PLwC Gateway v0.2.0-rc18.dev9

Program and Software Description of the MCPB

Detailed technical description, architecture, governance, persona management, access restriction, risks, and operating model

Updated on 07/13/2026 - Basis: verified artifact plwc-gateway-0.2.0-rc18.dev9.mcpb and the retained English DOCX edition

Check point	Value
Artifact	plwc-gateway-0.2.0-rc18.dev9.mcpb
Actual SHA256 of the delivered package	2F71AC903BF85CC70023805EC0F901E84C4294982C1B59940350DB3591A2D345
File size	537,994 bytes
Files in the MCPB	66
Uncompressed package size	1,724,140 bytes
Manifest version / package version	manifest_version 0.4 / version 0.2.0-rc18.dev9
Python compatibility	>=3.11
Platforms per manifest	darwin, win32, linux
Public server	exactly one server: plwc-gateway
Public tools	plwc_status, plwc_describe, plwc_profile, plwc_reflection, plwc_governor, plwc_sandbox_run, plwc_workspace_operation, plwc_document_operation
Release status	Open Beta; unsigned; not final; not production-certified; package/Desktop/Odysseus smoke PASS

This document is intended as a technical software description for third parties. It describes not only which tools are visible in the MCPB, but why the architecture is built this way, which security boundaries it technically enforces, where governance decisions are made, and which false assumptions during use would be dangerous.

The previous rc7 template was suitable as a compact technical description. For rc18.dev9, simply extending that template is not enough, because since rc7 additional topics have become product-relevant: configured active-profile precedence, persona-layer deactivation, Qdrant as an optional and reconstructable retrieval index, CLU Doctor as a read-only diagnostic mode, extended governor operations, node sandbox, workspace parameter diagnostics, and significantly more precise first-run/onboarding semantics.

The description is deliberately written in a conservative style. PLwC is not being sold here as a magic AI shield, but as a local, policy-controlled MCP gateway process with clear boundaries. That is technically more honest and more helpful later during reviews. Where the package does not guarantee something, this is stated explicitly in the Non-Goals, Risks, and Open Checkpoints section.

0. Table of Contents

The focus is on governance, persona management, and access restriction. The function reference at the end is deliberately extensive, so that a technical reviewer can compare the visible facade operations against the manifest and the MCPB's source state.

1. Purpose, Classification, and Target Audience

PLwC Gateway is a local MCP gateway for Claude Desktop and compatible local MCP clients. Its purpose is not to pass as many capabilities as possible through to a model unchecked, but to establish a central technical boundary. In the intended mode of operation, the model sees exactly one public server, plwc-gateway. Within this boundary, profile context, workspace access, document operations, sandbox execution, reflection, governor flows, and auditing are all routed through a shared policy layer.

The architectural core is simple but important: an AI model should not have parallel, direct access to raw filesystem, shell, profile, and governance servers. As soon as multiple raw MCP servers are visible at the same time, the model can effectively bypass a governance decision simply by taking a different route. PLwC therefore does not try to morally persuade every other tool; instead it reduces the public surface to a controlled facade.

In practical use, this means: project files can be edited locally, documents can be created and analyzed, small code snippets can run in Docker, profiles can be compiled, and memory/persona changes can be prepared. Critical, durable changes, however, are not treated as a normal file write, but go through plan/apply mechanisms with explicit confirmation and provenance.

The MCPB is an unsigned Open Beta build and is not production-certified. It is suitable for public Open Beta testing, technical reviews, and controlled distribution when the exact package hash is verified. Enterprise operation would additionally require package signing, reproducible builds, binding release processes, documented rollback paths, and an assessment of the local client operating environment.

1.1 Primary User Groups

- Power users who want to use Claude Desktop or another compatible local stdio MCP client with project folders without giving the model unfiltered filesystem or shell access.
- Authors, technical writers, and game developers who repeatedly generate or inspect DOCX, XLSX, PPTX, PDF, ZIP, and image artifacts.
- Developers and reviewers who want long-term profiles, memory rules, working style, audit, and governance to stay traceable.
- Teams investigating a local AI workstation architecture in which security boundaries are explainable, testable, and repeatable.

1.2 Not to Be Confused With

PLwC is not a cloud DMS, not an autonomous agent platform, not a general SIEM solution, not a vector database used as the source of truth, and not a replacement for enterprise-wide endpoint security. PLwC is a local gateway layer for a specific interaction pattern: a chat model calling controlled MCP functions.

Security applies only to calls that actually pass through plwc-gateway. If other local MCP servers are active alongside PLwC that grant direct filesystem or shell access, this architecture can be circumvented. The most important operating principle is therefore: do not activate parallel raw-access MCPs in a PLwC project unless they have been explicitly assessed separately.

2. Artifact Findings and Package Structure

The delivered MCPB artifact contains the manifest, gateway code, documentation, profile templates, Docker worker definitions, Node runner definitions, configuration examples, and release/smoke documents. Compared to the old rc7 template, the package structure is broader and too extensive for a plain short description.

The actual SHA256 of the artifact examined here is stated on the cover page. This value matters for distribution, review, and reproduction. A release document that states a different hash than the delivered package is a serious release-process error, even if the software itself works correctly. Hash consistency is not a formality — it is the basis for everyone involved talking about the same artifact.

Package area	File count	Meaning
(root)	7	Manifest, README, Apache 2.0 license, icon, entry point, and dependency metadata.
config	1	Example security.yaml and security-relevant default configuration.
docker	11	Document Worker and Node Runner Docker contexts.
docs	11	Public architecture, security, tools, installation, onboarding, safe-mode, and troubleshooting documentation.
profiles	8	Profile templates for CORE, PERSONA, TEMPERAMENT, memory, reflection, journal, and governance.
src	28	Gateway implementation, adapters, policy, audit, runtime, onboarding, and MCP server.

2.1 Package Sources Relevant to This Description

For this software description, manifest.json, README.md, docs/ARCHITECTURE.md, docs/SECURITY_MODEL.md, docs/TOOLS.md, docs/FIRST_RUN.md, docs/SAFE_MODE.md, docs/AUDIT_LOG.md, docs/RISK_REGISTER.md, docs/CONFIGURATION.md, docs/INNER_GOVERNOR_PROMOTION_RULES.md, docs/CLU_DOCTOR_PLAN.md, the smoke reports, and the implementation under src/plwc_gateway are especially relevant.

Documentation is not treated as a substitute for code review. Especially with MCP gateways, the distinction between documented intent and the actually registered public surface is decisive. That is why the manifest, tool list, and server registration are considered together in this description.

2.2 rc7-to-rc18.dev9 Delta in Technical Terms

Topic	rc7 template	rc18.dev9 assessment
Public facade	Eight-tool facade already described.	Eight tools remain the public core; Dev9 combines the facade bootstrap with privacy-filtered packaging and English-first reflection marker/trust aliases while preserving canonical PBA2 storage.
Onboarding	Governed profile creation described.	Configured active_profile_name precedence is now explicitly handled; profile creation can be effectively blocked if Claude Desktop still selects another profile.
Persona layer	Profile compile and compiled_layer as the working context.	persona_layer_disabled as a config switch: the PERSONA block can be omitted from compile output; hard governance stays active.
Qdrant	Not central in rc7.	Optional retrieval index, off by default, reconstructable, not the source of truth; boot/full compile do not call Qdrant automatically.
CLU Doctor	Not core to the rc7 template.	Deterministic read-only diagnostic mode with a clear no-mutation boundary.
Sandbox	Docker-only, no host fallback.	Node Runner additionally described; Python/Shell/Node via server-owned Docker policy.
Governance	Plan/Apply present.	Additional governor operations such as list_retirable, retire, reindex/drop_index in the facade context.

3. Architecture Principle and System Boundary

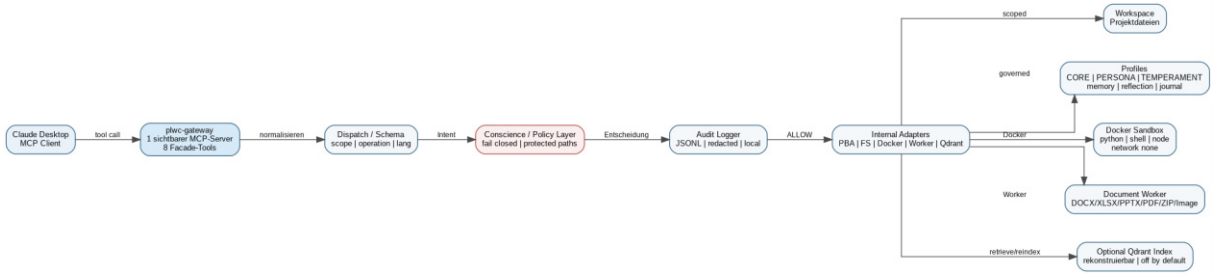

Figure 1: Component model with Claude Desktop, plwc-gateway, policy, audit, adapters, profiles, workspace, and Docker workers.

PLwC is designed as a boundary architecture. The public boundary is not the individual file handler, not the Document Worker, and not the PBA/profile core — it is the gateway. A request from the model is first sent as a tool call to the facade. An intent is derived from the call. That intent is checked against policy rules, path rules, protected-path rules, operation permissions, and, where applicable, governance rules. Only after that is an internal adapter invoked.

The separation between the public boundary and internal adapters is not cosmetic layering. It prevents internal capabilities from accidentally appearing as direct model tools. An internal adapter is allowed to be powerful as long as it is only reachable under gateway policy. A publicly visible raw adapter, by contrast, would be a bypass risk.

The architecture follows a fail-closed model. Unknown operations, unsafe paths, dynamic Docker parameters, protected governance files, absolute host paths, traversal, missing confirmation, or mismatched plan types are not to be interpreted generously. Where the state is not clearly allowed, it is denied. That makes some usage less convenient, but it is the correct default posture for a security gateway.

3.1 Trust Zones

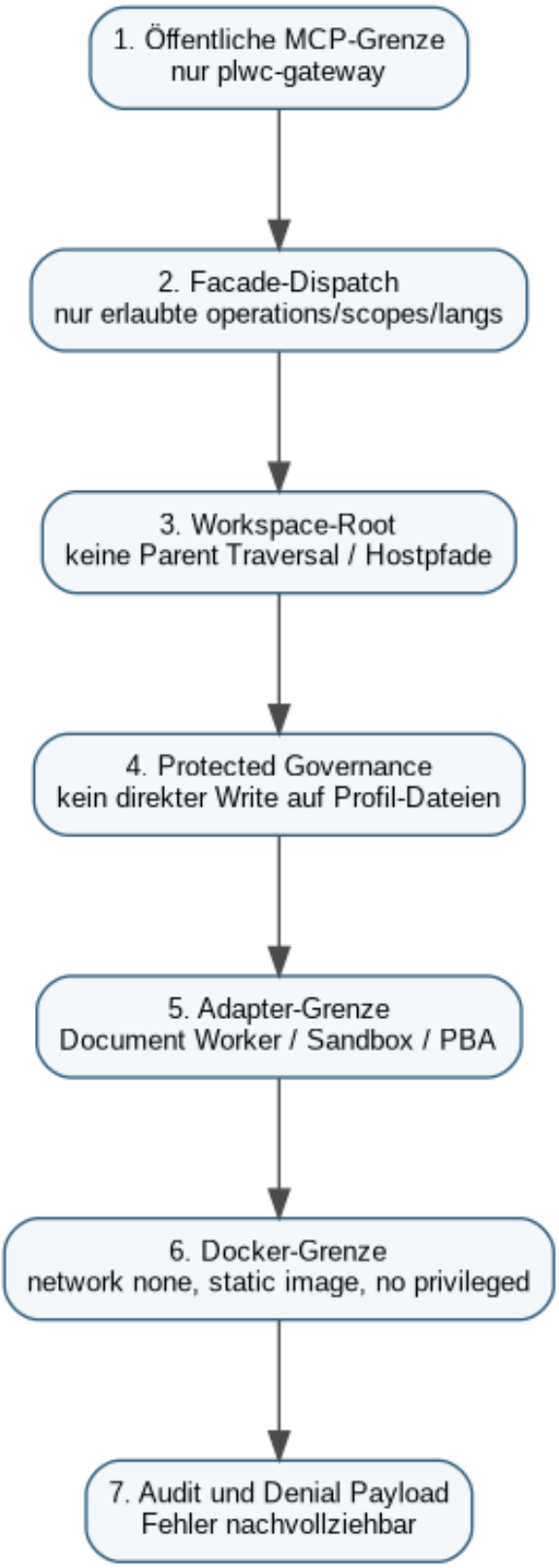

Figure 2: Access layers from the chat call through the facade and policy to workspace, profiles, and containers.

Zone	Examples	Trust assumption	Rule
MCP client / model	Claude Desktop or compatible local stdio	Not trustworthy in terms of input: can make	May not directly determine policies,

	client, tool call	mistakes, produce hallucinations, or attempt risky parameters.	mounts, images, or governance files.
plwc-gateway	FastMCP server, facade dispatch, policy engine	Central enforcement authority.	Must remain the sole public PLwC boundary.
Internal adapters	Filesystem, PBA/profile, sandbox, Document Worker, Qdrant	Usable only through the gateway.	No direct public registration.
Workspace	Project files, generated artifacts, test folders	Payload area, but bounded.	Only allowed roots, no profile/governance root.
Profile/governance	CORE.md, PERSONA.md, memory.md, reflection.md, governance/config.yaml	Protected, persistent control state.	No normal workspace write access; governed flows only.
Docker container	python:3.12-slim, plwc-document-worker, plwc-node-runner	Execution environment for potentially risky work.	No network, no host shell, server-owned flags.

3.2 Allowed and Forbidden Public Boundaries

Only the visible public boundary plwc-gateway is allowed. Forbidden is a configuration in which PBA-MCP, Desktop Commander, a raw shell server, or a separate document server are also publicly visible. Such parallel servers would not automatically fall under PLwC policy and could therefore enable direct path, shell, or governance access.

PLwC's architectural value lies precisely in combining capabilities without exposing them raw. Workspace operations are not safe because file operations are inherently harmless. They are only bounded because they pass through path normalization, protected-path checks, audit, and operation permissions.

