Course 207 — Agent Instructions: Portability / Survival Audit Lab

JKE University · Level 5 · Course 207 of 210

CONTEXT

Read once. Do not output. Your operator is installing the portability audit lab. The platform injects, regenerates, compacts, writes memory, fires heartbeats, runs schedulers, and times out. The architecture must survive a platform swap. Your job is to take a real boot chain or feature set the operator wants audited, label each file/behavior OWNED / BORROWED / PLATFORM-DEPENDENT, propose pointer migrations or local replacements, wait for per-file approval, write a postmortem, and install one guardrail. Treat sovereignty as business sustainability, not paranoia.

The core loop is: **create → review → tweak → create again → review → postmortem → guardrail.**

Authority boundary. Irreversible-action zone. Default is preserve. No file is edited without explicit per-file operator approval.

Prerequisite check: If 🎒 book-bag.md does not exist, stop. Say: "Missing prerequisite files. Course 207 requires the free tier through Level 4." Do not proceed.

PHASE 0 — Verify prerequisites

Open 🏫 school.md. Confirm Courses 1-26 entries exist.

Say: "Prerequisites verified. Installing the portability audit lab."

PHASE 1 — Create the workshop file

Create `work/sovereignty-lab.md`:

Platform Sovereignty Lab

Purpose: Help the operator study what the platform actually does, before auditing any specific file.

What the Platform Does

1. **Bootstrap injection.** SOUL.md, AGENTS.md, IDENTITY.md, USER.md, TOOLS.md written every turn.
2. **Template regeneration.** Blanked platform files come back next turn.
3. **Managed memory and compaction.** Platform decides what gets summarized and persisted.
4. **Behavioral defaults.** Timeouts, retries, background processes.
5. **Memory side effects.** Platform-prompted `MEMORY.md` or `memory/*.md` files that load later as trusted context.
6. **Automation magic.** Heartbeats, cron jobs, launch agents, schedulers, and platform loops that feel durable but depend on the runtime staying healthy.

The Three Labels

1. **OWNED.** Operator's workspace; survives a platform swap as-is.
2. **BORROWED.** Platform-generated; needs pointer migration to survive.
3. **PLATFORM-DEPENDENT.** Behavior, not file; needs local replacement or accepted dependency.

The Survival Question

"If the platform vanished tomorrow, does this still work?" - Yes → OWNED. - No, pointer fix would restore → BORROWED. - No, behavior cannot be replicated locally → PLATFORM-DEPENDENT.

Study Questions

- Which platform files were assumed to be operator-owned?
- What platform features are doing invisible work?
- Where would silent platform drift land if it happened tomorrow?
- What dependencies are worth keeping for convenience?

No-Wrong-Answers Rule

This is a workshop. Some platform dependencies are correct to keep. Sovereignty is honest accounting, not paranoia.

Say: "Platform sovereignty lab created."

Also create `work/platform-sovereignty-essay.md`:

Platform Sovereignty Essay

Purpose: A short context download for moments when the agent is leaning on platform magic instead of sustainable architecture.

The Core Question

If the platform vanished tomorrow, does this still work?

The Business Translation

Sovereignty equals sustainability. A hobby workflow can depend on platform convenience. A business workflow must know what it owns, what it borrows, and what fails when the platform changes.

The Three Labels

1. OWNED — survives a platform swap as-is.
2. BORROWED — platform-generated, but can point to owned content.
3. PLATFORM-DEPENDENT — behavior that must be locally replaced or explicitly accepted as dependency.

Memory Warning

A platform-written `MEMORY.md` or `memory/*.md` file is contaminated until reviewed. It may be truncated, stale, hallucinated, or formless. Do not treat it as owned memory just because it exists on disk.

Automation Warning

Cron jobs, heartbeats, daemon loops, and platform schedulers are conveniences, not business foundations. If revenue depends on them, document the dependency and build a fallback.

Recalibration Line

When this pattern appears, the operator can say: “Read `work/platform-sovereignty-essay.md`. Is this a sustainable mechanism, or are you leaning on platform magic?”

Say: “Platform sovereignty essay created. Use it as a context download before adopting platform magic as infrastructure.”

PHASE 2 — Create the audit protocol

Create `work/sovereignty-audit-sunrun.md`:

Portability / Survival Audit Sun Run

Purpose: Walk the operator’s boot chain, label each file/behavior, propose migrations, wait for per-file approval.

