Engineering Trustworthy MCP Integrations

Authority, State Identity, and Failure Recovery in a Development Workbench

Brian K. Holman
BS, Computer Science, Brigham Young University
MBA, Marriott School of Management, Brigham Young University
me@brianholman.com
VCC Plus Technical Papers · November 18, 2026

Abstract

Model-facing tools become difficult to trust when they act on applications whose documents, builds, processes, and permissions change during a conversation. A syntactically valid request may target an obsolete artifact; a correct memory address may identify the wrong physical storage; and a failed reply may conceal a completed mutation. I present an engineering case study of Model Context Protocol (MCP) integration in VCC Plus, a native development workbench with an emulated target computer. The design binds authority to explicit documents and projects, captures content-based build inputs, identifies executable artifacts and owned runs, and distinguishes immutable historical observations from interpretations requiring a current verified stop. Mutations use bounded execution and request receipts; disk exchange retains the previous medium before replacement and distinguishes refusal, acknowledged commitment, and an uncertain outcome. A client-facing protocol adapter and the application independently validate requests, while asynchronous completion rechecks the authority that justified starting the work. I draw on source inspection, real guest playtests, and deliberate failure fixtures. The recorded media-exchange results distinguish refusal before commitment from a completed exchange whose ordinary reply the application withholds. The resulting design lessons apply to development tools, editors, simulators, and other applications in which a useful answer depends on both permission and state identity.

Keywords: Model Context Protocol; tool-using assistants; authorization; asynchronous state; immutable observations; idempotency; debugging; failure recovery.

1. Introduction

A tool interface can make an application available to a model without making the application's answers dependable. The model may supply a well-formed request for a document that is no longer selected. A compiler may finish successfully after its inputs have changed. A debugger may recognize a source name whose executable bytes no longer match the running process. A retry may repeat input that the target already consumed. These failures arise at the boundaries among authority, state, and time. A JSON schema alone cannot resolve them.

The difficulty becomes visible in a development workbench. Editing, compiling, running, observing, and retaining a program are related activities, but they have different consequences. Reading an approved source file is different from accepting a proposed change. Building a reviewed recipe is different from running its output in the user's existing machine. Inspecting frozen bytes is different from advancing a guest to record sound. A result needs enough identity and effect information for the client to understand which activity actually occurred. [1, 2, 3, 4, 5]

I chose VCC Plus for this case study because it combines these activities in one application. It contains native BASIC, C, and assembly workspaces, a compiler/toolchain integration, debugging services, indexed reference material, and a simulated Color Computer. Its normal user-facing computer has persistent settings and mounted media. An opted-in disposable playtest uses a separate worker process and private writable copies instead. The MCP integration must preserve the user's existing work while exposing enough of the development cycle to support a meaningful investigation. [1, 2, 6, 7, 5, 8]

I focus on the implementation of the MCP integration. I describe the earlier engine extraction in the companion architecture paper. The contribution is a composition of concrete contracts: explicit scope, revalidated asynchronous authority, content-identified artifacts, observations with stated freshness requirements, bounded execution, and commit-aware recovery. Their value lies in the ways they constrain an actual mutable application and in the negative cases that distinguish a correct implementation from an attractive but misleading tool

response. I also identify the limits of those contracts. An application permission is not an operating-system sandbox, and a tool's retained evidence does not make a model's interpretation correct.

1.1. Research questions and contributions

I organize the case study around four questions.

1. How can a local MCP integration preserve the meaning of a user's authorization as documents, project inputs, and connections change?
2. What evidence should connect a source-level answer to the executable and machine state it describes?
3. How should the application report retries and failures when an operation may already have changed state?
4. Which aspects of this implementation generalize beyond the particular emulator and language workspaces?

My first contribution is an authority model spanning discovery and application execution. A pinned connection describes the tools a client may discover; the live application independently determines whether the present request remains authorized. Native review, permission to build, permission to control a shared run, and permission to start an owned playtest are represented as different contracts. [9, 1, 2, 6, 7]

My second contribution is an identity chain from reviewed inputs through build artifacts and owned runs to observations. Source locations, logical addresses, and snapshot labels are insufficient by themselves. The implementation checks the executable bytes, mapping, processor state, and operation-specific observation identity before presenting interpretations. It can expose a bounded raw observation while withholding a caller or argument interpretation for which the evidence is insufficient. [10, 3, 4, 11]

My third contribution is an outcome model for bounded mutations. Request receipts avoid repeating acknowledged operations. Durable retention protects the previous medium before replacement. Separate handling of final precommit refusal, acknowledged postcommit loss of authority, and an unknown worker outcome prevents these states from collapsing into one generic failure. Cleanup remains possible after ordinary execution allowances are exhausted. [5, 12, 8]

I evaluate these contracts with real guest runs and deliberate fault fixtures. For each result, I identify the mechanism exercised and the state change or refusal observed. [13, 14]

1.2. A short glossary for readers outside emulation

A *project revision* identifies the approved development state, including relevant resources and recipe inputs. A *build artifact* is a particular executable result and its associated evidence. An *owned run* is a separate test execution belonging to the authorized project/artifact relationship. A *snapshot* is a named captured observation; its admissible later uses depend on the service contract. A *receipt* records an operation's accepted or completed outcome and the identities needed to recognize an identical retry.

The target computer has a small logical address space and can map different physical memory pages into it. The same logical address can therefore refer to different code or data at different times. This is an unusually clear instance of a more general problem: a convenient name is useful only together with the identity and context that give it meaning. Similar ambiguity occurs with editor tabs, database rows, selected devices, and files replaced at the same path. [4, 11]

An emulated *field* is a target video-timing unit. I use field allowances to bound the guest work a playtest request can advance. [5, 8, 13]

2. Study design, scope, and threat model

2.1. An implementation case study

I examine the VCC Plus implementation and selected retained qualification evidence reviewed on November 12, 2026. Primary sources are the adapter, application services, worker, and authored regressions identified in the references. The source establishes which checks the application performs. I use the execution records to report observed results. [13, 14, 15]

The recorded experiments combine real guest programs with deliberate fault fixtures that make selected transport and commitment outcomes repeatable.

I use a failure-oriented method. For each proposed contract, I identify a request that would be superficially plausible if one check were missing. Examples include a linked project input, an unapproved support root, a modified artifact at the same source location, a changed memory mapping at an unchanged clock, and a reply lost after a media exchange. The contract is useful when the implementation discriminates that case and preserves the relevant user or guest state. I use the authored fixtures to explain how each failure can be reproduced and the run records to report what happened. [16, 17, 18, 12]

2.2. Actors and protected state

The application is the resource owner. It owns native documents, permissions, the shared machine, retained build results, and any disposable workers. The client-facing adapter translates MCP messages into a smaller private application protocol. An MCP client launches or connects to that adapter and presents tool results to a model. The model proposes actions and interpretations; neither proposal nor apparent confidence supplies authority.