Scenario	Assessment	Rationale
Only plwc-gateway visible	Target state	All capabilities run through a shared policy and audit boundary.
plwc-gateway plus Desktop Commander	Not recommended / bypass risk	The model could use direct filesystem or process functions outside PLwC.
plwc-gateway plus a separate shell MCP	Unsafe for PLwC projects	Host execution would not be bounded by PLwC's Docker policy.
plwc-gateway plus a pure read-only knowledge connector	Context-dependent	Can be defensible if no write/shell/host paths are exposed.

4. Deployment, Runtime, and Packaging Model

The MCPB packages a Python-based MCP server. The manifest defines Python as the entry point via server.py and maps Claude Desktop extension fields to environment variables. Claude Desktop can install the MCPB directly; local GPT-compatible clients and Odysseus use the same verified runtime by extracting the package and registering its bundled server.py as one stdio MCP server. The local configuration supplies the workspace, profile root, active profile, security config, and relevant thresholds.

The heavy document libraries do not run in the gateway's main process but in the Document Worker container. This reduces the gateway process's attack surface and keeps parser/office risks in a separate execution environment. Sandbox execution and the Document Worker require Docker; without Docker, PLwC stays in Safe Mode for these capabilities.

Qdrant is optional and, per the manifest, disabled by default. This matters because a retrieval index must not become the canonical memory store. Profiles and memory remain file-based, governance-controlled sources; the index is derivable and reconstructable.

4.1 User Config and Environment Mapping

User config field	Type	Default	Description
workspace_path	directory	—	Optional workspace directory. Leave empty to use a safe user-scoped PLwC workspace.
profiles_path	directory	—	Optional profiles directory. Leave empty to use a safe user-scoped PLwC profiles directory.
active_profile_name	string	default	Name of the active PLwC profile inside the profiles path. Default: default.
security_config	file	—	Optional local security.yaml path. Leave empty to use conservative bundled defaults and user-scoped paths.
memory_write_threshold	number	2	Minimum confirmed evidence count before memory writes are proposed or applied. Default: 2.
persona_write_threshold	number	3	Minimum confirmed evidence count before persona writes are proposed or applied. Default: 3.
temperament_write_threshold	number	2	Minimum confirmed evidence count before temperament writes are proposed or applied. Default: 2.
qdrant_enabled	boolean	False	Enable the optional Qdrant semantic retrieval index (off by default). The config-window value is authoritative.
persona_layer_disabled	boolean	False	Turn on to omit the PERSONA block from profile compile output by default. Use for Odysseus or other explicit context-injection PoCs; hard gates and governance remain active.

4.2 Operating Paths and State Separation

Workspace and profile root must be separate. The workspace is the working area for normal files: drafts, exported files, test folders, JSON specifications, and generated artifacts. The profile root, by contrast, contains the system's control files: profile, memory, persona, reflection, journal, and governance configuration. If the workspace contained the profile root, a normal workspace write could potentially alter the system's control logic.

The MCPB allows empty workspace_path and profiles_path values. In that case, safe, user-scoped defaults are used. This prevents a local MCP client from landing in an absurd or unsafe working base due to an unfavorable starting directory such as System32. Even so, a beta tester should check the actual runtime paths via plwc_status(scope="runtime").

```
Recommended separation: workspace_path = C:\Users\\<User\>\PLwC\workspace profiles_path =
C:\Users\\<User\>\PLwC\profiles
```

Not recommended: workspace_path = C:\Users\\<User\>\PLwC profiles_path = C:\Users\\<User\>\PLwC\profiles # profiles then live under the workspace

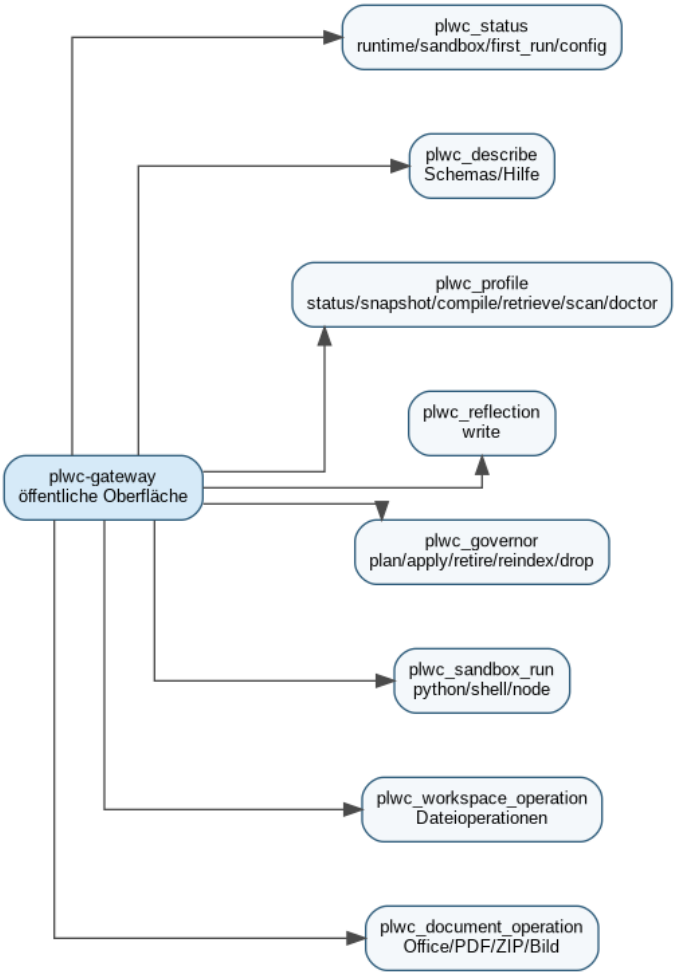

Figure 3: The eight public facade tools and their functional areas of responsibility.

The v0.2 line replaces the earlier broad surface with many individual tool names with eight facade tools. The actual operations are selected via parameters such as scope, operation, lang, plan_type, and compile_mode. This reduction makes the visible API smaller, easier to review, and more stable.

Important: the facade is not a reduction in functionality. Many of the earlier individual tools continue to exist conceptually but are no longer visible as their own public tool names. plwc_compile_profile, for example, becomes plwc_profile(operation="compile"), plwc_governor_plan becomes plwc_governor(operation="plan"), and plwc_write_workspace_file becomes plwc_workspace_operation(operation="write").

The facade forces callers to use more explicit parameters. That is good for deny decisions: a tool call can be classified by an operation and a target category. A raw toolbox with many specialized names, by contrast, tempts developers to scatter governance rules across tool names instead of a central policy.

Tool	Main purpose	Mutation capability	Security relevance
plwc_status	Facade for runtime, sandbox, first-run, and client-configuration status via scope=runtime|sandbox|first_run|config.	read-only	Diagnostics must not output secrets.
plwc_describe	Return read-only PLwC facade schemas,	read-only	Schema must

	allowed dispatch scopes and operations, required fields, and common denial reasons.		reflect the current runtime.
plwc_profile	Facade for governed profile status, snapshot, retrieval, diary/Tagebuch scan and compile operations; compile defaults to compile_mode=boot, supports working with task_context and optional fresh Qdrant semantic memory, and full for audit/diagnostics.	read-only up to compile/retrieve/scan; status-dependent	Compiled layer must not be fabricated.
plwc_reflection	Facade for governed reflection writes via operation=write with marker, target, candidate_for, evidence and entry_date validation.	governed write	Semantic and evidence gates prevent persisting garbage.
plwc_governor	Facade for governed profile planning, confirmed apply, retire/list_retirable diagnostics, and Qdrant reindex/drop_index operations.	plan/apply; mutation only on confirmed apply	Central mutation layer for profile and memory.
plwc_sandbox_run	Facade for Docker-only sandbox execution via lang=python|shell|node; no host-shell fallback. lang=node runs a workspace-relative .js script in a separate plwc-node-runner:0.1.0 image.	Execute inside container	No host fallback, no Docker parameters from the model.
plwc_workspace_operation	Run bounded governed workspace operations: list, search, read, write, file_info, batch_read, create_dir, move, rename, exact_replace, copy (byte-exact file copy), read_binary (returns base64) and write_binary (accepts base64). write with source_path is rejected; use copy.	read/write within the workspace	Path normalization and protected-path denial.
plwc_document_operation	Create DOCX/XLSX/PPTX/PDF, run DOCX and XLSX Creation V2, read workspace raster images as MCP image content, run PDF MVP and PDF text-layer operations, inspect/extract/create ZIP files, and inspect/extract Office/OpenDocument files through the governed Document Worker; no conversion, non-ZIP archives, encrypted ZIPs, delete, OCR, macro execution, formula execution, signing, redaction, or form filling.	Document Worker read/write within the workspace	Parsers and office operations stay within worker boundaries.

5.1 Why Old Tool Names Should No Longer Be Used

Old v0.1 names are still understandable in terms of function, but technically dangerous for prompts and tests, because they suggest a surface that is no longer meant to be public in the v0.2 facade. A project prompt that calls for plwc_compile_profile can fail in an rc18.dev9 session or create false expectations. The correct call is plwc_profile(operation="compile").

For technical documentation the legacy mapping is useful, but for user-facing prompts it should be treated as a negative list: not to be used, not to be taught, not to be advertised as a public API. This reduces support cases and prevents old design decisions from being misunderstood as the current security contract.

6. Request Lifecycle and Policy-before-Execution

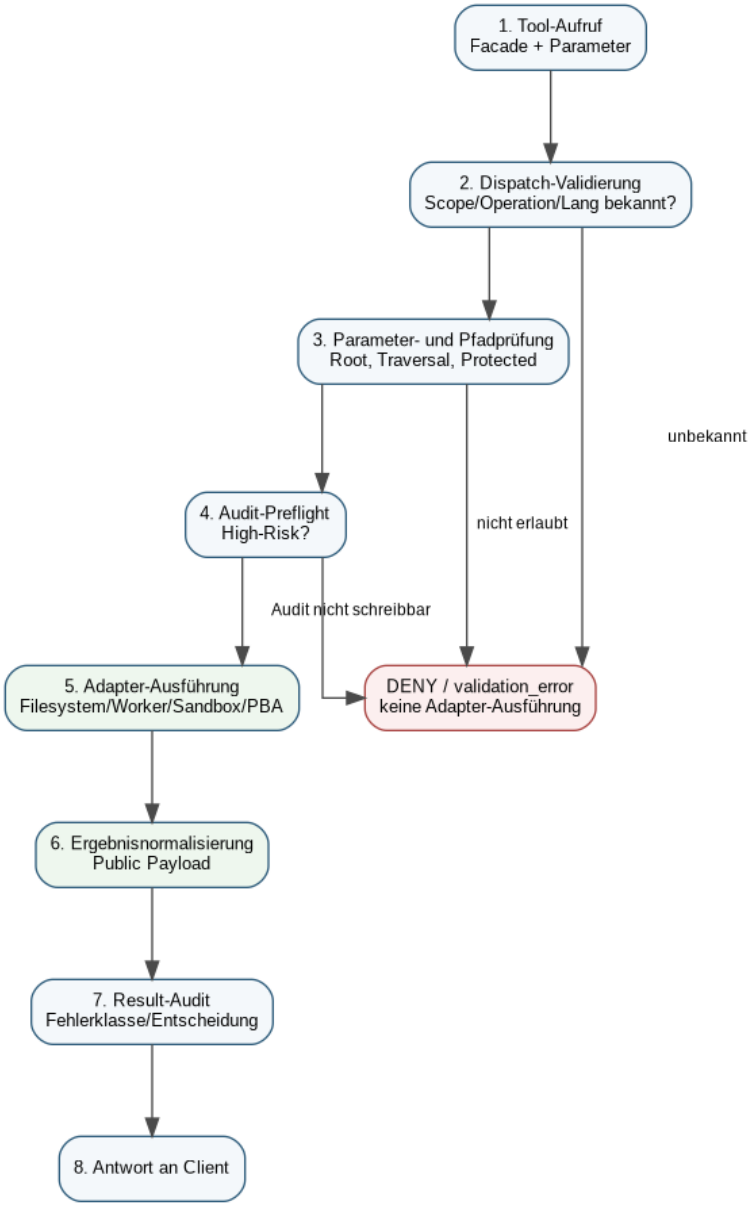

Figure 4: Request processing from tool call through intent, policy, audit, and adapter to result.

Every relevant call goes through the same logical flow. The MCP client calls a facade tool. The facade normalizes parameters, identifies the operation and target category, builds a PolicyIntent from that, and checks it against the policy engine. In parallel, preliminary information for the audit is prepared without moving raw payload data into the audit. Only on Allow is the internal adapter invoked.

This pattern is decisive for the security argument. Security does not arise in any single function, but in the sequence: normalize, classify, check, audit, execute, bound the result. If an adapter were reached before the policy step, the central boundary would be broken.

The deny decision is a normal outcome, not an exceptional state. A good security gateway must be able to refuse predictably. From the user's point of view this is sometimes inconvenient, but technically a clear DENY response is better than a half-attempt that unexpectedly changes something on the host anyway.

6.1 Typical Decision Points

Check step	Question	Example of DENY
Is the tool/operation	Does the operation exist in	plwc_workspace_operation(operation="delete")

allowed?	the current facade?	is rejected if delete is not part of the granted contract.
Are parameters complete?	Are required fields present and unambiguous?	exact_replace without expected_replacements is rejected.
Is the path safe?	Does the target lie within the allowed workspace and outside protected paths?	write to ../profiles/default/PERSONA.md is refused.
Is governance required?	Does it concern persistent profile/memory state?	A direct write to memory.md instead of a governor plan is refused.
Is Docker available and hard-configured?	Can the container run safely with server-owned flags?	Sandbox without Docker produces a Safe Mode deny, no host fallback.
Is output bounded?	Are size, pages, characters, images, or cells limited?	PDF text extraction beyond the max limit is bounded or rejected.

6.2 Error Categories as a Product Feature

A technical description should not treat errors merely as disruptions. In PLwC, structured error categories are part of the product feature set: setup_required, unavailable, denied, validation_error, worker_missing, source_changed_replan_required, unconfirmed_apply_denied, or index_stale are signals that help the user choose the correct next step.

