

Workstation Control - Making agent actions reach the intended environment

Evidence companion · Curated September 16, 2026 · Source snapshot
ef8546aaa7a79c006f75a68da027389912b1f2d1

An agent can generate the right configuration and still install it on the wrong machine, overwrite newer work, or mistake an unrelated browser for its own session. Workstation Control addresses the operating environment around the model: named host capabilities, authenticated machine-local services, explicit destinations, observable effects and application-session identity.

Justin Sublette created and directed the system, including its architecture, integration and evaluation, with extensive Codex implementation and testing. The exhibits show that engineering contribution through concrete mechanisms. They do not infer individual line authorship from commit identity.

1. Follow one configuration delivery from agent to file

The selected operation installs a complete configuration file on a selected worker. The retained orchestration client chooses the worker and invokes the operation; Workstation Control owns the worker-local execution boundary. This separation prevents a transport service from becoming the authority deciding what research or development work an agent should pursue.

Step	Owning component	What happens
Select	Retained worker-orchestration CLI	Resolves the named worker and repository-local source; passes its source reference, content and overwrite choice.
Transport	Worker client	Sends an authenticated request to that worker's configured file-push service.
Admit	Workstation HTTP service and target registry	Requires authentication, resolves a registered target, validates the source reference and bounds the destination to local configuration files.
Apply	File target implementation	Recognizes identical bytes, refuses a conflicting replacement without explicit overwrite, then replaces the destination through a temporary file.
Observe	File service	Returns action, destination metadata, byte count and a content fingerprint; records a receipt without file contents. File readback has its own separate service.

W1 - Actual caller dispatch, cli.py:

```
if args.command == "file" and args.file_command == "put":
    target = _select_file_push_target(workers, args.worker)
    source_file = _resolve_repo_local_file_target_source(root, args.source_file)
    source_ref = source_file.relative_to(root).as_posix()
    payload = make_file_push_client(target).put_file(
        target_id=args.target,
        file_content_base64=_read_file_as_base64(source_file),
        source_ref=source_ref,
        overwrite=args.overwrite,
    )
    payload["worker"] = target.worker_id
    out.write(json.dumps(payload, indent=2) + "\n")
    return 0
```

The worker argument and registered target are separate selections. The worker service resolves its own destination; the caller does not supply an arbitrary remote destination path. The HTTP service binds to loopback and requires bearer authentication for operational routes. Transport reachability and the host's configured access route remain separate operational dependencies.

W2 - Actual content-conflict and replacement logic, targets.ts:

```
if (currentBytes && currentBytes.equals(fileContent)) {
    return {
        action: 'already_matches',
        fingerprint,
        byteCount,
    }
}
if (currentBytes && !overwrite) {
    throw new FilePushTargetMutationError(
        'overwrite_required',
        `Worker file push target already contains different bytes: ${target.targetId}`,
    )
}

await writeFileAtomic(destinationFilePath, fileContent)
```

The called replacement helper stages bytes alongside the destination and renames the completed file:

```
async function writeFileAtomic(destinationPath: string, content: Buffer): Promise<void> {
    await fs.mkdir(path.dirname(destinationPath), { recursive: true })
    const tempPath = `${destinationPath}.${process.pid}.${Date.now()}.tmp`
    try {
        await fs.writeFile(tempPath, content)
        await fs.rename(tempPath, destinationPath)
    } catch (error) {
        await fs.rm(tempPath, { force: true })
        throw error
    }
}
```

If a completed write's response is lost, resending those same bytes can return `already_matches`. That is content-based idempotence, not an exactly-once transaction across the file and receipt store. A concurrent writer, crash durability and general filesystem containment would require separate claims and evidence. This excerpt demonstrates the implemented whole-file replacement mechanism without implying those stronger properties.

W4 - Authored HTTP test assertions, server.test.ts:

```
const secondJson = (await secondInstall.json()) as { action: string }
assert.equal(secondInstall.status, 200)
assert.equal(secondJson.action, 'already_matches')
// ... request setup omitted ...
assert.equal(conflictingInstall.status, 409)
// ... request setup omitted ...
assert.equal(overwriteInstall.status, 200)
const fileContents = await fs.readFile(destinationFile, 'utf8')
assert.equal(fileContents, secondContent)
```

The full test first installs content, repeats it without overwrite, attempts different content without overwrite, then explicitly overwrites and inspects the resulting bytes. Request setup between these assertions is omitted. Other authored cases inspect authentication, invalid source references, destination policy and disclosure in responses and receipts. These are inspectable tests, not a newly executed test result. The retained system record still identifies a missing checked-in manual file-push proof; this exhibit does not present the operation as comprehensively live-qualified.

2. A responding browser is insufficient evidence of ownership

Browser automation creates a subtler version of the wrong-target problem. A debugging endpoint can respond while belonging to a process using a different profile. The session manager compares the leased browser/profile/port against actual browser roots before permitting reuse.

W5 - Actual pre-reuse check, manager.ts:

```
const existingProcesses = await this.ops.listBrowserProcesses()
const matchingRoots = matchingManagedRootProcessesForSession(existingProcesses, session)
const portRoots = managedRootProcessesForSession(existingProcesses, session)
if (matchingRoots.length > 1) {
  throw new BrowserSessionEnsureError('Managed browser root ownership is ambiguous')
}

let version = await this.ops.fetchVersion(session.cdpUrl)
if (matchingRoots.length === 0 && version.ok) {
  throw new BrowserSessionEnsureError('The responsive CDP endpoint is not owned by exactly one matching managed browser root')
}
if (matchingRoots.length === 0 && portRoots.length > 0) {
  throw new BrowserSessionEnsureError('The CDP port is occupied by a browser root that does not match the leased profile')
}
```

W6 - Authored mismatch-test outcome, manager.test.ts:

```
await assert.rejects(
  () => manager.ensureSession(leased.session.sessionId),
  /responsive CDP endpoint is not owned by exactly one matching managed browser root/i,
)
assert.equal(stopCalls, 0)
assert.equal(launchOptions, null)
```

The test supplies a responsive endpoint and a process that does not match the requested profile. It expects refusal, no browser stop and no launch. Separate authored tests cover competing port/profile leases and

require exactly one matching root after launch. A successful HTTP response from a debugging port is therefore only one input to ownership validation.

3. Historical observation changed what counted as successful delivery

The March 2026 desktop investigation found a prompt persisted in shared conversation state while the intended text was absent from the visible desktop. It recorded the failed helper outcome instead of calling persistence successful delivery. Earlier stateful smoke tests had also created synthetic conversations in normal working state, prompting dedicated sacrificial workspaces.

The resulting operating model separated shared conversation identity from visible application interaction. A later **July 2, 2026 retained live proof** used exact visible sidebar selection to deliver disposable Protocol Runner prompts, then received accepted start and completion reports. That supports a concrete delivery-and-response workflow. Seamless refresh of an already-open worker conversation remained a separate unqualified behavior. These are historical observations reported by retained records; no desktop operation was performed for this curation.

4. A related interface result, with a precise denominator

Repository-wide tool-interface work made first contact discoverable through a non-mutating usage command, structured results, semantic selectors and consistent option handling. The closing June audit recorded **81 stable tools with zero interface findings and zero probe failures**, within a catalog of 100 tools. That demonstrates a checked invocation contract across the selected population. It does not establish that every underlying operation was correct or measure agent productivity.

Together, the exhibits make the contribution inspectable: connect the agent's intended action to the correct owner, preserve explicit conflict behavior, and distinguish component activity from the outcome the user actually needed.