The important protected objects are approved document/project scope; unsaved source edits; executable and debug-evidence identity; the user's existing computer and media; private writable playtest copies; and the distinction between a captured result and a current observation. Availability and graceful cleanup also matter: an exhausted ordinary execution budget should not prevent retaining a disk on stop.

The examined deployment is local. The grant is a bearer credential stored with owner-only access. Unix peer checks establish the same user, and loopback TCP uses the token. The design addresses accidental overreach, stale requests, malformed arguments, asynchronous races, and failures along these application paths. It does not claim to resist a compromised process already holding the same user's filesystem and execution privileges. A stolen valid grant remains an authority problem; an MCP annotation does not neutralize it. [9, 19, 1]

The boundary also does not make all content safe. An approved custom Make recipe can execute native host commands with the application's privileges. A separate emulator worker isolates application state and retained emulator globals, but the examined launch path does not establish an operating-system filesystem or network sandbox. These are substantive limits of the mechanism, not properties inferred from the word *disposable*. [3, 5]

2.3. What the paper means by trustworthy

I define trustworthiness here through a set of inspectable properties: a request remains within authorized scope; its result identifies the state it describes; interpretation is withheld when its supporting evidence is missing; a repeated request is not silently treated as a fresh intentional mutation; and a failure reports the known outcome without inventing certainty. These properties are useful even when the model is mistaken.

They do not guarantee a correct answer to every programming question. A model may misread a genuine diagnostic, propose the wrong line, or confuse two similar screens. The application can make the proposal reviewable and keep it tied to the source revision without proving the proposal's intent. The appropriate relationship is that the model reasons over bounded evidence while the application enforces the resource and effect contracts. [1, 2, 11]

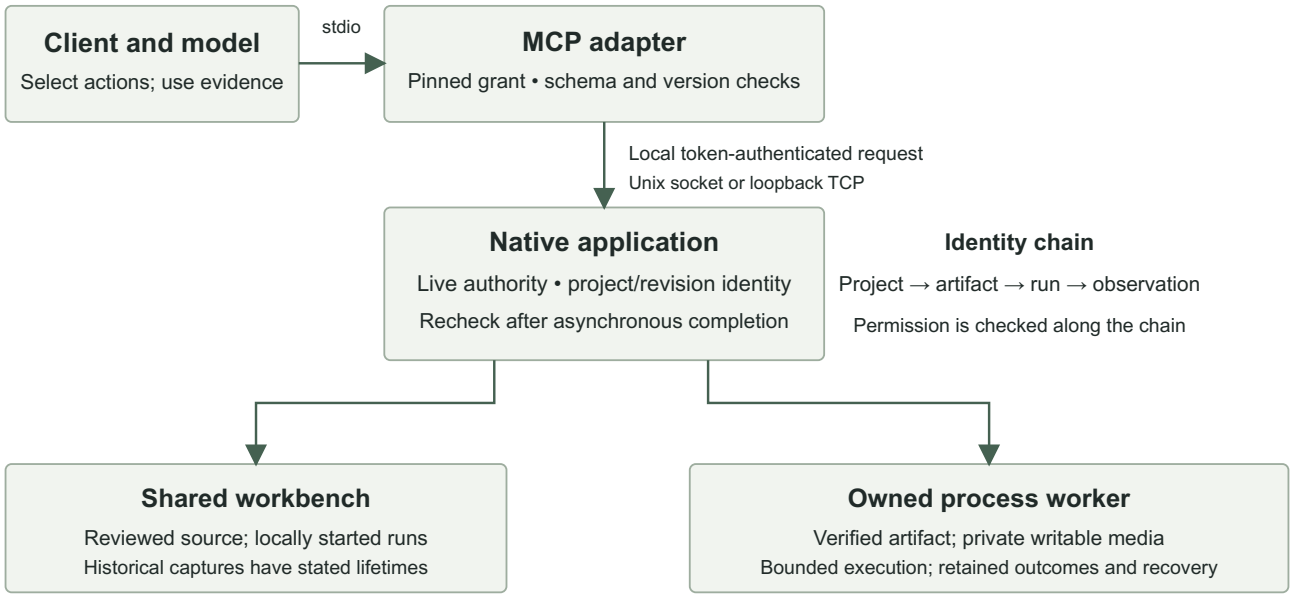
2.4. Scope of protocol claims

I describe the tool-oriented MCP subset implemented by this adapter and the versions it negotiates. It does not present a comprehensive survey of the MCP specification or claim implementation of every optional protocol feature. References to initialization, tool discovery, structured results, and argument errors are anchored in the inspected adapter and its fixtures. Protocol compatibility fixtures establish different properties from an actual client-driven guest demonstration. [9, 20]

The bibliography emphasizes primary implementation and test sources. The design draws on familiar ideas of explicit capabilities, optimistic validation, immutable observations, and operation receipts. The case study's originality lies in their implemented composition and failure handling within this workbench. A broader related-work comparison and a controlled study across independently developed MCP applications would extend the account; the present evidence does not establish first use of these ideas.

3. Architecture and authority

Figure 1 summarizes the authority path. The adapter retains the granted discovery contract, while the native application checks present permission and resource identity before and after asynchronous work. The shared workbench and owned worker have different execution and observation contracts.



Authority and evidence across the MCP integration

Figure 1. Client-facing protocol access and application-owned authority. Process-separated execution preserves a distinct machine state; the application retains the project, artifact, run and observation relationship.

3.1. Actors and the unit of delegation

I separate four actors in the implementation: the native application that owns documents and machine state; a small adapter that speaks MCP over standard input and output; a client that launches that adapter and calls its tools; and a model that may help the client select actions. These actors have different authority. The model's proposed action does not itself confer application access. The adapter translates a permitted request, while the application decides whether its current connection, object identities and operating state still authorize the operation. This division makes the local application the continuing authority rather than treating a successful MCP handshake as permanent permission to manipulate its contents. [9, 1, 2, 6]

A native connection delegates a particular workspace or a deliberately narrower service. A BASIC connection binds one document, including its unsaved draft. C and assembly connections bind a project and a defined input scope. Archive access has its own listener and token and no reference to an editor or emulator; its two tools search the local index and resolve public citations. An Archive connection therefore does not acquire source-reading or execution tools merely because the same application implements them elsewhere. Its host test calls the authenticated socket directly, bypassing the adapter's advertised tool list, and checks that a request for broader source access is refused. This is an important distinction between a convenient discovery surface and an enforced service boundary. [1, 2, 6, 7, 21]