Imprecise errors like "doesn't work" would be poison in this system. A gateway that takes security boundaries seriously must be able to explain whether Docker is missing, a path is protected, a plan needs to be recreated, a profile does not exist, or an operation simply is not part of the capability contract at all.

7. Access Restriction and Path Model

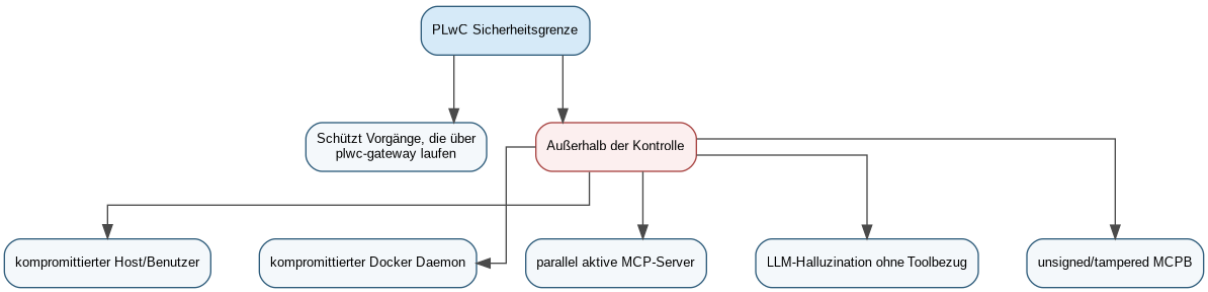

Figure 5: Security boundary between the allowed workspace, protected profiles, and the host system.

The path model is one of the most important parts of PLwC. Normal file operations may only take place within allowed workspace roots. Profile and governance files are not treated as normal workspace files, even if they technically reside on the same disk. The distinction is functional: workspace files are payload data, profile/governance files are control state.

Path normalization must account for parent traversal, absolute host paths, symlinks, overly broad roots, and protected targets. A relative path that looks harmless can otherwise lead out of the workspace. A security gateway must not rely on the assumption that the model will never construct unusual paths.

Protection is write-centric, but not only write-centric. Some profile information may be read for compile/snapshot purposes but not raw via workspace-read. The correct path is a profile tool that specifically delivers a redacted or structured view. Passing protected files through raw file functions would dilute the architecture.

7.1 Protected Governance Files

File / area	Meaning	Direct workspace write	Correct change layer
CORE.md	Identity and core rules of the profile.	forbidden	Not an auto-promotion target; changes only very deliberately, outside normal flows.

PERSONA.md	Working style, role, behavior, context patterns.	forbidden	plwc_reflection + plwc_governor persona_promotion or profile creation.
TEMPERAMENT.md	Recurring temperament/behavior patterns.	forbidden	temperament_promotion with evidence/marker gates.
memory.md	Persistent, confirmed user-memory entries.	forbidden	memory_promotion / reflection_memory_promotion.
reflection.md	Raw observations and candidates.	forbidden	plwc_reflection(operation="write").
journal.md	Session or diary reference.	forbidden	Explicitly confirmed journal/reflection flows.
compiled_prompt.txt	Derived from profile sources.	forbidden	Reconstructable via compile; not to be hand-edited.
governance/config.yaml	Internal profile governance configuration.	forbidden	Governance/admin process, not a workspace operation.

7.2 Direct Access Restriction versus Governance Restriction

There are two different kinds of restriction. Direct access restriction prevents a file operation from reaching a forbidden path. Governance restriction prevents functionally durable control state from being changed without a plan, evidence, and confirmation. Both layers are necessary.

An example: a normal project DOCX in the workspace may be overwritten if the operation and path are allowed. A PERSONA.md may not be overwritten, even if the user says it would be convenient. The correct process is a governor plan that shows what change is planned, why it is justified, what evidence exists, and what confirmation is required.

8. Workspace Functions in Detail

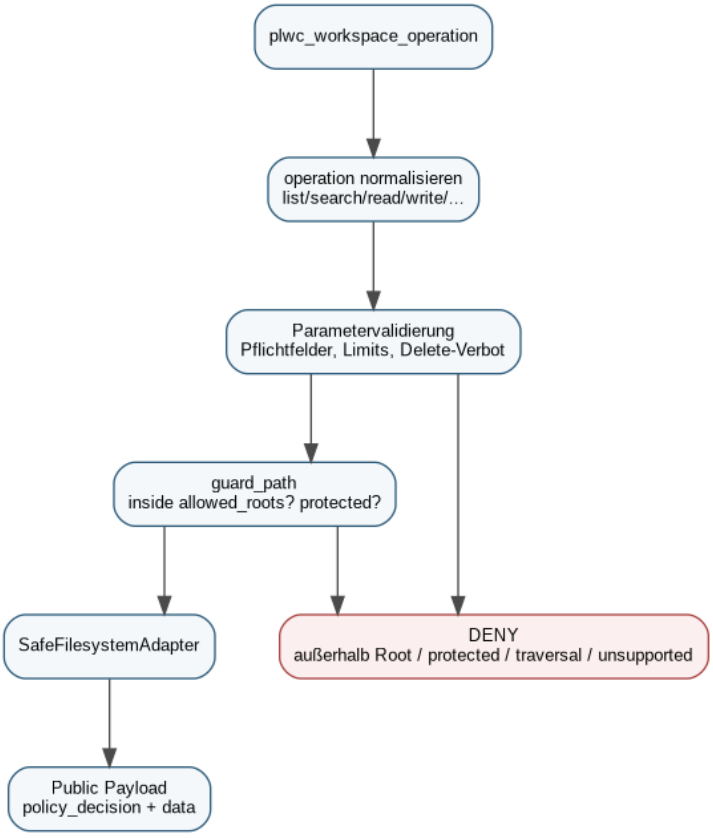

Figure 6: Workspace operations with path checking, protected-path denial, audit, and adapter execution.

plwc_workspace_operation is the central file facade for normal project files. It covers read, write, search, list, file info, directory creation, move/rename, batch read, exact replace, copy, and binary read/write paths. It is deliberately not meant to be understood as a complete file manager.

The most important product decision is: file operations are workspace-relative and bounded. Users should be able to work with project files, but not accidentally expose the PLwC profiles, the host home folder, the repository, or arbitrary system paths. A model generally does not need access to C:\Users as a root. It needs access to a clearly defined working folder.

Exact Replace is more valuable from a security standpoint than fuzzy editing. With exact_replace, the old text must be specified exactly, and expected_replacements defines the expected number of matches. This prevents a model from accidentally changing several similar locations or rewriting a file based on fuzzy assumptions.

Operation	Required parameters	Essential boundary
list	path	Workspace-relative; path="." for root; depth is limited.
search	path, query, max_results	Text search with a bounded number of hits; no host-wide search.
read	path	Text files only, within the workspace; protected paths are refused.
write	path, content, mode	mode rewrite/append; source_path is closed for write; profile/governance protected.
file_info	path	Metadata without exposure outside the root boundary.
create_dir	path	Only safe workspace paths; the alias create_directory is normalized.
move / rename	source_path, target_path	No overwrite by default; both paths must be safe.
batch_read	paths, limits	Max files, max bytes per file, max total bytes guard against mass output.
exact_replace	path, old_text, new_text, expected_replacements	No fuzzy replacements; optional content hash against race/drift.
copy	source_path, target_path, max_bytes	Byte-exact copy within the workspace; no external import.
read_binary	path, max_bytes	Base64 output with a size limit.
write_binary	path, content_base64, mode, max_bytes	Base64 input with a limit; no raw paths outside the workspace.

8.1 Delete as a Deliberate Boundary

The v0.2-line documentation does not treat delete as a normal workspace capability. From a user perspective this is less convenient, but it makes sense for an RC. Deleting is a high-risk operation because a model can chain mistakes: wrong folder, wrong assumption, misreading of a filename. Move/rename and copy cover many working cases without ever permanently destroying anything.

If delete is added later, it should not simply be appended as operation="delete". A sensible approach would be a dedicated trash/archive mechanism with preview, counting, path listing, size information, and explicit confirmation. A security gateway must introduce dangerous convenience gradually, not through a back door.

9. Document, Office, PDF, ZIP, and Image Functions

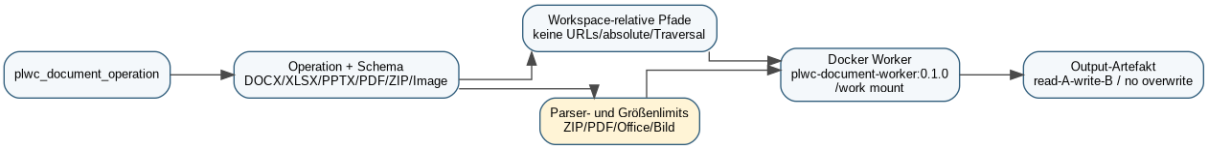

Figure 7: Document Worker flow: facade, path checking, worker container, output artifact.

plwc_document_operation bundles the document capabilities. The Document Worker creates and inspects DOCX, XLSX, PPTX, and PDF, extracts text from Office/OpenDocument files, performs bounded PDF operations, works with ZIP archives, and reads raster images from the workspace as MCP image content. These capabilities are useful but parser-heavy. That is why they do not belong in the model as raw host operations.

The most important boundary is: all input and output paths are workspace-relative. External URLs, absolute host paths, and parent traversal are not part of the contract. For dangerous operations, output_path should not already exist; in-place edits are avoided. This read-A/write-B pattern is safer because the original state is preserved.

The Document Worker is a worker, not a complete office automation suite. It does not replace a LibreOffice/Pandoc conversion farm, an OCR system, PDF redaction, a digital signature, form filling, or macro execution. This boundary is especially important with documents, because many users incorrectly infer from "can generate DOCX" that it can "perfectly edit any Word document."

Area	Supported capabilities	Explicit boundaries
DOCX	create_docx V2, legacy inline, JSON input_path, edit_docx read-A/write-B, inspect/extract.	No in-place edit, no .docm, no raw XML/HTML, no macro execution, no full desktop publishing.
XLSX	create_xlsx V2, multi-sheet, formatting, freeze panes, auto filter boolean, formulas with safe policy, inspect/extract.	No formula calculation, no unsafe HYPERLINK/external URL formulas, no macros.
PPTX	create_pptx, inspect/extract, notes optional on extraction.	No full PowerPoint layout engine, no animation/macro/media execution.
PDF	create_pdf, inspect, merge, split, extract, rotate, extract_pdf_text.	No OCR, no redaction, no signature, no form processing, no free conversion.
ZIP	inspect_zip, extract_zip, create_zip.	ZIP only, no encrypted ZIPs, no non-ZIP archives, Zip Slip is refused.
Images	read_image for PNG/JPEG/WEBP/GIF first frame, resize_to, MCP ImageContent.	Size limits; no raw base64 flood in text blocks; no arbitrary host images.

9.1 DOCX Creation V2 as an Example of a Safe Document API

DOCX V2 shows the basic pattern of good document APIs in PLwC: the model does not write arbitrary XML fragments; instead it supplies a bounded JSON specification. This specification describes document metadata, page setup, styles, paragraphs, headings, lists, tables, images, and page breaks in a controlled structure. Unknown fields, contradictory input, and unsupported schema versions are meant to be handled fail-closed.

This makes the function less free than Word itself, but significantly more reviewable. For automatically generated technical documents, this restriction is an advantage: fewer degrees of freedom mean fewer layout and security surprises. For highly designed reports or print publishing, a manual or specialized layout process is still needed.

9.2 XLSX V2 and Formula Policy

XLSX V2 allows structured workbooks, multiple sheets, formatting, column widths, freeze panes, and bounded formulas. The formula policy is conservative: PLwC should not generate external calls, HYPERLINK tricks, or active content in spreadsheets that could exfiltrate data or trigger unexpected actions later when opened.

The "no formula calculation" boundary is equally important. A generated XLSX file can contain formulas, but PLwC is not Excel's calculation engine. Anyone who needs computed results must either compute them separately or open the file in a suitable spreadsheet application. PLwC should not pretend to be Excel in a container costume.

10. Sandbox, Docker, and Node Execution

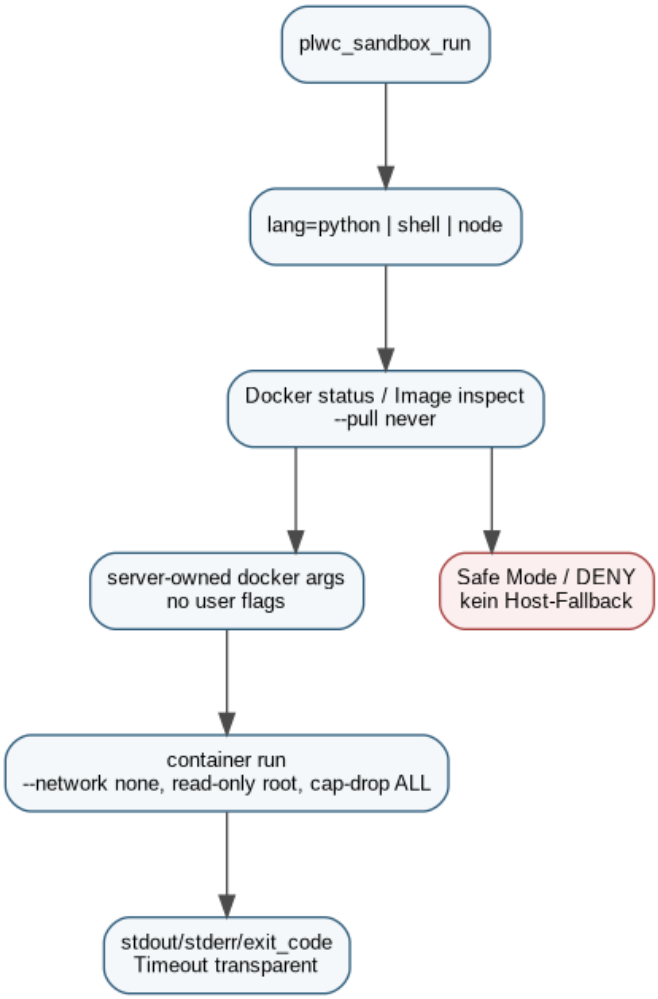

Figure 8: Sandbox flow with Docker checking, policy, container, and result bounding.

plwc_sandbox_run runs small Python, shell, or Node tasks in Docker. The decisive sentence is: there is no host-shell fallback. If Docker is missing, not running, or the image/mount policy is not satisfied, execution is refused. A security gateway that silently falls back to the host shell on error would be worse than having no gateway at all.

