HANDS-ON EXERCISE • SUPERWHISPER WORKSHOP

Exercise 6: Agentic Handoff Mode

Advanced **LOCAL • PHI-Safe** 15–20 minutes

LEARNING OBJECTIVE

Create a custom mode that transforms rambling spoken descriptions into structured task specifications optimized for AI coding agents. This is an *advanced pattern* for developers who use tools like Claude Code, Cursor Agent, or GitHub Copilot's chat features.

Key Insight: Structure Matters for AI Agents

AI coding agents perform **significantly better** when given structured task specifications rather than stream-of-consciousness instructions. The difference isn't subtle—structured prompts lead to fewer clarifying questions, fewer wrong turns, and faster completion. This mode acts as an **intermediary translator** between how you naturally think out loud and what agents need to execute effectively.

The Agentic Handoff Pattern

This mode creates a bridge between natural speech and agent-ready task specifications. The key is that SuperWhisper does the *structuring work*—you just think out loud.

$$\begin{tikzcd}
\text{Rambling speech} \arrow[r] & \text{SuperWhisper structures} \arrow[r] & \text{Paste to agent} \arrow[r] & \text{Agent executes}
\end{tikzcd}$$

Critical: The output is a *task specification*, not code. The AI agent then uses that specification to write code.

AI Agents This Works With

Any AI agent that accepts natural language instructions benefits from structured input. Common examples:

Claude Code	Cursor Agent	GitHub Copilot
Terminal agent	IDE agent	Chat mode

Before vs. After Structuring

Watch what happens when natural speech gets transformed into an agent-ready specification:

RAW SPEECH (UNSTRUCTURED)

"okay so there's this bug in the user settings page where when you try to change your email it doesn't validate properly and it lets you save invalid emails and also the success message shows even when it fails and oh we need to keep the existing CSS because design just approved it and I think the issue is somewhere in the handleSubmit function"

STRUCTURED HANDOFF (AGENT-READY)

Objective: Fix email validation bug in user settings page

Context: handleSubmit function in settings component

Requirements:
• Validate email format before saving
• Only show success message on actual success

Constraints:
• Do NOT modify CSS (design-approved)

Success: Invalid emails rejected, correct messages shown

Same information. Different packaging. The structured version gives the agent clear objectives, constraints, and success criteria—dramatically reducing ambiguity.

Step-by-Step Setup

1 **Create New Custom Mode**

Open SuperWhisper Settings → Modes → Create Custom Mode

> **Name:** "Agentic Handoff" or "Task Structuring"
> *This name should remind you the output goes TO an agent*

2

Select Local Models (PHI-Safe)

Even developer work may reference sensitive data. Keep processing local.

> **Voice Model:** Fast or Standard (LOCAL)
> **Language Model:** Any LOCAL model
> *Standard model provides better structuring quality*

3

Configure Auto-Activation

Trigger this mode when you're in your development environment.

> **Recommended Applications:**
> ☑ Terminal / iTerm / Command Prompt
> ☑ VS Code (when Copilot Chat is active)
> ☑ Cursor (AI chat panel)
> *Anywhere you're giving instructions to an AI agent*

4

Enable ALL Context Types

This mode benefits from maximum context—it needs to know what you're working on.

> ☑ **Application Context** — knows which tool you're in
> ☑ **Selected Text** — include highlighted code/files
> ☑ **Clipboard** — include copied errors, references, file paths
> *All three enabled = richer task specifications*

5

Enter AI Instructions

Copy and paste this prompt into the AI Instructions field:

```
<role>
You are a task specification writer for AI coding agents.
</role>

<instructions>
The User Message is raw dictated speech describing a coding task.
Convert it into a structured task specification that an AI agent
can execute effectively. Output ONLY the structured task, not code.
</instructions>
```

```
<format>
**Objective:** [One clear sentence describing the goal]

**Context:** [Relevant files, components, or systems mentioned]

**Requirements:**
- [Specific requirement 1]
- [Specific requirement 2]

**Constraints:**
- [What NOT to change or touch]
- [Limitations to respect]

**Success Criteria:**
- [How to verify the task is complete]
</format>

<context_usage>
- If Selected Text provided: Reference specific files/code in Context section
- If Clipboard provided: Include error messages or references
- If neither: Ask user to select relevant code first
</context_usage>
```