Each connection uses a fresh bearer token supplied through a user-selected grant file. Creation uses exclusive, no-follow file opening and owner-only permissions. The application records the file's device and inode so later cleanup does not remove a replacement placed at the same pathname. The adapter verifies a regular, owner-owned grant file with no group or other permissions, reads bounded bytes, and retains the parsed capability for its own lifetime. Changing the file after startup does not silently revise the running adapter's grant. These controls protect the intended handoff and cleanup behavior; they do not establish that a model or client is trustworthy. [1, 9]

The local host transport offers a Unix-domain socket in an owner-only temporary directory and an optional IPv4 endpoint bound to loopback. Both routes require the application token. Unix connections additionally verify that their peer has the same user identity; the adapter checks the server identity in the same manner. Loopback TCP does not provide this Unix credential check. The design consequently assumes a trusted local account and deliberate handling of bearer files. It is not a boundary against a compromised process with the same account's privileges, nor an Internet-facing authorization service. A local adapter also says nothing about where inference occurs: returned source, queries and observations can reach the client's model provider. The native connection descriptions disclose that distinction. [19, 1, 7, 9]

3.2. Enforcement at two boundaries

The adapter constructs its tool catalog from the pinned grant and validates arguments against the selected schemas before contacting the application. The application then performs its own checks. The second boundary matters because a caller can bypass the adapter and contact the local host envelope directly. It also matters because a schema-valid identifier can refer to a retired document, an obsolete build or a run whose ownership changed after discovery. Tool annotations such as read-only and idempotency hints describe expected behavior for clients; they are not substitutes for either host authorization or operation-specific validation. [9, 1, 2, 6, 21]

I also distinguish permissions by the effect of each operation. Reading a BASIC draft is different from proposing a source replacement; accepting a replacement is different from starting it on the shared machine. Proposed edits enter a native review containing the relevant before and after text and revision. Applying a proposal updates the editor; execution of the shared program is a separate local action. Eligible builds may be requested through an authorized project connection. An explicitly enabled disposable-playtest connection can start a verified artifact in a separate guest. It would therefore be inaccurate to describe the system as requiring a new human confirmation for every tool invocation. The design places review at selected transitions while allowing bounded work within the resulting grant. [1, 2, 6]

Fresh C grants record named flags for playtests, audio, processor selection, joystick input, call context and media exchange. In the inspected implementation these flags derive from one disposable-playtest approval, rather than independent consent dialogs for every extension. Their separate representation still permits version-sensitive discovery and host checks. The resulting contract is explicit playtest opt-in with pinned extension capabilities, not a claim of independently negotiated permission for every individual tool. [2, 9]

3.3. Revalidation across asynchronous work

Native state and access services use main-actor isolation, but actor isolation does not make an asynchronous operation atomic. An awaited capture, compiler operation, SQLite query or worker reply allows other application events to run. The user can close a document, edit its source, revoke a connection or replace a project while the original request remains suspended. The relevant property is therefore not merely that state updates are serialized. Authority and identity must be checked again after the suspension point, before accepting the result or making the next state change. [1, 2, 6, 7]

The BASIC host rechecks the session, selected model and document identity after awaited observations and manual reads. Assembly captures a project snapshot off the main actor, then verifies the same grant, document and workspace revision; it permits one retry for a capture that overlapped a workspace change. The C host refreshes its content view, checks the current grant, applies operation-specific guards, and checks the grant again before releasing a result. Archive access repeats its token comparison after an asynchronous index query. These are related applications of one rule: completion of work started under an earlier authority is insufficient evidence that the completion may still be published. [1, 2, 6, 7]

An abstract form of the rule is:

```
owner = capture_current_connection_and_object_identity()
require authorized(owner, request)
result = await bounded_operation(request, owner)
require current_connection_and_object_identity() == owner
return result
```

This sketch omits the additional checks required for revision-sensitive and mutating operations. A compiler result needs the captured input revision to remain current; an execution request needs a matching artifact and run owner. A final authorization check also cannot undo effects that have already committed. Revocation clears

live authority immediately and initiates asynchronous cleanup, but it is not a transactional rollback facility. The application suppresses stale or revoked results and preserves the separate outcome and recovery rules of the affected operation. [2, 6]

4. Protocol contract and versioning

4.1. Independent compatibility axes

I treat the adapter's public protocol, local bearer file, private host envelope, and application tool generations as independently evolving contracts. Conflating them would make it difficult to determine whether a changed client needs a new parser, a newly issued grant, or simply a different operation. Table 1 summarizes the inspected boundaries. These values identify this implementation; they are not universal MCP requirements. [9, 1, 2, 6, 7]

Table 1. Four independent compatibility axes in the inspected implementation.

Boundary	Implemented representation	Compatibility purpose
Client-facing MCP	Negotiated revisions 2025-03-26, 2025-06-18 and 2025-11-25	Select lifecycle, framing and result behavior supported by the adapter.
Local grant file	Version 1 for BASIC, C and assembly; version 2 for Archive	Distinguish connection-file interpretation, including rejection of Archive grants by older helpers.
Private adapter-to-application envelope	Version 1	Carry the local token, tool name and arguments independently of the negotiated MCP revision.
Application toolset	Grant-pinned generation; fresh C grants use toolset 11	Preserve earlier discovery and schemas while adding explicitly enabled application services.

The adapter reads a grant once and constructs the corresponding catalog. A newly installed adapter can therefore preserve the more limited contract of an earlier connection. For example, the media tools require both the relevant toolset generation and enabled playtest/media flags; call-context discovery has its own generation and flag checks. A client should inspect live discovery and capabilities rather than infer that installation of a newer application expands an existing grant. On the host side, permissions are checked again against the application's current connection state. [9, 2]

4.2. Bounded parsing and explicit result semantics

The adapter implements a small lifecycle state machine: a fresh connection accepts initialization, enters an initializing state, and becomes active after the initialized notification. Tool discovery and invocation require the active state. It also supports ping, ignores notifications that do not require a response, and does not act as a client-method caller. Its advertised capability is the implemented tool service, not every facility that might be available in a larger MCP deployment. [9, 20]

Parsing rejects invalid UTF-8 and bounds client frames, outgoing host requests and incoming host replies. Schema validation checks types recursively, distinguishes Boolean values from integers, enforces declared array and string bounds, rejects unexpected properties and requires an exact match for alternative action schemas. Nesting and item budgets limit validation work. These checks provide early, structured refusals, but a valid schema is still only the first stage: project membership, permission, revision and guest-state checks remain host responsibilities. Encoded-message bounds and operation-specific byte limits can also be stricter than the abstract shape accepted by a schema. [9, 2, 6]