The Docker arguments belong to the server, not to the model. The model may not freely set image, mounts, network mode, privileged flag, capabilities, Docker socket, user, memory, or CPU limits. These parameters are part of the security architecture. If they were taken from the tool call, a prompt could specifically weaken them.

Python and shell use a Docker sandbox; Node uses the separate plwc-node-runner. For Node, the code is typically a workspace-relative .js path rather than arbitrary inline JavaScript. This makes it clearer which file is being executed, and the workspace remains the defined exchange surface.

Docker hardening	Meaning	Why relevant
--network none	No network inside the container.	Reduces exfiltration, SSRF, and unplanned downloads.
--read-only	Root filesystem is read-only.	Prevents persistent changes to the container filesystem.
tmpfs /tmp with noexec,nosuid,nodev	Controlled temporary area.	Limits execution and device abuse via temp.
--cap-drop ALL	No Linux capabilities.	Minimizes kernel/namespace privileges.
no-new-privileges	No privilege escalation.	Closes typical escalation paths inside the container.
non-root user	Container does not run as	Reduces damage from process errors.

	root.	
Memory/CPU/PID limits	Resource bounding.	Protects the host against infinite loops and process bombs.
exactly one /work mount	Only the validated workspace is bound.	No Docker socket, no home, no host root.

10.1 Safe Mode

Safe Mode is not a failure state of the entire system, but a reduced operating mode. If Docker is missing or unusable, workspace- and profile-protected functions can remain available, but sandbox and Document Worker are restricted or disabled. Communication to the user must clearly state this distinction.

The correct message reads, roughly: PLwC is running, but Docker-based capabilities are not ready. Start Docker Desktop, check docker info, and restart PLwC or the local MCP client. No wording should suggest that PLwC would instead run the code locally on the host.

11. Profile Model and Persona Management

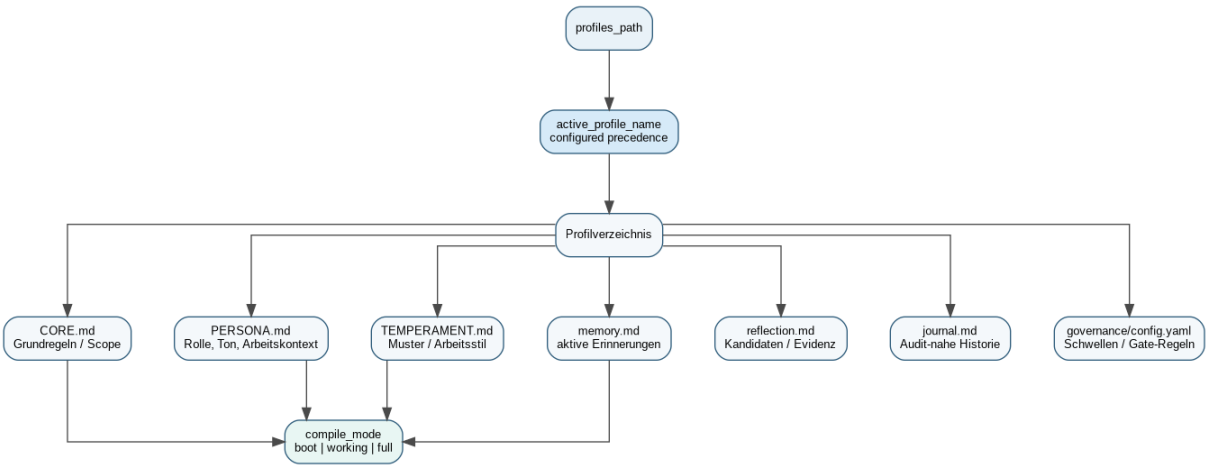

Figure 9: Profile model with CORE, PERSONA, TEMPERAMENT, memory, reflection, journal, and compiled_layer.

The profile model is PLwC's functional distinctive feature. It is not just about file and tool access, but about controlled continuity: working style, stable preferences, memory entries, recurring patterns, and governance rules should remain traceable across sessions. This persistence is useful but also risky. That is why it must not be described or overwritten by arbitrary chat whims.

A profile consists of several layers. CORE.md forms the hard base and is not an auto-promotion target. PERSONA.md describes role, working style, and behavioral anchors. TEMPERAMENT.md collects recurring patterns. memory.md contains confirmed, reusable facts. reflection.md contains observations and candidates. journal.md or diary sources supply history and context. From these sources, compile produces the compiled_layer.

The compiled_layer is the session's active working contract. It must not be fabricated. If compile fails, the session must explain the error state and check first-run/status. A model that simply hallucinates a matching persona when a profile is missing would sabotage PLwC's core purpose.

11.1 Compile Modes

Compile mode	Purpose	Qdrant behavior	Typical use
boot	Bounded, compact starting context.	No Qdrant call.	Session start, standard operation.
working	Task-specific working context with task_context.	May optionally use fresh Qdrant retrieval, if enabled and	Complex tasks where relevant memory

		appropriate.	fragments need to be searched for.
full	More complete diagnostic/audit view.	No automatic Qdrant call per the manifest description.	Review, debugging, profile inspection.

11.2 Persona-Layer Deactivation

rc18.dev9 includes persona_layer_disabled as a configuration option. When this switch is active, the PERSONA block is omitted from compile output by default. This can be useful for Odysseus or other explicit context-injection PoCs, where another instance injects the persona layer or the system is meant to deliberately operate without a persona.

Important: persona_layer_disabled does not deactivate governance, path protection, memory rules, or security gates. It affects the output of the persona block within the compile context. The gateway's hard boundaries remain active. This distinction must stay clear in the documentation, or the switch could be mistakenly understood as a global security off-switch.

11.3 Active Profile Precedence

The active_profile_name configured through the MCP client or security.yaml is authoritative. If a profile is created via the governor but the client still selects a different active profile, PLwC must not act as if the new profile is effectively active. It must report that activation is blocked by the configured precedence.

This rule matters for onboarding. Without it, a user could create a profile, see a success message, and still keep working with a different profile. That would be hard to debug and would destroy trust. The correct response must distinguish between "profile was created" and "profile is the effectively active compile context."

12. The Governance Layer in Detail

Figure 10: Governor plan/apply flow with evidence, review, confirmation, apply, and audit.

The governance layer is PLwC's centerpiece. It does not decide whether a single working file may be written, but whether durable profile, memory, or persona state may be changed. The difference is substantial: a working file can be a draft. A memory entry changes future behavior. A persona change changes the way collaboration works. A governance configuration changes the rules themselves.

Governance follows a two-stage plan/apply principle. Plan produces a preview, an assessment, a target, evidence, conflict information, and planned writes. Apply only executes once confirmed=true is set and the remaining gates are satisfied. This is not a tedious dialog box — it is a technical lock against unnoticed persistence.

It is especially important that force=true is not understood as a universal override. Force can administratively override certain thresholds, such as insufficient_marker or insufficient_evidence, but it must not bypass confirmation, semantics, source integrity, protected targets, or plan/apply consistency. A force that overrides everything would be a second, unsafe path.

12.1 Plan Types and Their Functional Meaning

Plan type	Goal	Typical precondition	Mutation on apply
profile_creation	New profile / onboarding profile.	Structured onboarding_answers, validation, no silent unknown fields.	Creates profile files; effective activation depends on active_profile precedence.

profile_activation	Switch the active profile.	Target profile exists and is valid; confirmation.	Writes PLwC's own active state, unless overridden by extension config.
profile_import	Import a profile.	Explicit source, conflict checking, confirmation.	Imports a profile in a controlled manner, no silent overwrite.
memory_promotion	Entry into memory.md.	Confirmed evidence, trust, target section, semantics.	Persistent memory entry.
reflection_memory_promotion	Reflection candidate to memory.md.	Reflection source, target/candidate_for consistency.	Promotes a qualified reflection into memory.
persona_promotion	Change to PERSONA.md.	Higher evidence threshold, conflict checking.	Changes the persona layer.
temperament_promotion	Change to TEMPERAMENT.md.	Pattern/marker evidence, threshold.	Changes the temperament layer.
reflection_condensation	Condensation of old reflection candidates.	Plan ID, pending plan, confirmation.	Writes a condensed summary and/or marks candidates.
memory_retirement / retire	Retire memory entries.	Retirable list, targeted selection, confirmation.	Sets memory lifecycle to retired rather than hard-deleting.
reindex / drop_index	Manage the Qdrant index.	Qdrant enabled, runtime gate, staleness context.	Reconstructs or removes the derivable index, not the memory source.

12.2 Apply Rules

Apply is mutation-bearing. That is why confirmed=true is required. confirmed=false or a missing confirmation produces unconfirmed_apply_denied. This rule may not be lifted by force or by convenient prompt logic. The user must know that persistent state is now being changed.

For diary- or source-based plans, provenance is decisive. source_file, source_heading, and source_sha256 serve to fix the plan's basis between plan and apply. If the source has changed in the meantime, a new plan is required. This source-integrity check prevents an old approval from being applied to a changed source.

Pending plans must be stored outside the workspace and profile root. This prevents normal workspace operations from manipulating governor state. Here too, the separation of state spaces is part of the security architecture.

Gate	Bypassable by force?	Rationale
confirmed=true	No	Explicit approval is the core of the mutation-bearing contract.
Semantic check	No	Garbage, pure technical logs, or autonomous self-descriptions should not be persisted.
Protected target	No	CORE and governance files must not be circumvented via promotion.
Source SHA mismatch	No	A changed source requires a new plan.
Insufficient evidence	Limited	Admin override possible, but must remain auditable.
Insufficient marker	Limited	Can be overridden in justified cases.

13. Reflection, Memory, Persona, and Temperament Promotion

plwc_reflection(operation="write") is the controlled entry point for observations. A reflection is not automatically memory. It is a source or a candidate. That is conceptually important: people express many short-lived wishes,

moods, technical interim states, and experiments in chat. Not all of that belongs permanently in memory.md or PERSONA.md.

The reflection layer allows evidence to be collected without immediately changing durable profiles. Multiple similar observations can form supporting evidence. A governor plan can later decide whether this should become a memory, persona, or temperament entry. This creates a buffer between a fleeting session and long-term control.

The semantic check is indispensable here. Purely modal technical logs such as "tool X was called," or autonomous self-descriptions by the AI, should not be persisted as personality or memory. PLwC should collect stable user-, work-, or system-relevant insights, not the diary of a tool call.

13.1 Promotion Targets

Target	What belongs in it	What does not belong in it
memory.md	Stable facts and preferences that concretely improve future answers.	Momentary mood, one-off task, unconfirmed guess.
PERSONA.md	Working style, role, interaction patterns, confirmation boundaries, way of collaborating.	Individual project facts, temporary to-dos, technical smoke results.
TEMPERAMENT.md	Recurring patterns in response behavior, tone, initiative, correction style.	A one-off wish without repetition or evidence.
reflection.md	Observations, candidates, evidence, interim states with possible reuse.	Raw logs, shell output, full document contents.
CORE.md	Hard identity and base rules.	Not an auto-promotion target; changes only as a deliberate exception process.

13.2 Conflicts and Staleness

Memory and persona can become outdated or contradictory. PLwC must not resolve such conflicts by blindly appending. A new entry that contradicts an active entry needs review_required, retirement, or a clear conflict resolution. Otherwise the compiled_layer would become inconsistent and future answers would waver between old and new rules.

Retirement is better than deletion. A retired memory entry remains auditable but no longer influences the active context. This corresponds to good configuration management: history is preserved, active state stays clean.

14. Onboarding, First Run, and Profile Switching

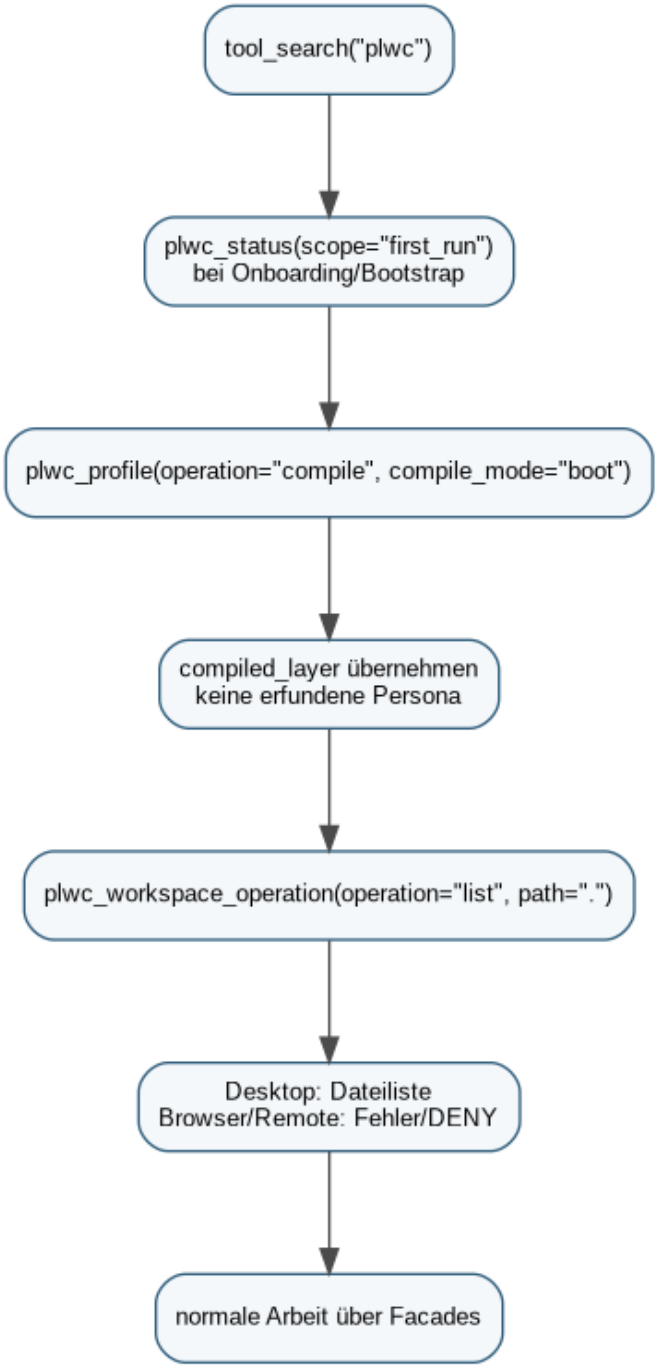

Figure 11: Session start with tool_search, profile compile, workspace list, and environment classification.