Output Format Reference

Expected Output Structure

Objective: [What you're trying to accomplish] **Context:** [Files, functions, systems involved] **Requirements:** - [Specific thing that must happen] - [Another specific thing] **Constraints:** - [What must NOT change] - [Boundaries to respect] **Success Criteria:** - [How you'll know it worked]

Why This Format Works

This structure maps directly to how AI agents plan work: they need to know the **goal** (Objective), the **scope** (Context), the **specifics** (Requirements), the **boundaries** (Constraints), and how to **verify completion** (Success Criteria). Missing any of these leads to clarifying questions or wrong assumptions.

Test Your Mode

Test Workflow

1. Open your terminal (Claude Code) or VS Code (with Copilot)
2. **Select some relevant code** in your editor (becomes Selected Text context)
3. **Copy an error message** to clipboard if you have one
4. Hold your recording hotkey and **describe the task naturally**
5. Release → SuperWhisper outputs structured specification
6. Paste the structured task into your AI agent

SAY THIS (MESSY)

"so the API is returning a 500 error when users try to upload files larger than 5 megs and I need to add better error handling and also show a helpful message to the user and the error is in my clipboard"

EXPECT THIS (STRUCTURED)

Objective: Add error handling for large file uploads

Context: File upload endpoint, [clipboard error]

Requirements:
• Handle 500 errors for files > 5MB
• Display user-friendly error message

Success: Clear feedback on upload size limits

▯ Experiment: Test Different Scenarios

Try the mode with different types of development tasks and notice how the structuring adapts:

→ **Bug fix:** "there's this weird bug where..." → structured debugging task

→ **New feature:** "I need to add a button that..." → structured feature spec

→ **Refactoring:** "this code is messy and needs to..." → structured refactor plan

→ **With error in clipboard:** Results should reference the error message

Why This Pattern Works

Benefits of Structured Handoffs

- ✓ **Reduces ambiguity:** Agents don't have to guess what you meant
- ✓ **Preserves constraints:** "Don't touch X" is explicit, not implied
- ✓ **Enables verification:** Clear success criteria let you know when it's done
- ✓ **Leverages context:** Selected code and clipboard errors are included automatically
- ✓ **Faster iteration:** Clear specs mean fewer back-and-forth clarifications
- ✓ **Better results:** Structured input produces more accurate agent output

PRO TIP: COMBINE WITH PROMPT ENRICHMENT

Use **Prompt Enrichment mode** for general AI tools like ChatGPT and Claude web interface. Use **Agentic Handoff mode** specifically for coding agents that execute tasks autonomously. The difference: enrichment improves your question, handoff structures your instruction. Both are "input method" modes—neither generates final answers.

Troubleshooting

Output Includes Code Instead of Specification?

The AI might be trying to solve the problem directly. Make sure your prompt includes: `Output ONLY the structured task, not code.` If it persists, add: `Never write implementation code. Only structure the task.`

Missing Context/Clipboard in Output?

Verify that Selected Text and Clipboard context are enabled in mode settings. Also check that you actually have text selected or clipboard content before dictating.

Output Too Verbose?

Add this to your requirements: `Keep each section brief. Maximum 2 bullet points per section.`

✓ Completion Checklist

- [ ] Created "Agentic Handoff" mode with LOCAL models
- [ ] Added auto-activation for terminal/IDE apps
- [ ] Enabled ALL three context types (App, Selected, Clipboard)
- [ ] Pasted the task structuring instructions
- [ ] Tested with selected code + rambling description
- [ ] Verified output is structured spec, NOT code
- [ ] Successfully handed off to an AI agent

Congratulations! You've Completed Part 1

You now have a comprehensive voice-driven AI toolkit: local and cloud email modes, chat modes, prompt enrichment, custom vocabulary, and agentic handoff. In **Part 2**, we'll explore advanced workflows—chaining modes together, building complex automations, and integrating SuperWhisper with other AI tools in your daily work.