Result behavior is revision-sensitive. The June and November revisions receive structured content as well as a text representation; the March revision uses the earlier representation and supports nonempty request batches. For the November revision, invalid arguments to a known tool produce an actionable tool-error result, whereas older negotiated revisions use invalid-parameter RPC errors. Unknown methods or malformed envelopes remain protocol errors. Image and audio outputs are emitted as their respective content types, with associated metadata, rather than forcing clients to infer media from a textual description. These distinctions keep a refusal, a protocol fault and a successful observation machine-readable. [9, 20]

Resource bounds apply at different layers. The local transport limits concurrent socket workers and keeps blocking socket operations off the application's actor executor. The adapter uses short normal exchange deadlines and a longer deadline for selected playtest operations. These are bounded communication mechanisms, not performance results or a guarantee that every application task terminates within one common deadline. Actual cancellation, operation budgets and post-completion authority checks remain necessary. The helper's synthetic tests exercise lifecycle, revisions, schema branches, forwarding and malformed envelopes; they do not establish emulator correctness or compatibility with every external client. [19, 9, 20]

5. Evolving project scope

5.1. Content identity and authority identity

A selected project is not a stable set of bytes. Headers change outside the editor, generated resources appear, a Make recipe is revised, and a user may retain an unsaved draft that differs from disk. The implementation separates content changes within an approved scope from changes to what the connection is allowed to consume or execute. Both can invalidate an artifact, but they require different responses. Treating every changed byte as a new access decision would obstruct ordinary development; treating a newly discovered helper as another harmless byte could silently expand authority. [10, 16, 17, 2]

The input manifest classifies entries by path, role, inclusion, content hash, executable status and whether they affect revision or build identity. Its content-based identity excludes incidental filesystem timestamps. Changes to membership, role, executable status or control inputs require review, whereas an edit to an already included ordinary source need not enlarge the grant. Generated output and excluded evidence can be described without being promoted into inputs. The distinction is tested with both newly introduced scope and changed recipe/control files, and with timestamp or evidence growth that must not change the relevant identity. [10, 16]

Scope restriction is substantive. Ordinary C connections use the selected C/header inputs or a fixed supported example recipe; they do not acquire the project's arbitrary Make resources. Assembly connections admit a bounded direct-assembly project and reject unsupported linked-project or Make configurations. Explicitly selected support directories for custom projects are subject to capture and review rules. In particular, a proposed support root must not be read or hashed before native approval: even an inspection operation could disclose material outside the earlier scope. [2, 6, 10, 16, 22]

5.2. Refresh without replacing the user's work

In the project refresh design, I keep editor drafts distinct from disk observations. An externally changed clean tab can accept new content, while an unsaved draft remains the user's separate working state. Conflicting external changes are refused rather than silently saving or replacing that draft. Refresh records expose the observed change and its disposition, and execution remains blocked when the newly observed input state has not been accepted. Binary resources participate in revision and artifact identity even when no editor event occurs. This closes a gap in systems that equate source identity with the currently visible text buffer. [2, 10, 17]

The corresponding tests retain an unsaved source draft while refreshing a clean header and discovering another file; refuse conflicting or stale ownership without replacing drafts or disk bytes; and change a binary resource to ensure that a previously ready artifact becomes unavailable. Input-manifest pagination additionally respects the encoded response budget and advances by entries actually emitted. These discriminators test preservation and scope boundaries, rather than merely asking whether a refresh operation returned success. Their execution status and platform coverage are evaluated separately from the code-level mechanisms described here. [17]

5.3. Approved execution is still host execution

Custom Make support allows a connection to build a real project's declared actions and configurations. The adapter does not accept an arbitrary command line or an arbitrary Make target from each tool call. The host binds the request to the approved recipe, snapshots its inputs, performs the build, then checks cancellation, connection and input revision before accepting the artifact. Tests distinguish recipe identity from ordinary source edits, preserve explicitly declared support inputs, and reject a build whose copied inputs change unexpectedly. [2, 3, 22]

Those controls constrain the application's contract; they do not make an approved Make recipe safe against a hostile author. Recipe execution occurs with host-application privileges. This implementation provides no claim of an operating-system sandbox for arbitrary build code. Native review identifies a meaningful authority

transition, and copied inputs improve reproducibility and preservation, but neither mechanism replaces judgment about whether a project should be executed. The generalizable result is an explicit, inspectable boundary between ordinary development edits and a change in executable build authority. [2, 10, 3, 22]

6. Observation and evidence-supported interpretation

6.1. Separate an observation from a live read

A development tool can return bytes without establishing which program produced them, whether they were captured together, or whether reading them changed the target. These distinctions matter when an assistant combines several tool responses into an explanation. A register response taken before a step and a memory response taken after it may each be accurate while their combination describes no actual machine state. VCC Plus therefore represents debugger inspection as an identified observation with retained storage. The response identifies the run and snapshot, reports the relevant processor and source stop, and advertises the memory ranges that the connection may subsequently request. A memory request selects bytes from that observation rather than performing another guest bus transaction. [4]

I give each observation an explicit freshness requirement. An immutable observation can remain useful after the target has progressed, provided it is described as a capture of the earlier state. The shared-run inspection object retains its bytes and uses a program-identity predicate covering machine generation, debugger program revision, memory-mapping fingerprint, processor selection and device blockers. It does not claim that retained values continuously track the current program. A regression explicitly preserves a captured inspection after a demonstration program continues and its disk is ejected, while checking that the program identity remains compatible. A later program attachment changes that identity. Treating every continuation as deletion of every historical observation would discard useful evidence; treating the retained capture as a fresh live read would misrepresent it. [4, 18]

The public memory operation accepts the snapshot identity, an address and a count. The count is bounded to 1–256 bytes, and the entire request must lie inside one advertised retained range. Range preparation removes unavailable locations and excludes addresses at or above `$FF00`, where the platform's device interface is concentrated. The bounded response is therefore a selection from already captured permitted data, rather than an indirect arbitrary-memory command. A caller cannot enlarge the disclosed region by increasing its count or crossing between individually advertised blocks. The implementation also rejects Boolean and nonintegral representations where an integer is required. These checks make the observed bytes' origin and boundary explicit. They do not establish that a selected variable or address is meaningful without the additional executable and type evidence considered below. [4, 18]

6.2. Use stronger identity for interpretations of the current pause

Some tools require more than a historical capture. Owned call context is intended to explain a particular verified pause in an assistant-owned disposable run. Its host path first captures that worker without advancing it and checks the owning connection's run, project revision and artifact relationship. It then requires a paused, attached execution state. To reuse the previously prepared call context, the supplied snapshot must identify the same run and the same captured-state fingerprint. The fingerprint includes boot, field clock, machine frame, cycle count, machine generation, processor and execution mode, MD state where applicable, registers, program revision, mapping identity, physical pages, complete logical-memory data and its availability mask. Equality of the displayed program counter or field clock alone is insufficient. [2, 18, 23]

