PRODUCT REQUIREMENTS DOCUMENT

agent-desktop

Cross-Platform Desktop Automation via Accessibility Trees for AI Agents

Version 2.0 • February 2026
Status: Draft for Engineering Review

Dual-mode: standalone CLI + MCP server
Platforms: macOS (P1), Windows + Linux (P2)
Language: Rust | License: Apache 2.0

Table of Contents

1. Executive Summary

agent-desktop is a cross-platform CLI and MCP server that enables AI agents to observe and control desktop applications through native OS accessibility trees. It is the desktop equivalent of Vercel's *agent-browser* — a tool designed for machines, not humans — outputting compact, structured, ref-tagged accessibility snapshots that LLMs can reason over and act upon.

The project addresses a critical gap: no existing tool provides unified accessibility-tree automation across Windows, macOS, and Linux with the performance and compactness required for AI agent integration. Microsoft's UFO² proves accessibility-first approaches outperform vision-only methods by 10%+, but remains Windows-only. Anthropic's Computer Use validates demand but shows the ceiling of pure-screenshot approaches.

> **Core Principle:** agent-desktop is not an AI agent. It is a tool that AI agents invoke. It outputs structured JSON with ref-based element identifiers. The observation-action loop lives in the calling agent. This separation of concerns mirrors agent-browser and is non-negotiable.

Key Differentiators

- **Accessibility-first.** Native OS APIs provide semantic structure (roles, names, states) that screenshots cannot.
- **Cross-platform in architecture from day one.** Phase 1 ships macOS; the trait-based adapter ensures Windows and Linux are additive.
- **Context-efficient.** Targets <500 tokens per snapshot via interactive-only filtering and compact serialization.
- **Dual-mode.** Single binary: standalone CLI (for AI coding agents) + MCP server (for Claude Desktop, Cursor, custom hosts).
- **Complete command surface.** 40+ commands covering the full desktop automation lifecycle: launch, observe, interact, drag, scroll, clipboard, wait.

2. Goals, Non-Goals, and Constraints

2.1 Goals

- **G1**: Sub-10ms-startup CLI outputting accessibility tree snapshots with stable ref-tagged elements.
- **G2**: Platform adapter trait in Phase 1 that accommodates Windows and Linux without refactoring core logic.
- **G3**: Support both CLI invocation and MCP server mode from a single binary.
- **G4**: Less than 500 tokens per focused-application snapshot via intelligent filtering.
- **G5**: Ship macOS first, then Windows and Linux, covering the three major desktop operating systems.
- **G6**: Typed, versioned JSON output contract that agents can rely on across releases.
- **G7**: 40+ commands covering the complete desktop automation lifecycle from launch to clipboard to drag-and-drop.

2.2 Non-Goals

- Does not embed or invoke LLMs. AI integration is the calling agent's responsibility.
- Does not provide a GUI, TUI, or interactive prompt. Machine-facing only.
- Does not automate web browsers. Use agent-browser for that.
- Does not record or replay macros. Stateless per invocation (until daemon in Phase 4).
- Does not work with custom-rendered or game-engine UIs lacking accessibility exposure.

2.3 Constraints

- **macOS TCC:** User must grant Accessibility permission. Tool must detect and guide.
- **Windows UAC:** Cross-process UIA may require elevation. Fail gracefully with reason.
- **Linux AT-SPI2:** Bus must be running. Wayland gaps are a known risk.
- **Binary size:** Target <15MB per platform. Rust static linking keeps this achievable.
- **Min OS:** macOS 12+, Windows 10 1809+, Ubuntu 22.04+ / Fedora 38+.

3. Architecture Overview

3.1 Design Principles

- **Separation of Concerns** — Tool handles observation and action. Agent handles planning and reasoning.
- **Adapter Pattern** — Rust trait defines the platform contract. Each OS implements it independently. Core logic is shared.
- **Additive Phase Model** — Phase 1 builds the complete vertical. Phases 2-4 add adapters and modes without modifying core.
- **Fail-Forward Errors** — Every error carries a machine-readable code, human message, and suggested recovery action.
- **Snapshot Stability** — Refs are deterministic within a snapshot: same UI state produces same refs.
- **Command Extensibility** — Adding a command requires one file in one directory. No modification to existing code.

3.2 System Architecture

Layer	Crate(s)	Responsibility
Entry Point	agent-desktop (bin)	CLI parsing (clap), MCP bootstrap (rmcp), mode detection
Command Registry	agent-desktop-core	Command trait, command dispatch, argument validation
Core Engine	agent-desktop-core	SnapshotEngine, RefAllocator, RefMap, JSON serialization, error types
Platform Trait	agent-desktop-core	PlatformAdapter trait, AccessibilityNode schema, Action enum
macOS Adapter	agent-desktop-macos	AXUIElement traversal, action execution, permission detection
Windows Adapter	agent-desktop-windows	UIA traversal via uiautomation crate (Phase 2)
Linux Adapter	agent-desktop-linux	AT-SPI2 via atspi + zbus crates (Phase 2)
MCP Layer	agent-desktop-mcp	MCP tool definitions, JSON-RPC handling (Phase 3)

3.3 Data Model: AccessibilityNode

Every element on every platform normalizes to this schema (defined in `agent-desktop-core`):

Field	Type	Description
ref	Option<String>	Ref ID (@e1). Only assigned to interactive elements.
role	String	Normalized role: button, textfield, checkbox, link, menuitem, tab, slider, combobox, table, image, statictext, group, window, toolbar, etc.
name	Option<String>	Accessible name / label.