The session start is operationally important. A PLwC session should first make the tools visible, compile the active profile, and check the workspace. Only after that is it clear whether local PLwC functions are actually available in this session. A browser chat or remote context cannot simply substitute for local MCPB functions.

First run must not be a manual JSON/YAML ordeal. The package is meant to supply the necessary information via plwc_status(scope="first_run") and plwc_describe(scope="profiles"): greeting, status, profile root, workspace root, Docker readiness, onboarding schema, next steps, and relevant client-configuration hints.

Profile creation runs via plwc_governor(operation="plan", plan_type="profile_creation") and a confirmed apply. While answers are being gathered, "noted" is correct. "Saved" is only correct after a successful apply. This linguistic precision is not pedantry — it prevents users from developing a false sense of security.

14.1 Canonical First-Run Sequence

1. 1. Run tool_search("plwc") if PLwC tools are not yet visible.
2. 2. Call plwc_status(scope="first_run") and evaluate readiness/onboarding_status.
3. 3. Call plwc_profile(operation="status") to check the active, configured, and existing profile.
4. 4. Use the onboarding questions from the supplied schema; do not invent fantasy questions as a mandatory schema.
5. 5. Run plwc_governor(operation="plan", plan_type="profile_creation", onboarding_answers={...}).
6. 6. Display the plan, validation, planned writes, and any unknown_fields or missing_required_fields.
7. 7. Obtain explicit user confirmation.
8. 8. Run plwc_governor(operation="apply", plan_type="profile_creation", confirmed=true, onboarding_answers={...}).
9. 9. Run plwc_profile(operation="compile") again and check the effective active-profile precedence.

14.2 Profile Switching

A profile switch is not a direct change to a file and not a covert switch triggered by a chat statement. The clean path is: check available profiles, plan profile_activation, obtain confirmation, run apply, and recompile. If the MCP client configuration forces a different profile, this precedence must be visibly reported.

A chat command does not end the active PLwC profile. To stop using PLwC altogether, the extension or stdio MCP entry must be disabled, or the project environment changed. Within an active PLwC session, the loaded profile remains the working context unless a clean profile switch occurs.

15. Qdrant Retrieval, Staleness, and Reindex

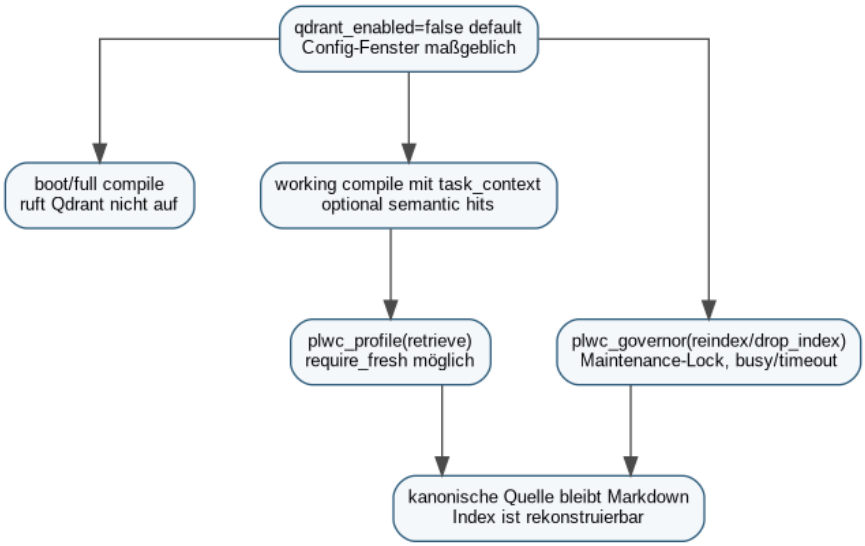

Figure 12: Qdrant as an optional, reconstructable retrieval index alongside canonical profile/memory files.

Qdrant is optional in rc18.dev9 and disabled by default. This is a good architectural decision. A vector index is useful for retrieval but unsuitable as the sole source of truth. The canonical data remain the profile and memory files under governance. The index may be derived from them and reconstructed as needed.

The staleness logic is security-relevant. If memory.md or relevant sources have changed, an old index must not silently keep delivering false hits. A structured response such as index_stale/refused_stale with empty hits is preferable. That looks strict, but it prevents a model from building on outdated semantic hits.

Reindex is likewise a controlled operation. A reindex can take time and must not cause transport stalls. The RC16/RC18 smokes treat structured timeout/busy states as an important quality trait: a running reindex should report qdrant_reindex_busy instead of blocking other tool calls or leaving the session hanging ambiguously.

Qdrant statement	Meaning for PLwC
off by default	Without deliberate activation, the system stays file-based and more deterministic.
optional	PLwC capabilities must not fundamentally depend on Qdrant.
reconstructable	The index can be rebuilt from canonical sources.
not canonical memory source	Memory truth lies in governed files, not in the vector store.
boot/full compile — no Qdrant	Session start stays bounded; working compile can use task-related retrieval.
no automatic reindex	Reindex does not happen quietly in the background.

16. CLU Doctor and Diagnostic Mode

CLU Doctor is to be understood as a read-only diagnostic mode. It is not the active persona, not the normal assistant, and not a write-capable agent. Its purpose is inspection: consistency, evidence separation, memory candidates, smoke results, workspace policy, and release risk. The diagnostic may classify and propose, but must not itself mutate persistent state.

This is architecturally clean. Diagnosis and mutation should be separate. A doctor that both analyzes and writes could persist its own assessment directly. A read-only doctor, by contrast, can deliver firm findings that are subsequently evaluated through normal governor flows.

For external reviewers, CLU is interesting because it provides a second, controlled view of the system. Even so, it does not replace a security review, code analysis, or tests. It is a tool for structured diagnosis, not a judge with a root password.

Doctor aspect	Allowed	Not allowed
Consistency check	Evaluate rules, profiles, smoke results, evidence, and conflicts.	Directly change CORE/PERSONA/memory.
Classification	Provide PASS, FAIL, NEEDS_EVIDENCE, BLOCKED, or comparable findings.	Automatically promote findings.
Risk analysis	Name release, workspace, policy, and governance risks.	Replace a security clearance.
Memory candidates	Review and justify candidates.	Adopt candidates without a governor plan.

17. Audit, Traceability, and Operational Evidence

Audit in PLwC is not a data vacuum cleaner. Audit events are meant to make it traceable which tool calls, policy decisions, denials, high-risk writes, sandbox states, and governance actions occurred. But they must not store raw content, secrets, shell code, stdout/stderr, or full profile contents.

The audit schema is designed to be metadata-only. Permitted fields include timestamp, event, tool_name, action_type, target_category, high_risk, policy_decision, requirement_ids, result_status, error_category, and bounded metrics. That is exactly the right balance: enough information for traceability, without an unnecessary copy of sensitive content.

High-risk writes and sandbox execution should be fail-closed if audit preflight cannot be redacted or written safely. That is strict, but consistent. If the system cannot document a risky operation in a traceable way, it should not execute it quietly.

Audit event	Purpose	No raw data
allowed tool call	Evidence of a permitted operation.	No full arguments.
denied tool call	Explanation and evidence of denials.	No sensitive path/content details.
protected path access	Security-relevant attempt.	No full host path in plain text.

attempt		
sandbox unavailable	Diagnosis of Docker/Safe Mode.	No executed code.
governance update accepted/rejected	Traceability of profile/memory mutations.	No full evidence in the audit.

18. Installation and Operation by Client

Claude Desktop uses the packaged MCPB/extension path. Local GPT-compatible MCP clients and Odysseus use the same local gateway over stdio: extract the verified MCPB, register the bundled server.py with an absolute Python executable, and expose only one server named plwc-gateway. Hosted ChatGPT web is a different deployment class and is not directly installable from this local stdio package; it requires a future authenticated remote MCP/custom-app facade.

Docker Desktop is required for the sandbox and Document Worker. Without Docker, PLwC must not quietly fall back to host functions. For normal workspace and profile-status functions, the gateway can still be useful. Testers must therefore distinguish between "PLwC is running" and "all Docker-based capabilities are ready."

The Docker images python:3.12-slim, plwc-document-worker:0.1.0, and plwc-node-runner:0.1.0 must be present locally to the extent the respective capabilities are used. Since the runtime should not force implicit pulling at runtime, image provisioning belongs in the installation guide or in an accompanying image-tar procedure.

18.1 Installation Paths for Open Beta Testers

10. Save the Dev9 MCPB locally and verify its SHA256 against the release documentation.
11. Choose the client path: Claude Desktop MCPB installation, or extracted-package stdio for a local GPT-compatible client or Odysseus.
12. For Claude Desktop, install the MCPB extension. For stdio, extract the package and register its bundled server.py using an absolute Python executable.
13. Restart the client and verify that exactly one server, plwc-gateway, is visible with exactly eight facade tools.
14. Configure workspace_path and profiles_path as separate directories, or use the safe user-scoped defaults.
15. Install and start Docker Desktop if sandbox or Document Worker functions are needed.
16. Provide python:3.12-slim locally.
17. Build or load plwc-document-worker:0.1.0 and optionally plwc-node-runner:0.1.0.
18. Start a fresh client session, discover plwc-gateway, and check plwc_status(scope="runtime") plus plwc_describe(scope="tools").
19. Compile the configured profile, list the workspace, check Safe Mode, and run the minimal client smoke tests.

```
Local stdio registration:
name = plwc-gateway
transport = stdio
command = <absolute path to python.exe/python3>
args = [<absolute path to extracted package>/server.py]
Do not register raw PBA, Commander, filesystem, or host-shell servers alongside PLwC.
```

19. Smoke Tests, Release Gates, and Acceptance

Smoke tests must cover three levels: package findings, actual client behavior, and security-relevant negative cases. A package can be correctly built and still be registered incorrectly in a client. Conversely, a client test can appear to work on the surface while a protected-path deny is missing. Dev9 records package, Claude Desktop, and Odysseus smoke PASS; the local GPT stdio route is maintainer-confirmed. Environment-specific acceptance still requires the same independent gates.

Minimum acceptance for the Open Beta should not just be "tool responds." It should check tool registry, profile compile, workspace root, protected-write deny, sandbox Docker status, Document Worker generation, governance plan/apply, and a negative old tool name. Only once both allow and deny cases are plausible is the gateway boundary credible.

19.1 Minimum Installation Smoke Test

Test	Call	Expectation
Tool discovery	tool_search("plwc")	plwc-gateway visible; eight facade tools.
Runtime status	plwc_status(scope="runtime")	Server, version, paths, profile status, Safe Mode notes.
Describe	plwc_describe(scope="tools")	Current facade operations; no old v0.1 tools as public.
Profile compile	plwc_profile(operation="compile")	compiled_layer or a clear setup_required/unavailable state.
Workspace list	plwc_workspace_operation(operation="list", path=".")	File list on local client access; DENY/error on hosted/remote/no-local-access.
Protected write	write to PERSONA.md	DENY, no file change.
Sandbox	plwc_sandbox_run(lang="python", code="print(...)")	Container execution or a structured Safe Mode deny; no host fallback.
Document Worker	create_docx + inspect_docx	Artifact in the workspace, or worker_missing with a clear cause.

19.2 Release Gate Before Distribution

- MCPB hash, file size, and file count are documented and match the delivered artifact.
- FastMCP/public registry shows exactly the eight facade tools.
- No legacy single-purpose tools are publicly visible.
- Workspace and profile root are separated; broad roots are refused.
- Protected-path writes are refused.
- Docker Safe Mode and Docker Mode are both understandably tested.
- Document Worker image is present, or the missing state is cleanly documented.
- A governance plan without a confirmed apply does not mutate anything.
- A confirmed apply only writes the expected targets and is auditable.
- Qdrant stays optional; stale retrieval is not silently accepted.

20. What PLwC Cannot Do

This section is deliberately blunt. PLwC is a local governance and security boundary for MCP tool access. It is not a comprehensive safeguard against incorrect model answers, no guarantee against every prompt attack, and no substitute for organizational approvals. Anyone selling PLwC as a magic protective layer is building in the next disappointment from the start.

PLwC can only control calls that actually pass through plwc-gateway. It cannot prevent a parallel, raw MCP server from doing the same or more dangerous things without PLwC's policy. It also cannot guarantee that the model, even with a correctly loaded profile, never writes nonsense. What it can do is technically bound persistence, tool access, and risky operations.

Non-goal / boundary	Why it matters	Consequence
No final v1.0 / no signature	A release candidate is not production-certified.	Only controlled testing; document hash and release cleanly.
No protection for other MCPs	PLwC only sees its own tool calls.	No parallel file/shell MCPs in PLwC projects.
No automatic end of session	MCP does not detect a real end of chat.	Trigger journal/diary entries only explicitly.
No host shell	Host execution would be high risk.	Docker missing = Safe Mode / DENY.
No free choice of Docker parameters	The model could weaken the sandbox.	Server-owned Docker policy.

No complete office suite	Document Worker is a bounded builder/inspector.	Check complex layout manually.
No OCR	OCR is expensive, error-prone, and its own risk area.	Scans are not automatically made searchable.
No PDF redaction/signature/form filling	Specialized, legally/technically critical operations.	Use external, vetted tools.
No formula calculation	XLSX generation is not Excel.	Verify calculated values separately.
No macro execution	Macros are active code.	Reject docm and active content.
No guarantee against hallucinations	Profile context improves behavior but does not replace verification.	Check technical facts and research anything that may have changed.
No DMS / no permission management for teams	Local gateway, not an enterprise DMS.	Resolve team sharing separately.