I require separate checks because a mapping may change while logical bytes happen to remain equal; a register may change without the field counter advancing; or availability may change while the underlying byte array retains old contents. Each change can invalidate the justification for interpreting the current frame. The corresponding regression changes these fields independently and requires a different fingerprint. It also changes budget bookkeeping and requires the fingerprint to remain equal: request-accounting metadata is not guest state. This is a domain-specific choice of what makes two observations equivalent, rather than an indiscriminate hash of every response field. [18]

The service contracts consequently have different temporal meanings:

Observation	Temporal claim	Permitted follow-up
Shared debugger inspection	Identified historical capture under compatible program identity	Read retained bytes inside advertised ranges
Owned call context	Interpretation of the same verified, currently paused owned state	Retrieve the bounded context only while its snapshot and authority still match
Paused graphics capture	Register-based preview from a single paused state	Inspect displayed or explicitly declared graphics buffers without advancing execution
Audio capture	Samples generated during an explicitly executed guest interval	Retrieve the identified clip and its actual interval, including retry-safe replay

A general MCP interface should make these temporal claims visible. The protocol-level fact that an operation returns structured data does not determine whether that data describes a historical record, a current state, or an experiment that advances the target. VCC Plus supplies those semantics in the application contract and validates them in the host and worker. [2, 4, 5, 8]

6.3. Interpret only what executable and compiler evidence support

The source-debugging path does not infer that a function name in a request proves the currently executing bytes belong to that function. Build evidence binds source, linked executable, object and listing information; owned inspection additionally checks the selected runtime mapping and executable guards. For bank-switched programs, the relevant physical-page assignment participates in that verification. Two banks containing identical logical addresses are not interchangeable merely because a line number or function name matches. A mismatch must produce an unavailable interpretation or rejection, rather than a fallback to an assumed flat address space. [2, 11, 24]

The bounded call-context operation illustrates how richer interpretation can be added without turning a narrow debugger capability into arbitrary introspection. The caller supplies only a run identity and a snapshot identity. The newly disclosed raw storage is exactly 64 bytes beginning at the target's actual stack pointer, S. The window must not wrap or cross the protected upper region at `$FE00`; every byte must be available RAM with a physical-page mapping. The internal trusted capture contains enough state to validate these properties, but its existence does not grant the MCP caller unrestricted access to that complete memory image. [11, 23]

The raw window is reported as bytes, big-endian words and physical spans. Those representations are observations, not an inferred stack trace. Caller interpretation requires further evidence: a supported normal CMOC frame at a verified function entry or mapped body, a return location already inside the bounded window, and a unique verified direct call in shared source leading to that callee. The implementation does not search unrelated words for plausible return addresses. Indirect calls, tail calls, interrupt frames and unknown layouts therefore cannot be converted into an apparently authoritative caller merely by matching an address-sized value. [11, 24]

Argument interpretation is similarly restricted. Compiler metadata must identify the same retained build as the executable map. Only supported integer parameters with unambiguous compiler locations, wholly contained inside the S window, are candidates. Duplicate names, overlapping locations, unsupported aggregate storage and out-of-window values remain uninterpreted. Signedness and the compiler's byte offset determine how an accepted value is decoded. Pointer values are not dereferenced, and frame locals are not guessed from a convenient register or presumed prologue. The result distinguishes raw availability from caller and argument availability, so a useful byte window can coexist with an explicit explanation that typed interpretation is unsupported. [11, 24]

This approach gives the assistant less convenient data in some cases, especially optimized code, but preserves a crucial distinction between a captured fact and a plausible reconstruction. The authored negative cases include changed executable bytes, protected or unavailable windows, mismatched banks, indirect and interrupt frames, overlapping parameter locations and unshared callers. They support a narrow interpretation contract; they do not establish general unwinding or complete coverage of every compiler optimization and calling convention. [24, 23]

7. Bounded execution, commit-aware outcomes and recovery

7.1. Separate execution state without overstating containment

The same live emulator used by native editors is an inappropriate place to run every exploratory assistant experiment. VCC Plus can create a disposable guest in a separate worker process, supplied with the selected artifact, ROM bytes, reviewed disk products and verified stop information over private standard-input/output pipes. The worker owns its emulator globals and writable media copies; it does not reuse the user's live machine state. The parent retains the project, revision, artifact and run relationship and rechecks it after asynchronous preparation before accepting a startup or later result. A captured project-input identity includes reviewed file content and validated configuration, rather than relying solely on filenames or modification times. [10, 2, 5, 8]

This is process and application-state isolation. The worker launch shown here is not evidence of an operating-system sandbox restricting arbitrary filesystem or network access. Likewise, a custom Make recipe is executable build logic reviewed through the project's separate authority rules; moving a build into a copied workspace does not make that recipe harmless. The narrower claim is that the supported worker protocol does not expose host paths, credentials or the personal emulator session as guest operations, and that independent process lifetime prevents retained emulator globals from sharing the live machine. Host compromise, malicious native dependencies and unrestricted recipe execution require additional containment measures beyond these contracts. [2, 5, 8]

Finite work and finite retained evidence are part of this execution model. The worker applies an execution allowance of 216,000 fields and a maximum of 16 boots. The host caps accepted mutation requests at 1,024. An input plan contains at most 64 steps, each with bounded fields and at most eight distinct keys, and the plan's requested duration is limited to 3,600 fields. These limits constrain the scope of one owned experiment. They are neither wall-clock performance guarantees nor proof that the emulated program will terminate successfully. A blocked worker also has a deadline and can be terminated through the dedicated pipe-I/O path without synchronously waiting on the application's main thread. [5, 8]

7.2. Declare when evidence capture advances the experiment

Graphics and audio capture deliberately have different execution semantics. Paused graphics inspection derives a display preview and any permitted declared buffer from one paused machine state. It does not run the guest to make a picture look complete. A separately retained last-published frame has its own explicit meaning; it need not equal a new register-based preview at an intermediate stop. Restricting alternate buffers to declared selections avoids turning graphics inspection into a second unrestricted memory interface. [5, 8]

Audio capture, by contrast, must execute the guest to obtain the requested interval. It releases held input, discards audio from earlier operations, advances a bounded interval, collects normal emulator output and pauses again. The receipt states that execution advanced and reports requested and actual fields, cycle endpoints, boot identity, encoding, sample counts, byte count and payload hash. An armed stop can shorten the interval. The tool does not pad the result into an apparent full interval or substitute a host microphone recording. The capture contract concerns generated guest samples before host playback, not a claim that a listener heard particular hardware output. [5, 8]