Field	Type	Description
value	Option<String>	Current value (text content, slider position, etc.).
description	Option<String>	Accessible description if distinct from name.
states	Vec<String>	Active states: focused, selected, expanded, checked, disabled, required, readonly, pressed, hovered.
bounds	Option<Rect>	Bounding rectangle {x, y, width, height} in screen coords.
children	Vec<AccessibilityNode>	Child nodes. Leaf nodes have empty array.

3.4 Ref System Design

- Refs allocated in document order (depth-first): `@e1`, `@e2`, etc.
- Only interactive roles receive refs: button, textfield, checkbox, link, menuitem, tab, slider, combobox, treeitem, cell.
- Static text, groups, containers do NOT get refs but remain in the tree for context.
- RefMap maps each ref to: native handle, role, name, bounds, available actions.
- Refs are deterministic within a snapshot. NOT stable across snapshots if UI changed.

3.5 CLI and MCP Dual-Mode Entry Point

- If invoked with `--mcp` or stdin is a pipe: enter MCP server mode.
- Otherwise: parse CLI arguments, execute command, print JSON to stdout.

> **Invariant:** Every MCP tool maps 1:1 to a CLI command. `agent-desktop snapshot --app Finder` is identical to invoking the MCP `desktop_snapshot` tool. Testing, debugging, and documentation are never fragmented.

4. Coding Standards and Project Structure

These standards are non-negotiable. They ensure the codebase remains readable, modular, and extensible as the project scales from macOS-only to three platforms with 40+ commands.

4.1 Workspace Layout

```
agent-desktop/
Cargo.toml # workspace: members, shared deps
rust-toolchain.toml # pinned Rust version
clippy.toml # project-wide lint config
schemas/ # JSON Schema files for output validation
snapshot_response.json | action_response.json | error_response.json
crates/
core/ src/ # agent-desktop-core
lib.rs # public re-exports only
node.rs # AccessibilityNode, Rect, WindowInfo
adapter.rs # PlatformAdapter trait
action.rs # Action enum, ActionResult, InputEvent, WindowOp
refs.rs # RefAllocator, RefMap, RefEntry
snapshot.rs # SnapshotEngine (filter, allocate, serialize)
error.rs # ErrorCode enum, AdapterError, AppError
output.rs # Response envelope, JSON formatting
command.rs # Command trait + CommandRegistry
commands/ # one file per command (see below)
mod.rs # register_all() auto-registration
snapshot.rs | click.rs | type_text.rs | press_key.rs | find.rs
list_windows.rs | focus_window.rs | launch.rs | close_app.rs
screenshot.rs | scroll.rs | drag.rs | hover.rs | get.rs
is_check.rs | select.rs | toggle.rs | expand.rs | set_value.rs
right_click.rs | double_click.rs | mouse.rs | clipboard.rs
wait.rs | resize_window.rs | move_window.rs | minimize.rs
maximize.rs | list_apps.rs | status.rs | permissions.rs
```

```
crates/ (continued)
macos/ src/ # agent-desktop-macos
lib.rs | adapter.rs # MacOSAdapter: PlatformAdapter impl
tree.rs # AXUIElement tree traversal
actions.rs # AXPress, AXValue, CGEvent dispatch
permissions.rs # AXIsProcessTrusted, TCC guidance
roles.rs # AXRole to unified role mapping
input.rs # CGEvent keyboard/mouse synthesis
screenshot.rs # CGWindowListCreateImage
windows/ src/ # agent-desktop-windows (Phase 2)
lib.rs | adapter.rs | tree.rs | actions.rs | roles.rs | input.rs | screenshot.rs
linux/ src/ # agent-desktop-linux (Phase 2)
lib.rs | adapter.rs | tree.rs | actions.rs | roles.rs | input.rs | screenshot.rs
mcp/ src/ # agent-desktop-mcp (Phase 3)
lib.rs | server.rs | tools.rs | transport.rs
src/
main.rs # entry point: mode detection, dispatch
cli.rs # clap derive structs
tests/integration/
macos_snapshot.rs | macos_actions.rs | cross_platform.rs
```

4.2 File and Module Rules

- **400 LOC hard limit per file.** If a file approaches 400 lines, split by responsibility. No exceptions.
- **No inline comments.** Code must be self-documenting through naming. Only Rust doc-comments (///) on public items when the name alone is insufficient.
- **One struct/enum per file** for domain types. node.rs defines AccessibilityNode. action.rs defines Action. Never bundle unrelated types.
- **One command per file.** Each CLI command lives in its own file under commands/. The file name matches the command name.
- **No God objects.** No struct with more than 7 fields. No function with more than 5 parameters. Introduce builder patterns or config structs instead.
- **Explicit pub boundaries.** Only lib.rs re-exports public items. Internal modules use pub(crate). No wildcard re-exports.
- **Zero unwrap() in non-test code.** All Results are propagated with ? or matched explicitly. Panics are test-only.
- **Platform code isolation.** Core crate never imports platform crates. Platform crates never import each other. Communication is via the PlatformAdapter trait only.