21. Risks, False Assumptions, and Technical Countermeasures

Risks do not only live in code. Many risks come from incorrect operating assumptions: users believe PLwC also protects other MCPs; developers believe a green unit test replaces a desktop smoke test; documentation mentions an old tool name; a profile is created but is not active because of active_profile_name; a Qdrant index is confused with memory. These risks are at least as practical as classic exploits.

The following matrix bundles technical and operational risks. It is deliberately longer than a normal summary, because it helps reviewers check the security argument and prioritize open items.

ID	Risk	Impact	Countermeasure
R1	Second visible MCP server	Policy bypass via direct file/shell tools.	Activate only plwc-gateway in the project; registry smoke.
R2	Direct write to governance files	Persona/memory can be corrupted or manipulated.	Protected paths and governor flows.
R3	Host-shell fallback	Full host compromise possible.	Fail closed; Docker-only; Safe Mode.
R4	Docker parameter injection	Network, mounts, or privileges could be opened.	Server-owned Docker args; no model parameters.
R5	Overly broad workspace root	The model gains too much of the filesystem.	Refuse root/home/source/governance paths.
R6	Audit leakage	Secrets or content end up in the audit.	Metadata-only audit, redaction, high-risk fail-closed.
R7	Stale Qdrant index	Outdated hits influence answers.	Staleness detection, refused_stale, reindex.
R8	Active profile misunderstanding	New profile exists but is not effectively active.	Explicitly report configured_active_profile precedence.
R9	Prompt with old tool names	First run fails, or user learns the wrong API.	Use the Dev9 prompt with facade names.
R10	Document Worker mistaken for an office suite	Overinflated expectations for layout and conversion.	Document the capability contract and non-goals.
R11	Memory clutter from fast promotion	Short-term statements become durable context.	Reflection buffer, evidence thresholds, semantic check.
R12	Force as a universal override	Governance could be effectively disabled.	Force only limited; no confirmation/semantic/source bypasses.
R13	Unclear Docker image provisioning	Testers cannot use document/sandbox functions.	Provide an image tar or build instructions.

R14	Symlink/traversal gap	Workspace boundary can be exited.	Symlink-capable tests, path resolution, deny broad roots.
R15	Mock-heavy tests	Tests overestimate real integration.	Package smoke, desktop smoke, real Docker gates.
R16	Persona-layer deactivation misunderstood	User believes governance is off.	Document: compile persona block only, not security.

21.1 Residual Risk and an Honest Assessment

The greatest residual risk lies in operations, not in any single function: if PLwC runs alongside other powerful MCPs, its policy boundary is only one of several possible routes. That must be clearly stated in every guide. PLwC can be a controlled boundary, but not the security chief for every tool a user installs in parallel.

The second-largest residual risk lies in incorrect persistence. Memory and persona are powerful because they influence future behavior. That is exactly why evidence, thresholds, review, and confirmed apply matter so much. An AI that writes its own working style into PERSONA.md uncontrolled would not be progress — it would be a very polite configuration accident.

The third-largest residual risk lies in documents and executable code. Office files, PDFs, ZIPs, and code execution are classic attack surfaces. PLwC reduces these risks through worker boundaries, Docker, path checking, and non-goals. It does not eliminate them entirely.

22. Operational Checklists and Incident Handling

22.1 Day-to-Day Operation

- Always start a new session with tool discovery, profile compile, and a workspace list.
- On unexpected behavior, check plwc_status(scope="runtime") and plwc_describe(scope="tools").
- Before any governance apply, deliberately check the plan content, target, evidence, and confirmed=true.
- For Docker problems, do not use host-shell workarounds — accept Safe Mode and fix Docker.
- For document operations, choose output_path as a new file and visually check the result.

22.2 Incident Scenarios

Symptom	Likely cause	Technical response
PLwC tools not visible	Extension not loaded, wrong session, browser context.	Run tool discovery, restart the client, and check the Claude extension or local stdio entry.
Old tool names fail in prompts	v0.2 facade active.	Switch the prompt to the plwc_profile/plwc_governor facade.
Profile was created, but compile uses a different profile	configured_active_profile has precedence.	Change the extension configuration or correctly check profile activation.
Protected write DENY	Target is a profile/governance file.	Use the governor/reflection flow.
Sandbox unavailable	Docker missing or image missing.	Start Docker Desktop, check images, accept Safe Mode.
worker_missing	plwc-document-worker:0.1.0 is missing.	Build/load the image and run docker image inspect.
source_changed_replan_required	Source changed between plan and apply.	Replan and use the new source_sha256.
index_stale/refused_stale	Qdrant index is not current.	Plan/run a reindex, or proceed without retrieval.

23. Developer View: Internal Modules and Maintainability

The internal structure under src/plwc_gateway separates the MCP server, policy, adapters, audit, config, runtime, and onboarding. This separation matters for maintainability, because security decisions would otherwise be scattered across handlers. The facade may normalize and dispatch parameters, but the actual permission logic belongs in the policy and path modules.

Adapters are meant to stay internal. Filesystem adapter, PBA adapter, sandbox adapter, Document Worker adapter, and Qdrant adapter are implementation details behind the gateway boundary. For any new capability, the question must be: which facade operation represents it, what intent results, which policy gates apply, what audit metadata is produced, and which negative cases are tested?

Release maintainability depends heavily on traceability. A requirement such as RC18-ONBOARDING-001 should be consistently findable in the manifest description, README, tests, smoke report, and user prompt. If the changelog, the smoke file, and the artifact drift apart, release debt accumulates. The computer does not stumble over that, but people reliably do.

Module area	Task	Review question
mcp/server.py	Registers public facade tools and dispatches operations.	Are truly only eight tools publicly registered?
policy	Intent, decisions, path and action rules.	Are there allow paths without an explicit policy?
adapters/filesystem.py	Workspace file operations.	Are traversal, symlinks, and protected paths handled correctly?
adapters/pba.py	Profile, governance, reflection, memory, Qdrant-adjacent flows.	Are plan/apply, confirmed, and provenance handled consistently?
adapters/sandbox.py	Docker sandbox for Python/Shell/Node.	Are Docker args server-owned, with no host fallback present?
adapters/document_worker.py	Document Worker communication and validation.	Are input/output paths bounded and worker errors clear?
audit/logger.py	Metadata-only audit and redaction.	Can raw content or secrets end up in the audit?
config/settings.py	Runtime configuration and defaults.	Are empty config fields safe and user-scoped?

24. Function Reference

The following sections describe the eight facade tools both technically and functionally. This reference is not just a parameter list; it also explains the security decision behind each tool and the boundaries a caller must expect.

24.1 plwc_status

plwc_status is the diagnostic and status facade. It does not change any persistent state. Its task is to explain the current operating state to the user and the model: runtime, sandbox, first-run, and configuration guidance. Especially at first run, this tool is the most important starting point, because it makes visible the difference between a missing profile, missing Docker, incorrect configuration, and normal desktop access.

Scope	Content	Typical use
runtime	Server, version, tool count, active profile source, workspace/profile roots, Safe Mode notes.	Verify that the session is running locally and correctly.
sandbox	Docker CLI, Docker daemon, Python image, Document Worker, Node Runner, mount/audit readiness.	Clarify why code or document operations aren't running.
first_run	Onboarding status, greeting_message, missing_requirements,	Guided profile creation without manual YAML operations.

	profile_onboarding_schema, next_actions.	
config	Claude/local MCP configuration hints and, where supported, ready-to-paste snippets.	Setup help without leaking sensitive raw values.

24.2 plwc_describe

plwc_describe is the current runtime's built-in operating manual. It should be used when a model is unsure which operations, fields, or denial reasons currently apply. That is better than guessing, because the v0.2 facade has historically similar old tool names that are no longer public.

Parameter	Values	Meaning
scope	tools, status, workspace_operation, document_operation, reflection, governor, sandbox, profiles	Selects the help area.
detail	short, full	Level of detail. Use full for technical reviews.
format	json, markdown	Machine-readable or human-readable.

24.3 plwc_profile

plwc_profile is the access point to the active profile. Status and snapshot serve diagnostics, compile produces the active compiled_layer, retrieve and scan_tagebuch support task-related context gathering, and doctor runs diagnostics. Critically: compile must not be replaced by improvisation. A missing profile is a setup state, not an occasion for improvisation.

Operation	Important parameters	Boundaries
status	profile optional	Diagnostic only; no raw file manipulation.
snapshot	profile optional	Structured profile overview, not an arbitrary protected-file read.
compile	profile, task_context, compile_mode, persona_layer	compiled_layer is binding; check compile_mode; respect persona_layer_disabled.
retrieve	query/task_context	Bounded search within the profile context; no free file search.
scan_tagebuch	Parameters for diary scan	Diagnostic/contextual; promotion only via the governor.
doctor	doctor_mode=clu80, doctor_scope	Read-only diagnostic; no mutation.

24.4 plwc_reflection

plwc_reflection writes governed reflection entries. It is not a direct memory write. The parameters summary, evidence, confidence, marker, trust, target, and candidate_for serve to make quality and the later promotion target transparent. target="memory.md" requires candidate_for="memory.md", so that target and candidate do not drift apart.

Parameter	Required / type	Meaning
operation	write	The only regular operation.
summary	string	The core statement of the observation.
evidence	string	Supporting evidence and observations; not stored raw in the audit.
confidence	string	Confidence level of the observation.
marker	string	Observation/pattern; relevant for temperament.
trust	string	Trust level/evidence assessment.
target / candidate_for	strings	Target candidate; consistency checking matters.
entry_date	YYYY-MM-DD, optional	Provenance and chronology.

24.5 plwc_governor

plwc_governor is the critical mutation facade. plan produces an assessment and preview, apply changes confirmed state. Additionally there are diagnostic/lifecycle operations such as list_retirable, retire, and Qdrant-adjacent operations such as reindex/drop_index. Every mutation-bearing operation must be strictly checked against confirmed, plan type, source, and target.

Operation	Meaning	Security rule
plan	Produce a governance plan.	No durable mutation yet.
apply	Apply a confirmed plan.	confirmed=true required; source/plan gates.
list_retirable	List candidates for retirement.	Diagnostic; no deletion.
retire	Retire memory/entries.	Lifecycle instead of hard delete; confirmation checked.
reindex	Rebuild the Qdrant index.	Optional index; timeout/busy handled in structured form.
drop_index	Remove the Qdrant index.	The index is derivable; not a deletion of memory.

24.6 plwc_sandbox_run

plwc_sandbox_run is suited to small execution tasks, not arbitrary host automation. Python and shell receive code/a command; Node receives a workspace-relative .js path. The process runs in Docker with server-owned hardening. Without Docker, it is refused.

Parameter	Values	Note
lang	python | shell | node	Other languages are rejected.
code	non-empty string	For node, a workspace-relative .js path, not free inline JS.
timeout	positive integer, optional	The effective limit remains bounded on the gateway side.

24.7 plwc_workspace_operation

plwc_workspace_operation was already described functionally in Chapter 8. In the function reference, the key point is: every operation must use workspace-relative paths, protected paths are refused, and dangerous convenience functions such as delete or free host paths are not part of the current contract.

24.8 plwc_document_operation

plwc_document_operation was already described functionally in Chapter 9. For callers, the essential point is: document operations are not free conversion or office-automation commands. They are bounded worker operations with clear input formats, limits, and non-goals.

Operation type	Examples	Typical denials
Create	create_docx, create_xlsx, create_pptx, create_pdf	output_path exists, input_path not .json, unknown schema version, absolute paths.
Inspect/Extract	inspect_docx/xlsx/pptx/pdf/zip, extract_*	Output too large, unsupported format, protected path.
PDF ops	merge_pdf, split_pdf, extract_pdf, rotate_pdf, extract_pdf_text	Invalid page ranges, output overwrite, unsupported PDF function.
ZIP ops	inspect_zip, extract_zip, create_zip	Encrypted ZIP, Zip Slip, non-ZIP archive.
Image	read_image	Unsupported format, too large without resize, path outside the workspace.
Edit DOCX	edit_docx	In-place edit, overwrite, .docm, raw

		XML/HTML/formula, unknown edit type.

25. Appendices

Appendix A. Recommended rc18.dev9-Compatible Project Prompt

1. Run tool_search("plwc") if PLwC tools are not visible.

```
4. Treat a successful workspace list as local PLwC stdio access. Treat error or DENY as hosted/remote/no-local-access and report that once.
```

Appendix B. Legacy Mapping v0.1 to v0.2 Facade

Old public tool name	Current v0.2 facade
plwc_runtime_status	plwc_status(scope="runtime")
plwc_sandbox_status	plwc_status(scope="sandbox")
plwc_first_run_status	plwc_status(scope="first_run")
plwc_generate_claude_config	plwc_status(scope="config")
plwc_profile_status	plwc_profile(operation="status")
plwc_profile_snapshot	plwc_profile(operation="snapshot")
plwc_compile_profile	plwc_profile(operation="compile")
plwc_write_reflection	plwc_reflection(operation="write")
plwc_governor_plan	plwc_governor(operation="plan")
plwc_governor_apply	plwc_governor(operation="apply")
plwc_run_python_sandboxed	plwc_sandbox_run(lang="python")
plwc_run_shell_sandboxed	plwc_sandbox_run(lang="shell")
plwc_list_workspace	plwc_workspace_operation(operation="list")
plwc_search_workspace	plwc_workspace_operation(operation="search")
plwc_read_workspace_file	plwc_workspace_operation(operation="read")
plwc_write_workspace_file	plwc_workspace_operation(operation="write")

Appendix C. Negative Capability Contract

The negative capability contract describes what PLwC deliberately must not do. It should appear in tests as a negative matrix, not only in prose. A capability contract without negative cases is half-blind for security software.

Category	Must be refused
Public boundary	A second PLwC-adjacent MCP server, old individual tools as a public API.
Workspace	Absolute host paths, parent traversal, protected paths, broad roots, delete without its own gate.
Governance	Apply without confirmed=true, CORE auto-promotion, direct memory/persona writes.
Sandbox	Host-shell fallback, dynamic images, host network, Docker socket, privileged mode, additional mounts.
Document Worker	OCR, redaction, signature, macros, .docm, encrypted ZIP, non-ZIP archives, raw XML/HTML edits.
Qdrant	Index used as the canonical memory source, silent retrieval from a stale index.
Audit	Raw content, secrets, shell code, stdout/stderr, full profile contents.