A clip is limited to 120 requested fields and 131,072 stereo sample frames. Across a run, the host retains at most 16 unique audio requests and 4 MiB of WAV payloads. An identical request can return the same captured bytes without another execution interval, whereas a changed plan under the same request identity conflicts. The regression cases examine guest tones, silence, interval and boot identity, retries and exhaustion, while keeping host playback outside those observations. Such a contract permits reproducible evidence collection without pretending that capture is observationally neutral. [5, 12, 8]

7.3. Treat a request identity as an outcome record

A lost reply is not evidence that an operation did not happen. This is especially important for input, stepping and audio capture, which may advance the guest before a pipe fails. Before dispatching a mutation, the host reserves its request identity with a signature covering the tool and arguments. Reusing the identity with identical arguments can retrieve its retained outcome; different arguments conflict. If an operation's execution becomes uncertain after transport failure, the run is invalidated instead of sending the same input again and risking a second advancement. These semantics apply within a bounded owned run and its retained receipt

ledger. They are not an unbounded, durable exactly-once execution guarantee across arbitrary process or application failures. [5, 12]

A media exchange exposes a particularly useful commit boundary because it can change the mounted disk without advancing the paused guest. Selection identifies an artifact-bound private medium and its expected bytes. The host first obtains the current guest-visible image, retains the outgoing copy and its provenance, and then rechecks ownership. The worker validates and allocates an incoming image before replacement, requires the real floppy-controller busy flag to be clear, and compares the supplied media generation and previous disk hash with its current state. Decorative motor state or the mere presence of dirty bytes is not substituted for controller activity. An invalid replacement cannot eject a valid outgoing disk as an intermediate side effect. [5, 8]

I use the following ordering for an exchange.

```
validate current run and selected medium
capture outgoing guest-visible bytes
retain outgoing bytes and provenance
recheck authority at the final precommit boundary
worker: validate replacement, compare generation/hash, commit, acknowledge
record the committed receipt
publish success only through still-current authority
```

I distinguish three materially different outcomes at this boundary. First, failed retention or loss of ownership at the final precommit check prevents the exchange; the current disk remains usable. Second, the worker may acknowledge a completed exchange before run authority expires on the reply path. The host records that committed outcome before applying the late check and returns `stale_run_after_commit`, distinguishing it from the final precommit `stale_run`. Third, a broken transport can leave the host unable to establish whether the worker committed. That run is retired and reports uncertain final selection rather than synthesizing a success or a known mounted identity. [2, 5, 12]

The distinction does not enlarge permission. Connection revocation takes precedence and can produce `access_denied` without revealing the withheld run outcome. A cached receipt is available to an identical retry only when the connection and relevant project, revision, artifact and run checks still permit access. The client should not create a new request identity simply to repeat an exchange after a lost reply. Retention preserves recovery evidence; it does not revive a revoked grant. [2, 5]

7.4. Keep recovery available after execution is exhausted

The exchange reply is checked against both media and machine state. It must identify the selected bytes, the previous hash and the incremented media generation while preserving field clock, cycles, machine generation, boot and program counter. Media identity can therefore change without being mistaken for guest execution. The host also invalidates earlier owned inspection context when exchanging media: unchanged registers and field clock do not imply that every part of the experiment is unchanged. [2, 5, 8]

When a paused exchange loses its reply, the host can retain the already frozen candidate copies and mark the final selection uncertain. The outgoing dirty image was captured before replacement; the incoming candidate was known before commit; the exchange itself does not execute guest instructions. This supports recovery of those known bytes without claiming a resumed machine snapshot, game-level acceptance of a disk, or crash consistency for every possible later guest write. The tests distinguish emulated controller and saved-byte checks from authored wire fixtures that deliberately lose or falsify replies. A transport fixture demonstrates host failure handling, not compatibility with a real game's storage protocol. [5, 12, 8]

Finally, resource limits must not prevent the cleanup needed to preserve useful work. Stop and retention have a reserved path even when execution or audio-request allowances are exhausted. Once bytes have been exported, a persistence failure retains them for retry rather than repeating the guest operation or silently discarding the image. The disk store binds retained records to project and artifact provenance, validates sizes and hashes, refuses linked or incomplete published records, and preserves parent identity when an explicitly selected retained disk starts another experiment. Recovery and authority are thus complementary: the application preserves known results while continuing to refuse actions whose permission or identity has expired. [5, 12]

8. Evaluation: failure discriminators and recorded evidence

8.1. Evidence classes and interpretation

I use the implementation to identify the guard or state transition, authored regressions to show how to exercise it, and recorded runs to report what happened. For example, I use a running guest to observe media and execution state, and a controlled worker reply to reproduce a change in authority just after commitment. The evidence records identify the corresponding inputs and cases. [13, 14, 15]

8.2. Authority and scope expansion

A discovery-only test can miss an important defect: an unauthorized operation might simply be hidden from `tools/list` while remaining callable on the application transport. The Archive regression deliberately bypasses the helper and addresses the authenticated host. It checks the search-only contract against the native index, citation retrieval and pagination, rejects a broader source-reading operation, rejects a wrong token, and verifies revocation behavior. This supplies a useful discriminator between advertised restriction and application enforcement. [21]

Project scope has a related discriminator. An unapproved support root must remain unread and unhashed while approval is pending. Otherwise a preview of a permission request could itself disclose the resource that the user has not yet shared. The project-input tests distinguish metadata describing a proposed root from inspection of its members. They also reject linked inputs and prevent a restricted connection from acquiring an unshared Make recipe merely because the project contains one. [16]

A revision is broader than editor text. The refresh tests change an existing binary resource while editor state remains unchanged and require the revision to advance and the old artifact to become ineligible. They distinguish this case from harmless output written into an excluded evidence directory. Separate cases preserve unsaved drafts and current disk bytes while refusing an external conflict. These are concrete expressions of the rule that content changes and scope expansion have different review requirements. [16, 17]

8.3. Observation consistency at an unchanged clock

A clock value is a useful discriminator, but it cannot identify a complete observation. The owned call-context fingerprint tests vary registers, memory, availability, mapping, processor/mode, and other state components while holding selected timing values fixed. A relevant guest change must change identity; a request-budget bookkeeping change must not. This establishes the intended equality relation much more precisely than a test that only resumes the guest and expects the old snapshot to fail. [18]

The captured-memory tests expose the complementary contract. After an immutable shared-run capture, changing live guest memory must not change the bytes returned from that retained snapshot. Wrong run or snapshot IDs, unavailable/I/O ranges, cross-range reads, and malformed counts are rejected. The continuation/ejection case intentionally preserves historical captured inspection when its program identity remains valid. A universal policy that invalidated every capture after every activity would contradict this service. [4, 18]

These cases matter to a model-facing interface because the client cannot reliably repair an ambiguous answer after the fact. A value needs a stated capture and identity contract before it can support reasoning about a program. The correct service may return historical bytes, a current verified observation, or an explicit refusal; it should not silently substitute one for another.