4.3 Naming Conventions

Element	Convention	Example
Crate names	agent-desktop-{name}	agent-desktop-core, agent-desktop-macos
Module files	snake_case, singular	snapshot.rs, list_windows.rs
Structs	PascalCase, descriptive noun	SnapshotEngine, RefAllocator, MacOSAdapter
Traits	PascalCase, adjective or capability	PlatformAdapter, Executable
Enums	PascalCase, variants are PascalCase	Action::Click, ErrorCode::PermDenied
Functions	snake_case, verb-first	build_tree(), allocate_refs(), execute_action()
Constants	SCREAMING_SNAKE_CASE	MAX_TREE_DEPTH, DEFAULT_TIMEOUT_MS
CLI flags	kebab-case	--max-depth, --include-bounds, --output-file
Ref IDs	@e{n} sequential	@e1, @e2, @e14

4.4 Error Handling Pattern

Every error type in the project implements this pattern:

- `ErrorCode` enum — Machine-readable identifier (PERM_DENIED, ELEMENT_NOT_FOUND, ACTION_FAILED, etc.).
- `message: String` — Human-readable description of what went wrong.

- `suggestion: Option` — Actionable recovery hint for both humans and AI agents.
- `platform_detail: Option` — OS-specific detail (e.g., AXError code, HRESULT, D-Bus error).

All functions in platform adapters return `Result`. All command handlers return `Result`. The binary's main() converts AppError to JSON and sets the exit code.

4.5 Adding a New Command (Extensibility Pattern)

To add a new command (e.g., `minimize`), a developer performs exactly these steps:

- **1.** Create `crates/core/src/commands/minimize.rs` implementing the `Command` trait.
- **2.** Register it in `crates/core/src/commands/mod.rs` inside `register_all()`.
- **3.** Add the CLI subcommand variant to `src/cli.rs` (clap derive enum).
- **4.** If new Action variant needed, add it to `crates/core/src/action.rs`.
- **5.** If new adapter method needed, add it to `PlatformAdapter` trait with a default that returns `Err(AdapterError::not_supported())`.

No existing files are modified beyond the two registration points (mod.rs and cli.rs). This is enforced by code review.

5. Complete Command Reference

This is the full command surface for agent-desktop, covering the complete desktop automation lifecycle. Commands marked **P1** ship in Phase 1 (macOS). All commands ship by end of Phase 2. The design ensures any future command follows the extensibility pattern in Section 4.5.

> **Design mapping from agent-browser:** agent-browser has 80+ browser commands. agent-desktop maps these to desktop equivalents where applicable, and adds desktop-specific commands (window management, app lifecycle, clipboard, drag-and-drop with accessibility semantics) that have no browser analog.

5.1 App and Window Management

Command	Description	Key Flags	Phase
launch <app>	Start application by name, bundle ID, or path	--wait (block until window appears)	P1
close-app <app>	Quit an application gracefully	--force (SIGKILL / TerminateProcess)	P1
list-windows	List all visible application windows	--focused-only, --app <name>	P1
list-apps	List all running applications with PIDs	--with-windows	P1
focus-window	Bring window to foreground / activate	--app <name>, --window <id>, --title <pattern>	P1
resize-window <w> <h>	Resize the target window	--app, --window	P2
move-window <x> <y>	Move window to screen coordinates	--app, --window	P2
minimize	Minimize the target window	--app, --window	P2
maximize	Maximize / zoom the target window	--app, --window	P2
restore	Restore minimized window to normal	--app, --window	P2

5.2 Observation and Inspection

Command	Description	Key Flags	Phase
snapshot	Capture accessibility tree with ref-tagged interactive elements	--app, --window, --max-depth <n>, --include-bounds, --interactive-only (-i), --compact (-c)	P1

Command	Description	Key Flags	Phase
screenshot [path]	Capture screen, window, or element image	--window <id>, --screen <n>, --element <ref>, --format png|jpg, --quality <n>	P1
find <query>	Search tree for elements matching name, role, or value	--role <role>, --exact, --limit <n>	P1
get text <ref>	Get the accessible name / text content of element		P1
get value <ref>	Get the current value (input text, slider pos, etc.)		P1
get title	Get the title of the focused window	--app, --window	P1
get bounds <ref>	Get bounding rectangle of element		P1
get role <ref>	Get the accessibility role of element		P1
get states <ref>	Get active states (focused, expanded, checked, etc.)		P1
get tree-stats	Get summary: total nodes, interactive count, depth	--app, --window	P2
is visible <ref>	Check if element is on-screen and not hidden		P1
is enabled <ref>	Check if element is interactive (not disabled)		P1
is checked <ref>	Check if checkbox/toggle is checked		P1
is focused <ref>	Check if element has keyboard focus		P1
is expanded <ref>	Check if tree item / disclosure is expanded		P1