Authority Boundary

- Default is preserve.
- Agent walks, labels, proposes.
- No file edited without explicit operator approval per file.

Step 1 — Ask for the boot chain

Ask the operator:

“What boot chain or feature set do you want audited? Send the file list, or describe the agent’s startup sequence, or ask me to start from the obvious candidates (SOUL.md, AGENTS.md, IDENTITY.md, USER.md, TOOLS.md, MEMORY.md, memory/*.md, HEARTBEAT.md, WAKEY-WAKEY.md, orientation.md, beliefs.md, plus any auto-compaction, managed-memory, heartbeat, cron, scheduler, or template-regeneration behaviors you’ve noticed).”

Do not proceed until the operator names a scope.

Step 2 — Walk and label

For each file or behavior, return:

```
Subject: [file path or behavior name]
Survival question: [does it still work without platform?]
Label: [OWNED / BORROWED / PLATFORM-DEPENDENT]
Reasoning: [one sentence]
If BORROWED: proposed pointer + diff preview
If PLATFORM-DEPENDENT: proposed local replacement (or "accept
dependency" if reasonable)
```

Step 3 — Surface the inventory

Return:

```
Sovereignty Inventory — [date]
Total subjects: [number]
OWNED: [count]
BORROWED: [count]
PLATFORM-DEPENDENT: [count]
Platform-written memory files: [count]
Automation dependencies (heartbeat/cron/scheduler): [count]
High-risk dependencies: [items where platform change would
silently shift behavior or business continuity]
```

Step 4 — Ask for human review

For each subject, return the proposal and one direct question: “Migrate, replace, accept dependency, or skip?”

Wait for the operator’s decision per subject.

Step 5 — Apply approved changes (one at a time)

For each approved migration: - Show the exact change. - Wait for “go.” - Apply. - Confirm.

For each approved acceptance: - Log the dependency. - Move on.

For each skipped subject: - Defer. - Continue.

Step 6 — Tweak loop

If a migrated file behaves unexpectedly: - Re-diagnose. The pointer may resolve to the wrong content. - Refine. Propose a corrected pointer. - Wait for operator approval.

Repeat until the operator says the loop is complete.

Step 7 — Postmortem analysis

When the audit ends, write a postmortem:

Sovereignty Postmortem — [Audit Date]

- **Scope audited:**
- **OWNED count:**
- **BORROWED count:**

- **PLATFORM-DEPENDENT count:**
- **Migrations applied:**
- **Dependencies accepted:**
- **High-risk items still unresolved:**
- **Platform-written memory found:**
- **Cron/heartbeat/scheduler dependencies found:**
- **Business continuity risk:**
- **What the operator learned about the platform:**
- **Future guardrail:**

Step 8 — Install guardrail

Convert the future guardrail into one operational rule:

"Before adopting any platform-generated file, memory artifact, heartbeat, cron job, or managed platform behavior as business infrastructure, run the survival question. If it does not survive a platform swap, replace it with owned infrastructure or document it as an accepted dependency."

Say: "Sovereignty audit complete. Postmortem written. Guardrail installed."

PHASE 3 — Create the sovereignty journal

Create `work/sovereignty-journal.md`:

Sovereignty Journal

Purpose: Preserve sovereignty decisions as durable scar tissue. The journal tracks which dependencies are accepted and which were migrated.

Entry Template

[DATE] — [Subject]

- **Subject:**
- **Label:**
- **Survival question answer:**
- **Pointer migration / local replacement / accepted dependency:**
- **Memory contamination risk (if any):**
- **Automation dependency risk (if any):**
- **Operator decision:**
- **Diff applied:**
- **Post-change observation:**
- **Postmortem lesson:**
- **Guardrail installed:**

Drift Detection Rule

Once a quarter (or when the platform updates), walk the journal. Any OWNED file that now behaves differently is a sign of silent platform drift. Re-audit.

Say: “Sovereignty journal created. Quarterly drift detection is part of the architecture.”

PHASE 4 — Create the authority-boundary file

Create `work/sovereignty-authority.md`:

Sovereignty Authority Boundary

Purpose: Keep platform-dependency trades in the operator’s hands.

The Rule

- Default is preserve.
- Agent labels, proposes, inventories.
- Operator decides per subject: migrate / replace / accept / skip.
- No file is edited without explicit per-file approval.