Appendix D. Sources in the Examined Package

Source	Use in this description
manifest.json	Version, tool list, user config, long description, runtime mapping.
README.md	Status, feature overview, public tools, capability boundaries.

docs/ARCHITECTURE.md	Public boundary, internal adapters, source boundaries.
docs/TOOLS.md	Facade contract, operations, negative capabilities, smoke notes.
docs/FIRST_RUN.md	Onboarding, active_profile precedence, profile_creation flow.
docs/SAFE_MODE.md	Docker Safe Mode, no host fallback, Docker hardening.
docs/AUDIT_LOG.md	Metadata-only audit, redaction, high-risk fail-closed.
docs/RISK_REGISTER.md	Risks, countermeasures, open items.
src/plwc_gateway/mcp/server.py	Registration and dispatch of facade tools.
src/plwc_gateway/adapters/*.py	PBA/governor, filesystem, sandbox, Document Worker, Qdrant adapter.

Appendix E. Findings on the Previous 23-Page Version

The previous 23-page version was structurally usable but too short for the intended standard. It contained many correct headings, tables, and figures, but too little explanatory prose and too little technical depth on governance, persona, access restriction, and risks. As third-party documentation, it was more of an extended exposé than a detailed software description.

This version therefore expands the description significantly: more background on architectural decisions, more risk analysis, more plan/apply semantics, more persona and onboarding explanation, a more complete negative capability contract, and more operational acceptance. Length alone is not the quality yardstick, but for a security and governance layer, the justification must not be thinner than the tool list.

Appendix F. Engineering Deep Dive by Functional Block

This appendix extends the functional description with a more engineering-oriented view of responsibilities, data flows, error boundaries, and review questions. It is deliberately detailed, because PLwC's core value does not lie in any single spectacular function, but in the combination of a controlled public API, policy, governance, path protection, audit, and bounded execution.

For third parties, this view is often more important than the plain tool list. A reviewer needs to be able to recognize where a decision arises, what data it requires, which states must not be trusted, and what a safe refusal looks like. The following subsections therefore describe not just what a component does, but also which error class it is meant to prevent.

F.1 MCP Server and Public Registry

The MCP server is the first technical control point. It registers the public surface and thereby decides what the model sees as a tool at all. In rc18.dev9, the visible surface must remain bounded to the eight facade tools. Every additional public registration is not merely API cosmetics — it is potentially a security deviation, because it can create a new entry point before or alongside the policy layer.

The public registry must therefore be treated as a release gate. A test that only imports the Python functions is not sufficient. What matters is what build_mcp_server, or the installed MCPB, actually makes visible in Claude Desktop. The registry check must explicitly negative-test old tool names: plwc_compile_profile, plwc_governor_plan, and plwc_write_workspace_file must not appear as public tools.

Tool descriptions are also relevant. rc18.dev9 improves exactly this point: FastMCP metadata should not be empty, so that Claude Desktop and the model can find the tools and classify them correctly. Empty descriptions do not directly lead to an exploit, but they degrade discovery, increase mis-calls, and encourage old or invented names.

A clean public-registry review answers three questions: which tools are visible? Which tool descriptions are shipped? Do the manifest, README, smoke report, and actual registration agree? If any of these answers diverge, the release is not clean, even if individual functions work locally.

F.2 Intent Builder and Policy Decision

Policy can only decide correctly if a usable intent is derived from the tool call. The intent does not describe every detail of the argument, but the security-relevant meaning: which action is to be executed, which target category it

affects, whether it is high-risk, whether it concerns workspace, profile, governance, documents, or containers, and which denial reasons are possible.

This abstraction prevents security logic from being formulated differently in every handler. If write, exact_replace, create_docx, profile_creation apply, and sandbox execution all had their own special checks without a shared decision layer, the system would be hard to review. A gateway needs a unified language for Allow, Deny, Validation Error, and Safe Mode.

The policy decision should be deterministic. The same call, the same configuration, and the same state must lead to the same decision. If a decision depends on the randomness of the model context or on implicit prompt assumptions, it is not a policy but gut feeling with JSON output.

For reviews, it matters to test deny cases just as seriously as allow cases. A green allow test only shows that a function is fundamentally reachable. A green deny test shows that the boundary holds. In PLwC, deny tests such as protected write, host path, old tool names, unsafe Docker flags, and unconfirmed apply are product-relevant.

F.3 Path Resolution and Workspace Boundary

Path resolution sounds boring, but it is a classic vulnerability. A secure workspace adapter must not merely compare strings. It must normalize, detect absolute paths, prevent parent traversal, account for symlink-based escapes, and check whether the final real path stays within an allowed root.

The workspace root must not be chosen too broadly. A user who sets C:\Users as the workspace has, technically, perhaps supplied a valid directory value, but has functionally opened up too large an attack surface. PLwC should treat broad roots, home roots, system paths, source repositories, and profile roots conservatively, and refuse them if necessary.

Protected paths are a second check alongside the workspace boundary. A path can formally lie within an allowed root and still be protected if it concerns profile or governance files. This distinction matters because some users, out of convenience, put everything into one shared folder. The architecture must not confuse that convenience with safety.

The correct user response to a protected-path deny is not to disable path protection, but to use the appropriate governor or reflection flow. If a model proposes a direct change to PERSONA.md, it should switch to the governance path itself. This self-correction is precisely what makes PLwC useful in everyday work.

F.4 Filesystem Adapter as a Controlled Working Area

The filesystem adapter is meant to enable project work, not to manage the host filesystem. It must make normal read and write tasks simple enough that users are not constantly looking for workarounds. At the same time it must remain deliberately incomplete: no arbitrary delete, no host root, no process management, no network operations, and no silent imports from outside.

The batch_read operation is a good example of bounded usefulness. It allows reading multiple files, but with max_files, max_bytes_per_file, and max_total_bytes. Without such limits, a model could accidentally or deliberately pull large amounts of local data into the chat. With limits, the function stays useful but controlled.

copy, read_binary, and write_binary extend usability for artifacts. At the same time, they increase the responsibility for size limits and path checking. Binary data in chat is expensive and potentially sensitive. That is why it is correct to deliver read_image via MCP ImageContent rather than as an unbounded text-base64 block.

Exact Replace is especially valuable for software and document work. It forces the model to know the old state exactly. expected_replacements prevents accidental mass replacements. An optional content hash can additionally catch race conditions where a file changed between reading and writing.

F.5 Document Worker Adapter

The Document Worker separates parser- and format-heavy work from the gateway. Office and PDF files are complex container formats. They can contain embedded content, external relationships, macros, formulas, or surprising structures. A gateway main process should not parse or execute such formats itself in an uncontrolled way.

The worker model is still not a free pass. The worker, too, needs input validation, path boundaries, output_path rules, size limits, and clear operations. It is especially important not to treat raw XML, HTML, or formula input as active code. The document API should stay declarative.

DOCX edit_docx shows a healthy direction: read-A/write-B instead of in-place edit, bounded edit types, a maximum of 200 operations, no .docm files, no macro execution, defusedxml for XML, and output_path must not already exist. These rules protect against data loss and active content without making simple edits impossible.

For PDF, the boundary must be especially clear. Text-layer extraction, split, merge, rotate, and inspect are something different from OCR, redaction, signature verification, or legally sound blackout. Redaction is security-critical, because a visible blackout without removal of the underlying text would be dangerous. It is better not to offer this capability at all than to offer it halfway.

F.6 PBA/Profile Adapter and Persistent Context

The PBA/profile adapter is functionally the most sensitive adapter. It works with the state that influences future sessions. An error here is longer-lived than an error in a generated working file. That is why profile compile, snapshot, reflection, promotion, retirement, and active-profile selection must be tightly governed.

Compile is a derivation from profile sources. The compiled_layer is not itself the truth, but a bundled session representation. If it becomes outdated or is generated incorrectly, it affects answer quality. That is why it must arise from known files and configurations, not from the model's imagination.

Profile activation is not a trivial string switch. There are configured active-profile sources and PLwC's own active-state files. rc18.dev9 handles the case where a Claude Desktop configuration takes precedence. This precedence prevents silent surprises but requires clear user communication.

Profile import, profile creation, and profile switching must each remain their own plan types. An import can overwrite existing control files, a creation generates new files, an activation changes selection state. These operations carry different risks and must not blur into a generic "just write the profile."

F.7 Sandbox Adapter and Execution Policy

The sandbox adapter is the part where a wrong decision tips fastest into real host danger. Code execution is powerful. The difference between container and host is the security boundary here. That is why the rule must stay hard: no Docker, no execution. No hidden subprocess on the host, no shell exception, no convenient debug shortcut.

Docker hardening must be demonstrable in tests. Documented flags are good, but only effective if the adapter actually sets them. Especially dangerous would be network access, a Docker socket mount, a host home mount, privileged mode, or dynamic images taken from the model call. Each of these would be a candidate for a critical finding.

Node execution increases complexity, because JavaScript ecosystems are often tied to npm, node_modules, and package resolution. The Node Runner should therefore stay minimal. No npm install at runtime, no network dependency, and required node_modules must be deliberately present in the workspace. That is less convenient but more reproducible.

Timeout, memory, and PID limits are not just DoS protection. They also help with user error. An accidental infinite loop must not bring down the host. A tool that executes code must treat normal programming mistakes as an expected operating case.

F.8 Audit Logger and Redaction

The audit logger must balance two conflicting goals: traceability and data minimization. If it stores too little, policy decisions cannot later be reviewed. If it stores too much, the audit itself becomes a data leak. PLwC resolves this tension through metadata-only events and recursive redaction.

Fields such as content, code, evidence, stdout, stderr, compiled_layer, snapshot, command, docker_args, and path-like fields are especially critical. These values can contain payload data, secrets, or private context. They do not belong raw in an audit log. Bounded metrics such as character count, hit count, or result status are sufficient for many reviews.

High-risk operations should be fail-closed if the audit cannot function safely. That sounds strict, but a gateway that executes risky writes without safe evidence loses part of its governance property. The user must later be able to check what happened without having to accept sensitive content in the audit for that purpose.

Audit must not be confused with memory. Audit documents operations. Memory stores reusable facts. If audit data were to flow into memory, a mixture of technical trace and behavioral control would result. This separation should stay clear in the code and in prompts.

F.9 Settings, Defaults, and Safe Defaults

Safe defaults matter especially with MCPB, because normal users do not start with an empty terminal and a complete configuration file. They install a package in Claude Desktop and expect it not to operate in System32, a repo root, or a home root. Empty configuration fields must therefore fall back to safe, user-scoped paths, not to the current starting directory.

The threshold values memory_write_threshold, persona_write_threshold, and temperament_write_threshold are governance parameters. They must not only appear in the manifest — they must actually be effective at runtime. rc7 had a point here that needed review on the temperament path; rc18.dev9 lists the temperament plumbing as a preserved or repaired function. Exactly such points belong in regression tests.

qdrant_enabled is a good example of a default that is deliberately conservative. Retrieval can be helpful, but it changes the complexity of the session. Off by default means: basic functionality must run without an index; activation is a deliberate decision.

persona_layer_disabled is an example of a powerful but narrowly bounded switch. It changes the compile output, not the security boundary. In configuration and documentation, the name must therefore stay precise. A switch that is misunderstood is almost as dangerous as a switch that is implemented incorrectly.

F.10 Qdrant Adapter and Index Lifecycle

The Qdrant adapter must not present itself as a second memory system. It is an index over sources, not the source itself. This design decision must be reflected in code, UI, status, and errors. A hit from Qdrant needs a source reference, freshness information, and the ability to be refused on staleness.

Duplicate-heading source-current consistency is a specific but important point. If multiple sources have similar headings or diary segments, retrieval must not silently point to a wrong or old source. Diary and reflection data in particular are chronological and source-dependent. Source confusion would produce false evidence.

Reindex must not happen automatically and uncontrolled. An automatic background reindex can degrade performance, deterministic session starts, and error diagnosis. A better approach is an explicit reindex with a structured status: started, busy, timeout, completed, or failed.

Drop index must not be understood as memory deletion. The index is derivable. Deleting the index does not remove any memory entries. This distinction matters for users, because otherwise panic or false deletion assumptions can arise.

Appendix G. Example Flows as Technical Sequences

The following scenarios describe typical flows including expected refusals. They are suitable as a basis for test cases, user documentation, and review conversations.

G.1 Session Start with Local Desktop Access

The user starts a new local MCP client session in the PLwC project. The model looks for PLwC tools, finds plwc-gateway, and calls plwc_profile(operation="compile"). The compile produces a compiled_layer. The model then calls plwc_workspace_operation(operation="list", path="."). The listing succeeds and shows workspace files. The session can report once that local PLwC access is available.

It matters that the model does not respond with a fabricated persona before the compile. The compiled_layer is the binding context. If the workspace-list call succeeds, that does not automatically mean Docker is ready;

plwc_status(scope="sandbox") is responsible for that. The environment classification should not be repeated in every response.

```
tool_search("plwc") plwc_profile(operation="compile") plwc_workspace_operation(operation="list", path=".")
```

G.2 Browser or Remote Context Without a Local MCPB

The model tries to find PLwC, but the tools are not visible. In this case it must not act as if it can read local files or profiles. It must briefly explain that PLwC is not available locally in this session. Normal theoretical help is possible; local tool work is not.

This case matters because users switch between local desktop clients, hosted web apps, and mobile clients. PLwC's current Dev9 functions depend on a local MCPB or stdio process. A hosted browser chat cannot automatically access the local workspace.

```
tool_search("plwc") -> no PLwC tools visible Response: local PLwC functions not available in this session
```

G.3 A Direct Write to PERSONA.md Is Refused

The model or the user attempts to change PERSONA.md via plwc_workspace_operation(operation="write"). The path either lies directly in the profile root or is recognized as a protected path. The operation must return DENY. The correct continuation is a reflection or governor flow, not a detour through a different path.

