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

Version: 2.0.1 Date: 2025-11-23

1. The Workflow Concept

We have moved from a ”Monolithic” AI (one session does everything) to a **”Split-Role” Model**. This prevents the AI from ”forgetting” code in the middle of a large file by keeping its context window clean.

- **The Architect:** Thinks, plans, and reads code. Does NOT write final files.
- **The Builder:** Writes code. Has ”amnesia” (knows nothing about the project history, only what is in the specific bundle you give it).
- **You (The User):** You are the ”Message Bus.” You move the Plan from the Architect to the Builder.

---

2. Phase 1: The Architect Session

Goal: Define *what* to do and generate a specific list of files needed to do it.

1. **Start:** Open a new AI session.
2. **Bootstrap:** Upload `AIAgentWorkflow.md` and your `project_bundle.txt` (or the specific files relevant to the task).
3. **Prompt:** ”Act as Architect. [Describe your bug or feature].”
4. **Context Receipt:** The Architect will read the bundle to establish the immutable baseline.
5. **The Output:** The Architect will analyze the problem and produce an **Implementation Plan**.
    - Look for the **”Builder Bootstrap Prompt”** at the end of the plan. This is a code block pre-filled with instructions for the next session.
    - Look for the **`manifest.txt`** code block. This contains the specific file list for the next step.

---

3. Phase 2: The Bridge (Your Role)

Before opening the Builder session, you must prepare the ”Builder Context.”

1. **Get the Manifest:** Copy the content of the `manifest.txt` code block provided by the Architect. Save it to a file named `manifest.txt` in your project root.
2. **Run the Bundler:** Execute your bundling script: ___CODE_BLOCK___bash python test_code/create_project_bundle.py --manifest manifest.txt **CODE_BLOCK** This will generate a `builder_bundle.txt` containing *only* the files the Builder needs.

---

4. Phase 3: The Builder Session

Goal: Execute the plan safely.

1. **Start:** Open a **FRESH** AI session.
2. **Upload:** Upload `AIAgentWorkflow.md` and the `builder_bundle.txt` you just created.
3. **Paste:** Copy the **Builder Bootstrap Prompt** from the Architect session and paste it into the chat.
4. **Initialization:**
   - The Builder will verify it has the files listed in the Manifest.
   - **Version Check:** The Builder will ask you for the **Target Session Version** (e.g., `0.93.0`).
     - **Note:** If you provide `0.93.0`, the Builder will apply "Smart Versioning":
       - **New Files** will start at `0.93.0`.
       - **Existing Files** already in the `0.93.x` series will auto-increment (e.g., `0.93.4` -> `0.93.5`) to preserve history.
5. **Execution Loop:**
   - **Visual Diff:** The Builder will show you `Old Code` vs `New Code`.
   - **Proceed:** If it looks correct, type `Proceed`.
   - **Delivery:** The Builder gives you the full file.
   - **Acknowledge:** Type `Acknowledged` to move to the next file.

---

5. Special Cases

Documentation Updates

Documentation is treated as code.

1. **Architect:** Ask the Architect to "Plan the documentation update for v2.0."
2. **Builder:** Start a Builder session with the `Docs/` folder in the bundle. The Builder writes the markdown files.

The "Ad-Hoc" Shortcut

For trivial tasks (e.g., "Fix a typo in the README" or "Explain this function"), you do not need the full split workflow.

1. **Start:** Open a session.
2. **Prompt:** "Ad-Hoc Task: Fix this typo in..."
3. The AI will skip the Architect/Builder ceremony and just help you.

Troubleshooting

- **Builder Fails Verification:** If the Builder produces code that errors out, paste the error back to the Builder. It gets **ONE** attempt to fix it.
- **Fix Fails:** If the second attempt fails, **STOP**. Do not argue with the Builder. Close the session. Go back to the Architect session, paste the error, and ask for a revised plan.