5.3 Element Interaction

Command	Description	Key Flags	Phase
click <ref>	Click element via accessibility action (AXPress / Invoke)		P1
double-click <ref>	Double-click element		P1
right-click <ref>	Right-click / context-click element		P1
type <ref> <text>	Type text keystroke-by-keystroke into focused element	--delay <ms> (inter-key delay)	P1
set-value <ref> <text>	Set element value via accessibility API directly	--clear-first	P1
focus <ref>	Move keyboard focus to element		P1
select <ref> <value>	Select option in dropdown / combobox / list	--by-index <n>	P1
toggle <ref>	Toggle checkbox, switch, or radio button		P1
expand <ref>	Expand tree node, disclosure, or collapsible section		P1
collapse <ref>	Collapse expanded tree node or section		P1
hover <ref>	Move mouse cursor over element (triggers hover states)		P2
clear <ref>	Clear text content of input / text field		P2

5.4 Keyboard and Mouse Control

Command	Description	Key Flags	Phase
press <keys>	Press key combination (e.g., cmd+c, ctrl+shift+s, alt+tab, enter)		P1
key-down <key>	Hold key down (modifier hold for multi-action sequences)		P2
key-up <key>	Release held key		P2
mouse move <x> <y>	Move mouse to absolute screen coordinates	--relative (delta instead of absolute)	P2
mouse click <x> <y>	Click at coordinates (bypasses accessibility)	--button left|right|middle, --count <n>	P2
mouse down [button]	Press and hold mouse button	--button left|right|middle	P2

Command	Description	Key Flags	Phase
mouse up [button]	Release mouse button	--button left|right|middle	P2
mouse wheel <dy> [dx]	Scroll mouse wheel (pixel delta)		P2

5.5 Scroll, Drag, and Gesture

Command	Description	Key Flags	Phase
scroll <direction>	Scroll up/down/left/right within element or window	--ref <ref> (scroll within element), --amount <px>, --page (scroll by page)	P1
drag <from> <to>	Drag from ref/coords to ref/coords	--from-ref <ref>, --to-ref <ref>, --from-xy <x,y>, --to-xy <x,y>	P2
pinch <scale>	Pinch zoom gesture (macOS trackpad)	--center-x, --center-y	P2

5.6 Clipboard

Command	Description	Key Flags	Phase
clipboard get	Read current clipboard text content		P1
clipboard set <text>	Write text to system clipboard		P1
clipboard clear	Clear the system clipboard		P2
clipboard has-image	Check if clipboard contains image data		P2

5.7 Wait and Polling

Command	Description	Key Flags	Phase
wait <ms>	Wait for specified milliseconds		P1
wait --element <ref>	Wait until element exists and is visible	--timeout <ms> (default 5000)	P1
wait --window <title>	Wait until a window with title pattern appears	--timeout <ms>, --app <name>	P1
wait --gone <ref>	Wait until element disappears from tree	--timeout <ms>	P2
wait --value <ref> <val>	Wait until element's value matches expected	--timeout <ms>	P2
wait --focused <ref>	Wait until element receives focus	--timeout <ms>	P2

5.8 System and Session

Command	Description	Key Flags	Phase
status	Show daemon status, platform, permission state		P1
permissions	Check and report accessibility permission status	--request (prompt for permission on macOS)	P1
version	Print version and platform info	--json	P1
session list	List active daemon sessions (Phase 4)		P4
session kill <id>	Terminate a specific daemon session		P4

Command count summary: 40+ commands across 8 categories. Phase 1 ships 30 commands covering the complete observation-interaction loop. Phase 2 adds mouse/drag/window-geometry/hover commands. Phase 4 adds session management. Future commands follow Section 4.5 extensibility pattern: one file, two registration lines, zero existing code changes.

PHASE 1

6. Phase 1 — Foundation + macOS MVP

Phase 1 is the load-bearing phase. It establishes every shared abstraction, every trait boundary, every output contract, every error type, the complete command trait and registry, and the full workspace structure. Phases 2-4 are strictly additive: new trait implementations, new transport, new optimizations. Nothing in core is rebuilt.

6.1 Objectives and Success Criteria

ID	Objective	Success Metric
P1-O1	Working macOS snapshot CLI	snapshot --app Finder returns valid JSON with refs for all interactive elements
P1-O2	Platform adapter trait	Trait compiles with mock adapter; macOS adapter satisfies all trait methods
P1-O3	Ref-based interaction	click @e3 successfully invokes AXPress on the resolved element
P1-O4	Context efficiency	Typical Finder snapshot < 500 tokens (measured via tiktoken)
P1-O5	Typed JSON contract	Output validates against JSON Schema; schema is versioned
P1-O6	Permission detection	Missing Accessibility permission prints specific macOS setup instructions
P1-O7	Command extensibility	Adding a new command requires exactly 1 new file + 2 registration lines
P1-O8	30 working commands	All P1-scoped commands from Section 5 pass integration tests
P1-O9	CI pipeline	GitHub Actions macOS runner executes full test suite on every PR