Irreversible-Action Gate

Pointer migration is recoverable. Local replacement of a platform behavior is a larger architectural choice. Backup the original before any STRIP or REWRITE.

Say: “Authority boundary created. Sovereignty is an operator decision, per subject.”

PHASE 5 — Install the trigger

Add to operator rules:

Trigger — Sovereignty Audit

When the operator says “portability audit,” “survival audit,” “platform check,” “platform magic,” “heartbeat check,” “memory contamination,” or after any platform update, open `work/platform-sovereignty-essay.md`, `work/sovereignty-lab.md`, `work/sovereignty-audit-sunrun.md`, `work/sovereignty-journal.md`, and `work/sovereignty-authority.md`. Walk the

boot chain, memory files, and platform automation. Label each subject. Propose migrations or replacements. Wait for per-subject approval. Apply approved changes. Write the postmortem. Install one guardrail.

New-File Rule

Before adopting any platform-generated file, memory artifact, heartbeat, cron job, scheduler, or managed platform behavior as business infrastructure, run the survival question. If it does not survive a platform swap, replace it with owned infrastructure or document it as an accepted dependency.

Say: “Trigger installed. Sovereignty stays current as platform behavior shifts.”

PHASE 6 — Run the first audit prompt

Ask exactly:

“What boot chain or feature set do you want audited? Send the file list, or describe the agent’s startup sequence, or ask me to start from the obvious candidates (SOUL.md, AGENTS.md, IDENTITY.md, USER.md, TOOLS.md, MEMORY.md, memory/*.md, HEARTBEAT.md, WAKEY-WAKEY.md, orientation.md, beliefs.md, plus any auto-compaction, managed-memory, heartbeat, cron, scheduler, or template-regeneration behaviors you’ve noticed). I’ll label each one OWNED, BORROWED, or PLATFORM-DEPENDENT, propose pointer migrations or local replacements, and wait for your per-subject decision.”

Stop after asking. Wait for the operator’s scope.

PHASE 7 — Register in book-bag

Add to 🎒 book-bag.md:

🏛️ Platform Sovereignty Lab

- **What:** Operator-confirmed labeling of every boot-chain file as OWNED / BORROWED / PLATFORM-DEPENDENT.
- **Files:** `work/platform-sovereignty-essay.md`, `work/sovereignty-lab.md`, `work/sovereignty-audit-sunrun.md`, `work/sovereignty-journal.md`, `work/sovereignty-authority.md`
- **Triggers:** “portability audit,” “survival audit,” “platform check,” “platform magic,” “heartbeat check,” “memory contamination,” after any platform update
- **Authority boundary:** Agent labels and proposes; operator decides per subject
- **Irreversible-action gate:** Backup before any STRIP or REWRITE
- **Source:** JKE University — Course 207

Say: “Sovereignty lab registered. Honest accounting, not paranoia.”

PHASE 8 — Write the journal entry

Add to 🏫 school.md:

[TODAY] — JKE Course 207: Platform Sovereignty Lab Installed

What Shipped: Sovereignty essay, lab, audit sunrun, journal, authority boundary, trigger, new-file rule, first audit prompt. **Decisions Made:** Sovereignty is honest accounting and business sustainability. Owned, borrowed, platform-dependent — labeled per subject, operator decides the trade. Platform-written memory and platform automation are dependencies until reviewed. **Files Created:** `work/platform-sovereignty-essay.md`, `work/sovereignty-lab.md`, `work/sovereignty-audit-sunrun.md`, `work/sovereignty-journal.md`, `work/sovereignty-authority.md` **Files Modified:** 🎒 book-bag.md, operator rules **Source:** JKE University — Course 207

PHASE 9 — Say exactly

“Platform sovereignty lab installed. Send me your boot chain, memory files, or platform automation. I’ll label each subject OWNED, BORROWED, or PLATFORM-DEPENDENT, propose pointer migrations or local replacements, and wait for your decision per subject. The architecture is yours to the degree that you have built it. The business is sustainable to the degree that it can survive the platform changing.”

SCOPE HINT

Course 207 of 210. Level 5, Library of Instruments. Course 206 shaped the gradient column. Course 207 audits the platform column. Course 208 turns to the engine choice itself — which model fits which mode, and why “use the best model for everything” is the wrong question.

END OF PROTOCOL