AI Agent User's Guide (Split-Role Workflow)

Version: 4.0.0 Date: 2025-12-07

--- Revision History ---

[4.0.0] - 2025-12-07

- Major rewrite to align with `AIAgentWorkflow.md` v4.5.0.

- Defined the "Two-Step" Architect workflow (Analysis -> Halt -> Planning).

- Documented new triggers: "Generate Builder Execution Kit" and "Initiate Architect Handoff".

- Added "Operational Disciplines" section covering Macros, Overrides, and Exact Prompts.

[3.2.0] - 2025-12-06

- Updated Phase 1 Output: Added "Reconciliation Updates" to Roadmap expectations.

- Added "Troubleshooting State Drift" section.

1. The Workflow Concept

The workflow is a strict **State Machine** designed to prevent hallucinations and ensure code integrity. It operates on a **"Funnel Architecture"**:

- **The Architect (Maximum Context):** Sees the **Full World Truth** (Code, Docs, Data). Analyzes the problem, defines the solution, and generates the *instructions* for the Builder.
- **The Builder (Minimum Viable Context):** Sees **Only** the files listed in the Manifest. Operates in a "Safe Mode" to execute the plan without inventing non-existent dependencies.
- **You (The User):** The "Message Bus" and "FileSystem." You act as the bridge, moving files and authorizing state transitions.

2. Phase 1: The Architect Session

Goal: Analyze the problem and generate the blueprints (`ImplementationPlan.md`).

1. **Start:** Open a new session.
2. **Bootstrap (The World Truth):** Upload the **FULL** project context:
    - `AIAgentWorkflow.md` (The Rules)
    - **`project_bundle.txt`** (The Source Code)
    - **`documentation_bundle.txt`** (The Docs)
    - **`data_bundle.txt`** (The Data Schemas)
    - **`ArchitectureRoadmap.md`** (The Long-Term Memory)
3. **Prompt:** ”Act as Architect. [Describe task].”
4. **Analysis Phase:** The Architect will analyze the request, identify scope, and check files.
    - **CRITICAL:** The Architect will **HALT** here. It will *not* generate the plan yet.
5. **Triggering the Plan:** The Architect will ask you to proceed.
    - **Command: ”Generate Builder Execution Kit”**
    - *Note:* This command is universal for creating or updating the plan.
6. **Output:** The Architect produces the **Builder Execution Kit**:
    - **Implementation Plan:** The specific blueprints.
    - **Manifest:** A precise list of files required.
    - **Technical Debt Register:** (Passive) Cleanup opportunities.

Phase 1.5: The Architectural Relay

Goal: Save state before ending the session to prevent ”Context Amnesia.”

1. **Trigger:** When you are ready to stop or switch sessions.
    - **Command: ”Initiate Architect Handoff”**
2. **Output:** The Architect generates the **Initialization Kit**:
    - **`ArchitectureRoadmap.md:`** Updated status of all phases.
    - **`ArchitectHandoff.md:`** Narrative context and ephemeral constraints.
3. **Action:** Save these files immediately. They are required to start the next Architect session.

3. Phase 2: The Bridge (The Trinity)

To move from Planning to Execution, you must assemble **”The Trinity”**:

1. **Workflow:** `AIAgentWorkflow.md`
2. **Plan:** `ImplementationPlan.md` (Generated by Architect)
3. **Bundle:** `builder_bundle.txt`
    - *Action:* Create this file containing **ONLY** the files listed in the `manifest.txt`.

4. Phase 3: The Builder Session

Goal: Execute the plan with rigorous verification.

1. **Start:** Open a **FRESH** session.
2. **Upload:** Upload **The Trinity**.
3. **Trigger: "Act as Builder"**
   - *Note:* This is a **System Macro**. It automatically loads strict formatting rules (Anti-Nesting, Headers).
4. **Confirm Context:** The Builder will list the files it is locking to.
   - **Command: "Confirmed"**
5. **Execute:** Follow the delivery loop.
   - **Pre-Flight Checklist:** The Builder MUST output a checklist of files to change.
   - **Proceed:** Type `Proceed` to authorize file generation.
6. **Sanitize (CRITICAL):**
   - The Builder will replace internal triple-backticks (in docstrings/markdown) with `__CODE_BLOCK__`.
   - **Action: Find & Replace** `__CODE_BLOCK__` with real triple-backticks (` ``` `) in your editor after saving.
7. **Verify:**
   - **Code:** Run the provided CLI command (e.g., `python main_cli.py ...`).
   - **Docs:** "Verification: Visual Inspection of the output above."

5. Operational Disciplines

The Exact Prompt Protocol

If the Agent asks you to provide a specific phrase (e.g., *"To proceed, please type: 'Generate Builder Execution Kit'"*), you **MUST** type it exactly. Synonyms or "Okay, do it" will be rejected to ensure safety.

The Doctrine of Conversational Override

- **Rule:** Your verbal instructions in the chat **supersede** any written plan or file.
- **Warning:** If you give an instruction that conflicts with the protocols, the Agent will output: `[OVERRIDE WARNING] Your verbal instruction X conflicts with Protocol Y...` This confirms the Agent heard you and is consciously disobeying the protocol to serve your request.

6. Troubleshooting

- **"Missing Context" Error:** The Builder only sees `builder_bundle.txt`. If it needs a file not in the bundle, you must go back to the Architect and update the Manifest.

- **Agent Stalls/Refuses:** Check if you used the **Exact Prompt**. Did you say ”Make the plan” instead of ”Generate Builder Execution Kit”?