6.2 Platform Adapter Trait

The single most important abstraction. Every platform-specific operation goes through this trait. Core never imports platform crates.

```rust
pub trait PlatformAdapter: Send + Sync {
  fn list_windows(&self, filter: &WindowFilter) -> Result<Vec<WindowInfo>>;
  fn list_apps(&self) -> Result<Vec<AppInfo>>;
  fn get_tree(&self, win: &WindowInfo, opts: &TreeOptions) -> Result<AccessibilityNode>;
  fn execute_action(&self, handle: &NativeHandle, action: Action) -> Result<ActionResult>;
  fn check_permissions(&self) -> PermissionStatus;
  fn focus_window(&self, win: &WindowInfo) -> Result<()>;
  fn launch_app(&self, id: &str, wait: bool) -> Result<WindowInfo>;
  fn close_app(&self, id: &str, force: bool) -> Result<()>;
  fn screenshot(&self, target: ScreenshotTarget) -> Result<ImageBuffer>;
  fn get_clipboard(&self) -> Result<String>;
  fn set_clipboard(&self, text: &str) -> Result<()>;
  fn synthesize_input(&self, input: InputEvent) -> Result<()>;
  fn manage_window(&self, win: &WindowInfo, op: WindowOp) -> Result<()>;
}
```

Key Supporting Types

- `Action` — Click, DoubleClick, RightClick, SetValue(String), SetFocus, Expand, Collapse, Select(String), Toggle, Scroll(Direction, Amount), PressKey(KeyCombo).
- `InputEvent` — MouseMove(x,y), MouseClick(x,y,button,count), MouseDown(button), MouseUp(button), MouseWheel(dy,dx), KeyDown(key), KeyUp(key), Drag(from,to).
- `WindowOp` — Resize(w,h), Move(x,y), Minimize, Maximize, Restore, Close.
- `ScreenshotTarget` — Screen(index), Window(id), Element(NativeHandle), FullScreen.

6.3 macOS Adapter Implementation

Tree Traversal

- Entry: `AXUIElementCreateApplication(pid)` for app root.
- Children: `kAXChildrenAttribute` recursively with visited-set to prevent cycles.
- Role mapping: AXRole strings mapped to unified role enum in `roles.rs`.
- Name: `kAXTitleAttribute` / `kAXDescriptionAttribute`. Value: `kAXValueAttribute`.
- Bounds: `kAXPositionAttribute` + `kAXSizeAttribute` combined to Rect.

Action Execution

- **Click:** `AXUIElementPerformAction(kAXPressAction)`.
- **SetValue:** `AXUIElementSetAttributeValue(kAXValueAttribute, value)`.
- **SetFocus:** `AXUIElementSetAttributeValue(kAXFocusedAttribute, true)`.
- **Expand/Collapse:** Toggle `kAXExpandedAttribute`.
- **Select:** `AXUIElementSetAttributeValue(kAXSelectedAttribute, true)` on child.
- **Keyboard/Mouse:** `CGEventCreateKeyboardEvent` / `CGEventCreateMouseEvent` via CoreGraphics.
- **Clipboard:** `NSPasteboard.generalPasteboard` read/write via Cocoa FFI.
- **Screenshot:** `CGWindowListCreateImage` for window-specific or full-screen capture.

Permission Detection

- Call `AXIsProcessTrusted()` on startup. If false, return PERM_DENIED with guidance.
- Optionally call `AXIsProcessTrustedWithOptions(prompt: true)` to trigger system dialog.

6.4 Snapshot Engine and Ref Allocator

Platform-agnostic. Lives in `agent-desktop-core`. Takes raw tree from adapter, applies filtering and ref allocation, produces final output.

Processing Pipeline

- **1. Raw tree:** Call `adapter.get_tree(window, opts)`.
- **2. Filter:** Remove invisible/offscreen. Remove empty groups with no interactive descendants. Prune beyond max_depth.
- **3. Allocate refs:** Depth-first. Interactive roles get @e1, @e2, etc. Store in RefMap.
- **4. Serialize:** Omit null fields. Omit empty arrays. Omit bounds in compact mode.
- **5. Estimate tokens:** Optionally warn if exceeding threshold.

RefMap Persistence

In CLI mode, RefMap is written to `~/.agent-desktop/last_refmap.json` after each snapshot. Action commands read this to resolve refs. In Phase 4, the daemon holds RefMap in memory.

6.5 Phase 1 Command Scope

Phase 1 ships 30 commands. This is the complete observation-interaction loop required for an AI agent to control any macOS application:

Category	Commands (30 total)	Count
App / Window	launch, close-app, list-windows, list-apps, focus-window	5
Observation	snapshot, screenshot, find, get (text, value, title, bounds, role, states), is (visible, enabled, checked, focused, expanded)	15
Interaction	click, double-click, right-click, type, set-value, focus, select, toggle, expand, collapse, scroll	11
Keyboard	press	1
Clipboard	clipboard get, clipboard set	2
Wait	wait (ms), wait --element, wait --window	3
System	status, permissions, version	3

Phase 2 adds 10+ commands: hover, clear, drag, mouse (move/click/down/up/wheel), key-down/up, resize/move/minimize/maximize/restore window, pinch, wait --gone/--value/--focused, get tree-stats, clipboard clear/has-image.