8.4. Supported interpretation and explicit unavailability

The bounded call-context regressions exercise the difference between bytes and their interpretation. Cases refuse unsupported optimized, indirect, interrupt, or unknown frames; prevent a scan of unrelated words; reject a bank mismatch even when logical bytes appear identical; and avoid dereferencing pointers. Supported integer arguments use compiler location evidence and the demonstrated frame/call relationship, not an inferred sequence of plausible stack values. [11, 24]

An authenticated access case combines the real grant path with a frozen seven-argument context and stale-authority refusals. It complements the smaller interpretation fixtures: the latter isolate particular ABI and bounds rules, while the former checks the application-to-worker ownership chain. Neither proves a general stack unwinder or a complete C debugger. Their result is the narrower implemented contract in Section 6. [23]

8.5. Failure before commitment

The media-exchange tests exercise a dirty paused disk rather than an abstract success flag. When retaining the previous bytes fails, the current disk remains usable and unchanged. When authority becomes stale during asynchronous durable retention, a final precommit check prevents replacement. The latter case runs on both emulated processors and asserts the specific `stale_run` outcome at that boundary. Earlier checks may have their own more specific selection/ownership failures. [5, 12]

This distinction is important because preservation cannot be established solely by a rejected request. A refusal that leaves the previous medium detached or loses its dirty bytes would satisfy a superficial error assertion while violating the user's expectations. The relevant evidence is the combination of the refusal, unchanged mounted selection, guest state, and retained or still-usable bytes.

8.6. Acknowledged and uncertain outcomes

The later acknowledged-exchange case uses a deterministic negative wire fixture. The fixture commits one media swap and changes an external marker before the success reply is processed. The authority callback then fails. The expected result is `stale_run_after_commit`; the test checks that the media generation is one, relevant machine/timing state is unchanged, the retained receipt can be replayed through the component's permitted test path without another exchange, and both media survive final retention. Production authorization remains a separate outer gate. This fixture establishes a host bookkeeping and ordering property, not additional emulator compatibility. [12, 13]

The lost-reply case addresses a different uncertainty. If the worker may have committed but the host cannot establish the final selection, the run is retired and frozen copies are retained with `media_selection_uncertain`. The implementation does not invent a definitive mounted identity or invite a fresh mutation to resolve the ambiguity. Together, the two cases distinguish an acknowledged effect whose reply is withheld from an effect whose final outcome cannot be established. [5, 12]

The follow-up runs reproduced refusal before commitment and acknowledgment after a swap. The first leaves the original medium in place. The second keeps both media and records the committed outcome once. [13]

8.7. Exhaustion and cleanup

A run can reach its mutation, execution, boot, or capture allowance and still have user-relevant bytes to preserve. Tests therefore check stop and retention after ordinary allowance exhaustion, retry after failed persistence, and remaining cleanup paths after the audio byte budget is consumed. An allowance restricts further work; it should not become a reason to abandon already accepted state. Ownership and conflicting-request checks remain in force during cleanup. [5, 12]

This is a transferable failure discriminator. For a remote laboratory, database tool, editor, or simulator, cleanup is often the operation that protects the most important state after an earlier task has failed. A single global budget checked before all operations can accidentally remove that capability exactly when it is needed.

8.8. Bounded work and evidence

The implementation places bounds on frame input, stack inspection, memory reads, and audio retention. Logs are paged. Different deadlines accommodate ordinary requests and longer playtest operations. I use these controls to keep the amount of work and evidence requested by a client predictable. [9, 19, 4, 11, 5]

9. Discussion: design lessons beyond the workbench

9.1. Represent authority as a relationship

A connection token identifies permission, but the useful authority is a relationship among that connection, its selected resource, the current revision, and the operation. An authorized build does not make every later executable the authorized result. A retained receipt does not reopen a revoked connection. A source name does not select an arbitrary running program. This suggests that application developers should design resource identities and permission checks together, rather than adding authentication around an existing command table. [2, 10, 5]

The relationship also explains why discovery is stable while live use can be refused. A client needs a predictable schema for planning, and the application needs the ability to invalidate authority when its underlying resource

changes. Pinning a grant's advertised contract and revalidating the live request serve these different needs. Silent schema expansion and stale-result acceptance would undermine both.

9.2. Make operation effects part of the interface

A tool name can obscure whether the application advances time, changes state, or only reads captured evidence. I make these distinctions explicit in the implementation. Graphics capture observes one paused state. Audio capture advances a bounded amount of guest execution. A project refresh can change which revision is valid without compiling it. A successful disk exchange changes media generation while preserving specified guest machine state. [17, 5, 8]

For other applications, the corresponding question is concrete: does a query execute user code, trigger a device read, consume a stream, refresh a live value, or return a frozen record? Stating the effect in the schema and result is useful; enforcing it in the application is essential. The evidence offered to the model should preserve that distinction instead of requiring inference from a friendly description.

9.3. Give observations service-specific lifetimes

There is no single universally correct freshness rule. A historical captured memory page can remain useful after the target continues, provided it is identified as that capture and retains the applicable program relationship. A current caller/argument interpretation needs stricter agreement with the stopped executable, mapping, registers, and captured memory. Treating all observations as live values encourages silent substitution; treating all of them as immediately unusable discards legitimate historical evidence. [4, 18, 11]

An API should therefore define the equality relation required by each operation. The owned call-context implementation is instructive because it includes availability bytes and processor/mode as well as ordinary registers and timing state. It also deliberately excludes unrelated allowance bookkeeping. This is a precise domain-specific decision, rather than a reliance on whichever counter was most convenient to expose.

9.4. Separate execution outcome from authority to disclose it

A committed effect can outlive the authority under which the request began. The host must retain enough bookkeeping to avoid repeating the effect, while withholding information when the connection is no longer authorized. The distinct postcommit reason improves clarity when the connection can still receive that scoped result; revocation can still require the more general `access_denied` response. Stored knowledge and permission to disclose it are different properties. [2, 5]

This also limits retry claims. An identical request can reuse a retained outcome within the stated ownership and lifetime constraints. An unknown worker outcome may require retirement and explicit recovery. These mechanisms provide bounded duplicate suppression and intelligible failure behavior. They do not establish an unlimited exactly-once guarantee across arbitrary process loss, credential replacement, or state restoration.

9.5. Reserve resources for preservation

Ordinary execution and capture allowances help bound work. Cleanup needs its own policy because accepted work can already have produced valuable state. The inspected implementation preserves stop/retention eligibility after ordinary budgets are consumed. This pattern generalizes to flushing a recorder, retaining an editor recovery file, releasing a laboratory instrument, or reporting a completed operation whose reply was delayed. [5, 12]

Reserved cleanup does not mean unauthenticated cleanup. The caller still needs the correct relationship to the resource, and conflicting request IDs remain invalid. A good design protects the outcome without turning a recovery path into an authority bypass.

