
  "types": {
    "aliases": "aliases",
    "tags": "tags",
    "type": "text",
    "status": "text",
    "owner": "text",
    "project": "text",
    "priority": "number",
    "cadence_days": "number",
    "due": "date",
    "last_met": "date",
    "reviewed_at": "datetime",
    "shipped": "checkbox"
  }
}
```

Git Hygiene for Plain-Text Vaults: Diff-Friendly YAML & Sorted Canvas Keys

Commit your `.md`, `.base`, `.canvas`, and `.obsidian/types.json` files to Git! Instruct agents to preserve existing node IDs when updating a `.canvas` board rather than regenerating random UUIDs—keeping your `git diff` clean and human-reviewable.

Cross-Tool Synergy Pipeline: Raw Inputs to Visual Vault

Native `.base` views and `.canvas` boards shine brightest as the visual presentation layer at the end of an automated AI research and engineering pipeline—turning messy transcripts, repositories, and PDFs into a living operating system.



Four-Stage Weekly Operating Rhythm

1 Capture Raw Signal Into `_inbox/`

Drop raw call transcripts, PDF papers, or Graphify codebase reports into a dedicated `_inbox/` folder without worrying about formatting or metadata during deep work.

2 Run Schema-Guided Extraction

Ask Claude Desktop or Antigravity to distill `_inbox/` items into typed notes inside `Specs/`, `People/`, or `Papers/` strictly adhering to your `AGENTS.md` YAML keys.

3 Refresh Operational `.base` & `.canvas`

Your existing `.base` tables pick up new notes automatically via folder/tag filters! Have the agent update `architecture.canvas` whenever new cross-service links appear.

4 Review on Desktop or Mobile, Then Commit

Triage priorities in your `.base` Table over morning coffee on iPad or iPhone, sketch new edges on Canvas, and commit the plain-text diff cleanly to Git.

Troubleshooting Clinic & Pre-Flight Checklist

Nearly every broken `.base` column or unreadable `.canvas` board traces back to one of seven syntax mismatches. Keep this diagnostic table handy for ten-second fixes.

Top 7 Issues & 10-Second Fixes

Symptom in Obsidian	Root Cause on Disk	10-Second Deterministic Fix
1. All `.canvas` cards stacked in a pile at `(0, 0)`	Agent omitted grid spacing or set constant `x: 0, y: 0` across nodes.	Re-prompt with explicit grid math: `x = col * 380`, `y = row * 240`, and run `validate_canvas.py` to assert zero bounding-box overlaps.
2. Opening a `.canvas` file shows a blank/error pane	Trailing comma after last array item or `//` comments inside JSON.	Run `python3 -m json.tool board.canvas > /dev/null` to pinpoint the offending line and strip trailing commas/comments.
3. `.base` date sorting & `today()` formulas fail	Human-written date strings like `Nov 12, 2026` or `12/11/26` in YAML.	Normalize all YAML date fields to unquoted or quoted strict ISO-8601 `YYYY-MM-DD` (e.g., `due: 2026-11-12`).
4. `.contains()` filter misses matching notes in `.base`	Property mixes scalar strings (`tags: "q4"`) in some notes and YAML lists in others.	Standardize multi-value fields as YAML arrays everywhere (`tags: [q4]`), or wrap mixed properties with `list(tags).contains("q4")`.
5. Canvas `file` node says "File not found"	Agent wrote an OS absolute path (`/Users/...`) or omitted the `.md` extension.	JSON Canvas `"file"` paths must be vault-root relative WITH extension (e.g., `"Specs/auth-gateway.md"`, no leading slash).
6. Warning triangle next to property in Sidebar	Note value conflicts with global type registered in `.obsidian/types.json`.	Check `.obsidian/types.json`; convert stray quoted numbers (`priority: "1"` → `1`) or unquoted wikilink arrays (`"[[Ada]]"`).
7. Long Markdown text cut off inside Canvas cards	Fixed `height: 120` used on a 10-line bulleted `"type": "text"` node.	Scale height dynamically by line count: `height = max(160, 90 + line_count * 24)` and widen dense cards to `width: 320`.

6-Point Pre-Flight Checklist Before Shipping Any Agent Output

☑ **1. Schema Contract Present:** Root `AGENTS.md` defines canonical `snake_case` keys and allowed enum values.

☑ **2. Quoted YAML Wikilinks & ISO Dates:** Every wikilink in frontmatter is quoted (`"[[Target]]"`) and dates match `YYYY-MM-DD`.

☑ **3. Valid `.obsidian/types.json`:** New numeric, checkbox, list, and date fields are registered cleanly in `types.json`.

☑ **4. Quoted `.base` Expressions:** Filter and formula strings containing `!=`, `:`, or `<` are wrapped in single quotes in YAML.

☑ **5. Groups First & Integer Coords:** In `.canvas`, `group` nodes appear first in `nodes[]` and all `x, y, width, height` are integers.

☑ **6. Zero Collisions & Valid Paths:** `validate_canvas.py` reports `0` missing vault files, unique IDs, and `0` overlapping cards.