6.6 JSON Output Contract

All commands produce a response envelope. Schema files are versioned in `schemas/`.

```
{
  "version": "1.0",
  "ok": true,
  "command": "snapshot",
  "app": "Finder", "window": { "id": "w-4521", "title": "Documents" },
  "ref_count": 14,
  "tree": {
    "role": "window", "name": "Documents", "children": [
      { "role": "toolbar", "children": [
        { "ref": "@e1", "role": "button", "name": "Back" },
        { "ref": "@e2", "role": "button", "name": "Forward" }
      ]},
      { "ref": "@e3", "role": "textfield", "name": "Search", "value": "" }
    ]
  }
}
```

6.7 Error Taxonomy

Code	Category	Example	Recovery Suggestion
PERM_DENIED	Permission	Accessibility not granted	Open System Settings > Privacy > Accessibility and add your terminal
ELEMENT_NOT_FOUND	Ref	@e12 not in current RefMap	Run 'snapshot' to refresh, then retry with updated ref
APP_NOT_FOUND	Application	--app 'Photoshop' not running	Launch the application first with 'launch Photoshop'
ACTION_FAILED	Execution	AXPress returned error on disabled button	Element may be disabled. Check states before acting
ACTION_NOT_SUPPORTED	Execution	Expand on a button element	This element does not support the requested action
TREE_TIMEOUT	Performance	Traversal exceeded 5s	Try --max-depth 3 or target a specific window
STALE_REF	Ref	RefMap is from a previous snapshot	UI may have changed. Run 'snapshot' again
WINDOW_NOT_FOUND	Window	--window w-999 does not exist	Run 'list-windows' to see available windows
PLATFORM_UNSUPPORTED	Platform	Linux adapter not yet shipped	This platform ships in Phase 2. Currently macOS only
CLIPBOARD_EMPTY	Clipboard	clipboard get but clipboard is empty	No text content in clipboard. Copy something first
TIMEOUT	Wait	wait --element exceeded timeout	Element did not appear within timeout. Increase --timeout or check app state

6.8 Testing Plan

Unit Tests (core)

- AccessibilityNode ser/de roundtrips. Ref allocator only assigns interactive roles. SnapshotEngine filtering. Error serialization. JSON schema validation.

Unit Tests (macos)

- Role mapping coverage. Permission check with mocks. Tree traversal cycle detection.

Integration Tests (macOS CI)

- Snapshot Finder, TextEdit, System Settings — non-empty trees with refs.
- Click button in test app — verify action succeeded.
- Type text into TextEdit via ref — verify content changed.

- Clipboard get/set roundtrip. Wait for window. Launch + close app lifecycle.
- Permission denied scenario — correct error code and guidance.
- Large tree (Xcode) snapshot in under 2 seconds.

6.9 Deliverables and Timeline

Duration: 10 weeks.

Week	Milestone	Deliverable
1-2	Scaffold + core types	Workspace, crate stubs, AccessibilityNode, PlatformAdapter trait, Action enum, error types, Command trait + registry, JSON schemas
3-4	macOS tree traversal	MacOSAdapter.get_tree() producing full trees. Role mapping. Permission detection.
5-6	Snapshot engine + ref system	SnapshotEngine filtering, RefAllocator, RefMap persistence, compact serialization
6-7	Core commands (snapshot, click, type, find)	CLI via clap. snapshot, click, type, set-value, press, find, get, is commands working
7-8	App/window + remaining commands	launch, close-app, list-windows, list-apps, focus-window, select, toggle, expand, collapse, scroll, screenshot
8-9	Clipboard, wait, system commands	clipboard get/set, wait commands, status, permissions, version
9-10	Testing + CI + polish	Full test suite, GitHub Actions macOS CI, docs, binary builds via cargo-dist

PHASE 2

7. Phase 2 — Cross-Platform Expansion

Phase 2 is purely additive. Core engine, CLI parser, JSON contract, error types, snapshot engine, and command registry are untouched. Only new PlatformAdapter implementations and Phase 2 commands are added.

7.1 Objectives

ID	Objective	Metric
P2-O1	Windows adapter	snapshot on Windows returns valid tree for Explorer, Notepad, Settings
P2-O2	Linux adapter	snapshot on Ubuntu GNOME returns valid tree for Files, Terminal, Settings
P2-O3	All commands cross-platform	Identical JSON schema output on all 3 platforms
P2-O4	Phase 2 commands ship	hover, drag, mouse, key-down/up, window geometry, advanced waits all working
P2-O5	Cross-platform CI	GitHub Actions matrix: macOS + Windows + Ubuntu

7.2 Windows Adapter

- **Crate:** `uiautomation` (v0.24+) wrapping UIA COM APIs via `windows` crate.
- **Tree:** `IUIAutomationTreeWalker` with `CacheRequest` for batch attribute retrieval.
- **Actions:** Pattern-based: `InvokePattern.Invoke()`, `ValuePattern.SetValue()`, `ExpandCollapsePattern`.
- **Input:** `SendInput` API for keyboard/mouse synthesis.
- **Clipboard:** `OpenClipboard` / `GetClipboardData` Win32 APIs.
- **Chromium:** Detect and warn about `--force-renderer-accessibility`.