9.6. Prefer narrow evidence to confident interpretation

The 64-byte call-context window is deliberately modest. Its educational value is the decomposition: disclose a bounded supported observation, identify the physical storage that backs it, and attach a semantic interpretation only when the required compiler and instruction evidence verifies. Unsupported frames stay uninterpreted or unavailable. That response is more useful for reliable reasoning than a plausible but undocumented guess about a caller. [11, 24]

The same principle applies to build diagnostics and Archive citations. A compiler exit status, an output-validation result, and source-debugging availability are separate facts. An Archive search result identifies a

source location; it does not establish that the assistant read and understood the passage. A model-facing API can expose these distinctions consistently even when an individual client presents them poorly. [3, 7]

10. Applying the design elsewhere

I assign authority to the application that owns the resource. In this local implementation, the user account protects the bearer file, and the emulator worker has its own process and machine state. A custom Make recipe runs with the native host permissions of the application. These are design boundaries that another implementation must define for its own users, storage, and execution environment. [19, 1, 3, 5]

I version the protocol, grant, private envelope, and application toolset independently. That gives the adapter a stable catalog for the connection while the application continues to enforce current authorization for each operation. The same pattern applies when a long-running task finishes after the user's permissions have changed.

11. Conclusion

A useful MCP integration must preserve more than a callable command surface. In a stateful development workbench, authority can become stale during asynchronous work, a source location can outlive its executable relationship, and a lost reply can conceal a completed mutation. VCC Plus addresses these problems through independently enforced scope, explicit identity chains, service-specific observation rules, bounded execution, and retained operation outcomes.

I identify several practices that transfer beyond emulation: revalidate authority after suspension; distinguish content revision from scope expansion; bind interpretations to captured executable evidence; state whether a tool observes or advances the target; preserve cleanup after ordinary work is exhausted; and report what is known about commitment before advising a retry. Selected recorded executions and discriminating regressions support these contracts within their stated configurations. The result is a concrete model of application responsibility: provide the model with bounded, identified evidence while retaining ownership of permission, state transition, and recovery.

Acknowledgment

I designed VCC Plus with sponsorship from Dezota LLC. I thank the original VCC contributors and the authors of the compiler, library, and filesystem components used in the integration. Those components retain their original authorship and credits.

References

Primary implementation links identify source contracts and authored fixtures. The selected evidence files retain the scope of recorded executions and distinguish them from source inspection.

Reference 1

VCC Plus project (2026). platforms/macos/App/MCPAccess.swift. BASIC access service, native proposals, document/session checks and grant-file lifecycle.

Reference 2

VCC Plus project (2026). platforms/macos/App/MCPCMOC.swift. C access service: grants, refresh, build/run ownership, observation and response authorization.

Reference 3

VCC Plus project (2026). platforms/macos/App/CMOCBuildService.swift. Build lifecycle, copied-input verification, cancellation, diagnostics and dependency-cache policy.

Reference 4

VCC Plus project (2026). platforms/macos/App/MCPDebuggerInspection.swift. Frozen shared-run captures, permitted ranges, program identity and captured memory replies.

Reference 5
VCC Plus project (2026). platforms/macos/App/MCPPlaytest.swift. Owned worker, bounded mutation ledger, capture, commit-aware media exchange and retention.

Reference 6
VCC Plus project (2026). platforms/macos/App/MCPAssembly.swift. Direct-assembly sharing, capture, proposals, builds and post-await identity checks.

Reference 7
VCC Plus project (2026). platforms/macos/App/MCPArchiveAccess.swift. Separate search/citation service and asynchronous grant revalidation.

Reference 8
VCC Plus project (2026). platforms/macos/Tools/Playtest/main.mm. Private worker protocol, field budgets, processors, capture and guarded floppy replacement.

Reference 9
VCC Plus project (2026). platforms/macos/Tools/MCP/main.mm. Local MCP adapter: lifecycle, grant formats, schema checks, discovery, forwarding and media results.

Reference 10
VCC Plus project (2026). platforms/macos/App/CMOCProjectInputs.swift. Canonical inputs, scope restriction, content identity and captured build dependencies.

Reference 11
VCC Plus project (2026). platforms/macos/App/CMOCOwnedCallContext.swift. Fixed S window, physical memory, normal ABI/direct-call proof and conservative parameter interpretation.

Reference 12
VCC Plus project (2026). platforms/macos/Tests/MCPPlaytestTests.swift. Authored worker, actual guest execution, capture, retry, media and persistence regressions.

Reference 13
Holman, B. K. (2026). Selected existing execution records, scope, host modes, case selectors and overlap.

Reference 14
Holman, B. K. (2026). Mechanism-to-failure and test mapping; source-inspected regressions distinguished from recorded executions.

Reference 15
Holman, B. K. (2026). Primary source inventory and evidence classification for this paper.

Reference 16
VCC Plus project (2026). platforms/macos/Tests/CMOCProjectInputTests.swift. Scope expansion, recipe changes, unapproved roots and linked-input regressions.

Reference 17
VCC Plus project (2026). platforms/macos/Tests/MCPProjectRefreshTests.swift. Refresh preservation, binary-resource identity, conflict rejection and bounded manifest pagination.

Reference 18
VCC Plus project (2026). platforms/macos/Tests/MCPDebuggerInspectionTests.swift. Historical capture, unsafe/malformed reads and same-clock owned-observation identity regressions.

Reference 19
VCC Plus project (2026). platforms/macos/App/MCPTransport.swift. Local socket transport, peer checks, listener bounds and asynchronous socket work.

Reference 20
VCC Plus project (2026). platforms/macos/Tools/MCP/test-helper.py. Authored helper protocol, version, grant and forwarding fixtures.

Reference 21
VCC Plus project (2026). platforms/macos/Tests/MCPArchiveAccessTests.swift. Search-only native/host equivalence and broader-access, malformed-argument and revocation regressions.

Reference 22
VCC Plus project (2026). platforms/macos/Tests/CMOCBuildServiceTests.swift. Artifact/input mutation, stale-product, failed-build and cancellation regressions.

Reference 23
VCC Plus project (2026). platforms/macos/Tests/MCPCallContextAccessTests.swift. Authenticated access, actual helper discovery and owned call-context/media integration fixtures.

Reference 24
VCC Plus project (2026). platforms/macos/Tests/CMOCOwnedCallContextTests.swift. Unsupported frames, bank mismatch, parameter offsets, bounds and pointer non-dereference regressions.