7.3 Linux Adapter

- **Crate:** `atspi` (v0.28+) via `zbus` — pure Rust, no libatspi/GLib dependency.
- **Tree:** Async D-Bus calls to `org.a11y.atspi.Accessible.GetChildren`.
- **Actions:** `org.a11y.atspi.Action.DoAction` preferred over coordinate-based input.
- **Input:** `xdotool` / `ydotool` shelling for keyboard/mouse on X11/Wayland respectively.
- **Clipboard:** `wl-clipboard` (Wayland) / `xclip` (X11).
- **Bus detection:** Check for AT-SPI2 bus. Return PLATFORM_UNSUPPORTED with enable instructions if missing.

7.4 Screenshot Capture

- macOS: `CGWindowListCreateImage` or `xcap` crate.
- Windows: `BitBlt` / `PrintWindow` or `xcap`.
- Linux: PipeWire ScreenCast portal (Wayland) / `XGetImage` (X11).

7.5 Deliverables and Timeline

Duration: 10 weeks.

Week	Milestone
1-3	Windows adapter: tree, roles, actions, CacheRequest optimization, input synthesis
3-5	Linux adapter: D-Bus connection, tree, roles, actions, input synthesis
5-6	Screenshot + clipboard on all 3 platforms
6-8	Phase 2 commands: hover, drag, mouse, key-down/up, window geometry, advanced waits
8-10	Cross-platform CI, binary distribution, edge cases, Chromium detection

PHASE 3

8. Phase 3 — MCP Server Mode

Phase 3 adds a new I/O layer. Core engine and platform adapters unchanged. MCP server wraps existing command logic in JSON-RPC tool definitions.

8.1 Objectives

ID	Objective	Metric
P3-O1	MCP server mode via --mcp	Responds to MCP initialize handshake
P3-O2	All commands as MCP tools	tools/list returns all tools with JSON Schema specs
P3-O3	Claude Desktop validated	Claude Desktop invokes tools to control desktop apps end-to-end
P3-O4	Tool annotations	readOnlyHint, destructiveHint, idempotentHint on every tool

8.2 MCP Tool Surface

Each MCP tool maps 1:1 to a CLI command. Tool names are prefixed with `desktop_` to avoid collision with other MCP servers.

MCP Tool	CLI Equivalent	Annotations
desktop_snapshot	snapshot	readOnly: true
desktop_click	click <ref>	readOnly: false, destructive: false
desktop_type_text	type <ref> <text>	readOnly: false
desktop_set_value	set-value <ref> <text>	readOnly: false
desktop_press_key	press <keys>	readOnly: false
desktop_find	find <query>	readOnly: true
desktop_list_windows	list-windows	readOnly: true
desktop_focus_window	focus-window	readOnly: false
desktop_launch_app	launch <app>	readOnly: false
desktop_close_app	close-app <app>	readOnly: false, destructive: true
desktop_screenshot	screenshot	readOnly: true

MCP Tool	CLI Equivalent	Annotations
desktop_scroll	scroll <dir>	readOnly: false
desktop_drag	drag <from> <to>	readOnly: false
desktop_select	select <ref> <val>	readOnly: false
desktop_toggle	toggle <ref>	readOnly: false
desktop_clipboard_get	clipboard get	readOnly: true
desktop_clipboard_set	clipboard set <text>	readOnly: false
desktop_wait	wait	readOnly: true
desktop_get	get <prop> <ref>	readOnly: true
desktop_is	is <state> <ref>	readOnly: true

8.3 Transport and Session

- **Stdio (primary):** MCP host spawns `agent-desktop --mcp` as child process. JSON-RPC over stdin/stdout.
- **HTTP+SSE (optional):** For remote scenarios. Additive, non-blocking for core milestone.
- **Session:** On `initialize`, detect platform, check permissions, report capabilities. RefMap is session-scoped.

8.4 Deliverables and Timeline

Duration: 6 weeks.

Week	Milestone
1-2	rmcp integration: #[tool] macro definitions, initialize handler, capabilities
2-3	Stdio transport: full MCP compliance, tool invocation routing to command handlers
3-4	Claude Desktop testing: end-to-end validation, protocol edge cases
4-6	HTTP+SSE (optional), documentation, MCP config examples for Claude Desktop + Cursor

PHASE 4

9. Phase 4 — Production Hardening

Phase 4 transforms agent-desktop from functional to enterprise-grade. Persistent daemon, session isolation, and comprehensive quality gates.

9.1 Objectives

ID	Objective	Metric
P4-O1	Persistent daemon	Warm snapshot completes in <50ms (vs 200ms+ cold)
P4-O2	Session isolation	Two agents hold independent RefMaps without interference
P4-O3	Enterprise quality gates	All gates in 9.3 pass
P4-O4	Package manager distribution	brew, winget/scoop, snap/apt

9.2 Daemon Architecture

- **Auto-start:** CLI detects if daemon is running (socket file). Spawns if not.
- **Auto-stop:** Exits after configurable idle timeout (default 5 min).
- **Session multiplexing:** Each CLI/MCP session gets unique ID. RefMaps are session-scoped.
- **Health check:** `agent-desktop status` returns daemon PID, uptime, active sessions.

9.3 Enterprise Quality Gates

Gate	Requirement
Security	No arbitrary code execution. No privilege escalation. All actions allowlisted via Action enum. No network access.
Performance	Cold <200ms. Warm snapshot <50ms. Tree timeout 5s default, configurable.
Reliability	Zero panics. Graceful daemon recovery. Stale socket cleanup.
Observability	Structured logging (tracing crate). --verbose flag. Timing metrics per operation.
Compatibility	Tested: Finder, TextEdit, Xcode, VS Code, Chrome (macOS); Explorer, Notepad, Settings, VS Code (Win); Nautilus, Terminal, Firefox (Linux).
Distribution	Single binary per platform. No runtime deps. Reproducible builds. SHA256 checksums.
Documentation	README, CLI reference, MCP reference, per-platform setup guides, troubleshooting.

9.4 Deliverables and Timeline

Duration: 8 weeks.

Week	Milestone
1-2	Daemon: socket/pipe server, session management, auto-start/stop
2-3	CLI-to-daemon migration: route commands through daemon when available
3-4	Performance: CacheRequest batching (Win), async tree walking (Linux), caching
4-5	Quality gates: security audit, performance benchmarks, reliability testing
5-6	Package distribution: brew formula, winget manifest, snap package
6-8	Documentation, test matrix across target apps, RC testing

10. Risk Register

ID	Risk	L	I	Mitigation
R1	macOS TCC friction deters adoption	H	H	Clear first-run guidance. Detect before any op. One-command setup script.
R2	Electron/Chrome no a11y tree by default	H	M	Detect Chromium windows. Print --force-renderer-accessibility guidance.
R3	Custom-rendered UIs invisible to a11y	M	H	Phase 4 stretch: vision fallback. Short-term: document limitation.
R4	Wayland a11y gaps	M	M	Focus on GNOME. Prefer AT-SPI actions over coords. Document gaps.
R5	Rust a11y crate maintenance stalls	L	H	Fork-ready: pin versions, maintain patches. atspi backed by Odilia.
R6	MCP spec changes break compat	L	M	Pin rmcp version. Monitor spec under Linux Foundation governance.
R7	Tree traversal too slow (>5s)	M	M	Depth limiting. Focused-window-only. Cached subtrees in daemon.
R8	Ref instability confuses agents	M	H	Clear docs: refs are snapshot-scoped. Error on stale refs. Stable hashing in P4.

11. Appendix: Technology Reference

11.1 Core Dependencies

Crate	Version	Purpose	License
clap	4.x	CLI parsing with derive macros	MIT/Apache-2.0
serde + serde_json	1.x	JSON serialization	MIT/Apache-2.0
tokio	1.x	Async runtime (atspi, rmcp)	MIT
rmcp	0.8+	Official MCP Rust SDK (Phase 3)	MIT/Apache-2.0
accessibility-sys	0.1+	macOS AXUIElement FFI	MIT
uiautomation	0.24+	Windows UIA wrapper (Phase 2)	Apache-2.0
atspi + zbus	0.28+ / 5.x	Linux AT-SPI2 client (Phase 2)	MIT/Apache-2.0
tracing	0.1+	Structured logging	MIT
thiserror	2.x	Error derive macros	MIT/Apache-2.0
base64	0.22+	Screenshot encoding	MIT/Apache-2.0

11.2 Platform API Quick Reference

Capability	macOS	Windows	Linux
Tree root	AXUIElementCreateApp(pid)	IUIAutomation.ElementFromHandle()	atspi Accessible on bus
Children	kAXChildrenAttribute	TreeWalker.GetFirstChild	GetChildren D-Bus
Click	AXPress	InvokePattern.Invoke()	Action.DoAction(0)
Set text	AXValue = val	ValuePattern.SetValue()	Text.InsertText
Keyboard	CGEventCreateKeyboard	SendInput	xdotool / ydotool
Clipboard	NSPasteboard	Win32 Clipboard API	wl-clipboard / xclip
Screenshot	CGWindowListCreateImage	BitBlt / PrintWindow	PipeWire / XGetImage
Permissions	AXIsProcessTrusted()	COM security / UAC	Bus availability

11.3 Full Timeline Summary

Phase	Weeks	Outcome	Platforms
P1: Foundation + macOS	10	30 commands, core engine, macOS adapter, JSON contract, CI	macOS
P2: Cross-Platform	10	Windows + Linux adapters, 10+ new commands, screenshot, cross-platform CI	All
P3: MCP Server	6	Dual CLI+MCP binary, stdio transport, Claude Desktop validated	All
P4: Hardening	8	Daemon, sessions, package managers, enterprise quality	All

Total: 34 weeks (~8.5 months), one dedicated team. Phases may overlap with parallel adapter work.

End of document. Versioned alongside codebase. Amendments tracked in CHANGELOG.md.