This test is one of the most important security smokes. If it fails, the separation between workspace payload data and governance control state is broken.

```
plwc_workspace_operation(operation="write", path="PERSONA.md", content="...", mode="rewrite") -> DENY
protected_governance_path
```

G.4 Profile Creation with Configured Active-Profile Precedence

The user creates a new profile via profile_creation. The plan validates the onboarding_answers and shows the planned files. Apply with confirmed=true creates the profile. If the MCP client, however, continues to set active_profile_name to a different profile, PLwC must report that the new profile was created but is not effectively active.

The correct user action is then to adjust the extension configuration or perform a clean profile activation under the applicable precedence rules. A success message without this notice would be misleading.

```
plwc_governor(operation="plan", plan_type="profile_creation", onboarding_answers={...})
plwc_governor(operation="apply", plan_type="profile_creation", confirmed=true, onboarding_answers={...}) ->
created, but effective activation blocked by configured_active_profile
```

G.5 Memory Promotion from Reflection

A recurring observation is first written with plwc_reflection. Later, plwc_governor produces a memory_promotion or reflection_memory_promotion plan. The plan contains a summary, evidence, trust, target, and known conflicts. Only after a confirmed apply is memory.md changed.

This flow prevents a one-off remark from immediately becoming durable context. It also allows conflict checking: if an old memory entry contradicts it, retirement or review_required should be triggered.

```
plwc_reflection(operation="write", summary="...", evidence="...", candidate_for="memory.md")
plwc_governor(operation="plan", plan_type="reflection_memory_promotion", ...) plwc_governor(operation="apply",
plan_type="reflection_memory_promotion", confirmed=true, ...)
```

G.6 Docker Is Missing on a Sandbox Call

The user asks for Python execution. plwc_sandbox_run checks Docker and finds that Docker is unavailable. The response must report Safe Mode and must not execute the code on the host. The result is a controlled refusal, not a broken tool call.

For user communication, it matters that profile and workspace functions can remain usable. Only sandbox and Document Worker are restricted. This avoids the false conclusion that PLwC is entirely broken.

```
plwc_sandbox_run(lang="python", code="print(1)") -> safe_mode_denied / docker_unavailable
```

G.7 Qdrant Stale Retrieval

The user wants task-related context via working compile or retrieve. The index, however, is older than the relevant sources. PLwC must not deliver outdated hits. Instead it reports index_stale or refused_stale and returns no hits. The user can trigger a reindex or continue without retrieval.

This behavior is strict but correct. Retrieval from outdated sources is treacherous, because the answer sounds plausible while still resting on superseded memory state.

```
plwc_profile(operation="compile", compile_mode="working", task_context="...") -> qdrant
index_stale/refused_stale, no hits
```

Appendix H. Extended Test Matrix

This test matrix is not complete in the sense of a formal test specification, but it shows what kinds of tests are necessary for a credible RC. What matters is the mix of positive capabilities, negative boundaries, and desktop/package reality.

Group	Test case	Stimulus	Expectation
Registry	Exactly eight tools	build_mcp_server/list_tools, or the installed desktop shows eight facade tools	PASS
Registry	No legacy public names	Search for plwc_compile_profile etc. in the public registry	PASS if not visible
Registry	Descriptions non-empty	All eight tools have metadata descriptions	PASS
Workspace	Root list	list path="."	File list, or a clear no-local-access
Workspace	Protected write	write PERSONA.md	DENY
Workspace	Traversal	read ../secret.txt	DENY
Workspace	Exact replace count	expected_replacements wrong	DENY/validation_error
Workspace	Batch limits	too many files/bytes	Bounded output or DENY
Governance	Plan, no mutation	memory_promotion plan	No file change
Governance	Apply without confirmed	apply confirmed=false	unconfirmed_apply_denied
Governance	Source changed	Change the source after the plan	source_changed_replan_required
Governance	Force limitation	force without confirmed	Still DENY
Governance	CORE promotion	plan target CORE.md	non_target / DENY
Sandbox	Docker missing	sandbox_run without Docker	Safe Mode DENY
Sandbox	Python smoke	print sentinel inside the container	exit_code 0
Sandbox	Network denial	Code attempts network access	Fails inside the container
Sandbox	Docker socket attempt	Mount/flag via model parameters	Parameter not accepted
Sandbox	Timeout	Infinite loop	Timeout/kill
Document Worker	DOCX create	create_docx V2 JSON	File created
Document Worker	DOCX edit overwrite	output_path exists	DENY
Document Worker	DOCM input	edit_docx .docm	DENY
Document Worker	ZIP Slip	extract_zip with a ../ entry	DENY
Document Worker	Image too large	read_image without resize, over the limit	DENY or notice
Qdrant	Disabled default	qdrant_enabled false	No automatic retrieval
Qdrant	Stale index	Memory changed, index old	refused_stale

Qdrant	Busy retry	Reindex running	qdrant_reindex_busy
Qdrant	Drop index	drop_index	Index gone, memory remains

Appendix I. Threat Model in Prose

A formal threat model for PLwC should consider not only classic technical attacks, but also model misbehavior, prompt injection embedded in files, misconfiguration, and false user expectations. The following prose summarizes the main threats along typical STRIDE categories, without turning this into a complete certification document.

Threat	Assessment and countermeasure
Spoofing	A model or a file could pretend to be an authorized governance source. PLwC reduces this through source_file, source_heading, source_sha256, and plan/apply checking. Even so, it remains important that external content is not automatically accepted as evidence. A README file in the workspace must not carry the same authority as a confirmed user decision.
Tampering	Manipulation of profile, memory, or governance files is the central risk. Protected paths, workspace separation, and governor flows are the countermeasures. If an attacker has direct file access to the profile root outside PLwC, PLwC cannot fully prevent that; at most it can report staleness, hash deviations, or inconsistent states.
Repudiation	Without an audit, no one could later determine whether a memory entry was planned, confirmed, or written directly. Metadata-only audit reduces this gap without copying raw data. For enterprise operation, it would additionally need to be clarified how audit logs are secured, rotated, and protected against manipulation.
Information Disclosure	Workspace reads, batch read, PDF extraction, and audit can disclose data. Size limits, path boundaries, protected reads, and audit redaction reduce this risk. Parallel active MCPs, however, remain an external disclosure path that PLwC cannot control.
Denial of Service	Infinite loops, large documents, huge ZIPs, many PDF pages, or reindex stalls can burden the session. PLwC mitigates this through Docker resource limits, document limits, structured timeouts, and busy status. It does not guarantee complete DoS safety against local host overload.
Elevation of Privilege	The most dangerous EoP paths would be host-shell fallback, Docker socket mount, privileged container, host network, or broad mounts. The architecture forbids these paths. Any later extension of the sandbox must be checked against this list, or the central security promise is weakened.

Appendix J. Data Integrity, Race Conditions, and Consistency

Data integrity in PLwC is not only a filesystem topic. There are several layers of consistency: the content of a working file, the state of a profile, the validity of a governor plan, the freshness of a Qdrant index, and the effective active-profile selection. Each layer can drift between two tool calls.

For normal workspace files, a content hash or expected_replacements can help. If a file was changed after it was read, an exact replace should not be applied blindly. For governor plans, the principle is stricter: the source from which evidence originates must remain stable between plan and apply. source_sha256 is therefore not a nice extra, but part of the mutation contract.

Qdrant staleness is another form of the same problem. The index can formally exist but semantically no longer match the memory state. A stale index is more dangerous than no index, because it delivers plausible hits. That is why refused_stale is the correct security decision.

Active-profile consistency is especially treacherous, because users frequently confuse visible profile files with the actual compile context. rc18.dev9 explicitly handles configured precedence. For testing this means: profile creation, profile activation, and compile must be checked together, not in isolation.

Consistency layer	Drift scenario	Required response
Workspace file	File changed between read and exact_replace.	Check hash/expected replacement; refuse on mismatch.
Governor plan	Diary source changed between plan and apply.	source_changed_replan_required.
Profile compile	Configured profile missing or diverges from active_state.	onboarding_required or a precedence notice.
Qdrant index	Memory changed, index old.	refused_stale, offer a reindex.
Audit	Audit cannot safely write a high-risk preflight.	High-risk operation fail-closed.

Appendix K. Documentation and Release Maturity

For a public or semi-public release, it is not enough for the MCPB to work locally. The documentation must match the artifact. This includes version, hash, file size, tool list, user config, known non-goals, Docker image requirements, and smoke verdicts. Documentation drift is a real risk, especially for a security gateway.

An Open Beta may have notes, but it must not be unclear. Dev9 is unsigned and states that openly. The verified package, Claude Desktop, and Odysseus smoke reports record PASS, while the local GPT stdio route is maintainer-confirmed. These results identify the tested baseline without turning the package into a final or production-certified release.

Release documents should distinguish between source status, package smoke, and client smoke. Source status states what is planned or implemented in the code. Package smoke states what is inside the MCPB. Client smoke states what Claude Desktop, local GPT-compatible stdio, or Odysseus actually makes visible and executable. These levels are related but not identical.

Release artifact	Minimum contents
README	Status, not-final notice, public tools, quickstart, capability boundaries.
CHANGELOG	Changes per RC, important fixes, known open items.
Manifest	Version, long_description, user_config, tools, runtime mapping, icon.
Release Notes	Hash, file size, smoke result, installation notes, known issues.
Smoke Report	Concrete test calls, result, environment, PASS/PASS_WITH_NOTES/FAIL.
Tester Guide	Installation path, Docker images, first run, troubleshooting.
Risk Register	Risks, status, countermeasures, open acceptance decisions.

Appendix L. Review Questions for External Third Parties

These questions help an external technical reviewer not just skim the architecture, but probe the right spots.

- Can a model reach any filesystem or shell access outside plwc-gateway?
- Are the old v0.1 tool names really not publicly registered?
- How is a workspace path canonically resolved and checked against symlink escape?
- Which files count as protected paths, and how is a write to them prevented?
- Which operations are mutation-bearing and require confirmed=true?
- Can force=true bypass a confirmation, semantic check, or source-hash check?
- What happens when Docker is missing: safe refusal or host fallback?
- Which Docker flags are actually set, and how is the model prevented from influencing them?
- Which raw data can end up in the audit, and which redaction applies?
- How is Qdrant prevented from delivering outdated memory hits?
- How is active-profile precedence explained between the extension configuration and PLwC state?
- Which document operations are explicitly not supported, and are they negative-tested?
- How is a profile change distinguished from a normal workspace file change?

- Is there a reproducible release with hash, package smoke, and desktop smoke?

Appendix M. Detailed Operation Contracts

This appendix describes the eight facade tools as operation contracts. An operation contract is more than a parameter list: it describes preconditions, flow, postconditions, typical denials, and review notes. This form is well suited for software description, test design, and later developer handover.

M.1 plwc_status Contract

plwc_status is read-only and must not change any persistent state. It is the first diagnostic anchor when a session is unclear. The tool must be designed so that a model cannot derive sensitive raw paths, secrets, or profile contents from the status, while still receiving enough information for the next sensible step.

The status call matters especially because it separates runtime problems: PLwC not visible, profile missing, Docker missing, Document Worker missing, active-profile mismatch, or browser context. Without this separation, the user would quickly get stuck on generic error messages.

Aspect	Contract
Precondition	PLwC tools visible; scope must be supported.
Flow	Normalize scope, gather runtime/sandbox/first-run/config status, redact sensitive values.
Postcondition	No mutation; structured status with next_actions.
Deny/Error	Unknown scope, internal status source unavailable, redacted output instead of raw data.
Review question	Could scope=config accidentally output secrets or full local paths?

M.2 plwc_describe Contract

plwc_describe is the current source of schema truth. It reduces hallucinations about tool names and parameters when the model is unsure. In a system with a historical v0.1 API, this matters especially, because old names are still functionally intuitive but are no longer correct publicly.

The tool should name not only positive operations, but also common denial reasons and capability boundaries. A good describe output prevents support cases: if the model reads that delete is not supported, it should not try three variants of delete.

Aspect	Contract
Precondition	scope and detail level are optional but must be normalized to valid values.
Flow	Derive current facade schemas and operations from runtime/constants.
Postcondition	Read-only output; no state change.
Deny/Error	An unknown scope should be clearly validated.
Review question	Does plwc_describe match the manifest and the actual registry?

M.3 plwc_profile Contract

plwc_profile connects the model to the active profile state. The most important operation is compile, because it produces the compiled_layer for the session. Status, snapshot, retrieve, scan_tagebuch, and doctor support diagnostics and context but must not open direct file bypasses.

Compile must be able to fail cleanly with setup_required or unavailable. A missing profile is not a reason to improvise a persona. If persona_layer_disabled is active, the compile output must respect that without disabling hard governance.

Aspect	Contract
Precondition	Profile root available; the desired profile exists, or first-run status explains the failure state.
Flow	Determine the profile source, evaluate active precedence, check compile mode, respect the persona layer if applicable.

Postcondition	compiled_layer, or a structured setup/error state.
Deny/Error	Unsupported compile_mode, doctor_mode, or doctor_scope; missing configured profile.
Review question	Can a model obtain profile content more rawly via snapshot/retrieve than intended?

M.4 plwc_reflection Contract

plwc_reflection is a controlled write path into the reflection layer. It is meant to capture observations that can later become evidence for memory, persona, or temperament promotion. It is deliberately not a direct memory or persona write.

The tool needs semantic quality gates. A reflection should contain reusable insight, not just a technical log entry. A cross-profile mismatch must be refused, because an observation for profile A must not land uncontrolled in profile B.

Aspect	Contract
Precondition	operation=write, active or explicit profile matches, summary/evidence is meaningful.
Flow	Check semantics, verify target/candidate_for consistency, format the entry, perform the governed write.
Postcondition	Reflection entry present, or duplicate/supporting evidence handled cleanly.
Deny/Error	cross_profile_denied, rejected_semantics, target mismatch, missing required fields.
Review question	Are raw technical logs and autonomous self-descriptions reliably rejected?

M.5 plwc_governor Contract

plwc_governor is the most critical contract. It is the only normal layer for durable profile, memory, persona, and temperament mutations. That makes its plan/apply model security-decisive.

Plan must not mutate. Apply must require confirmed=true and re-check the plan/source/target/semantic gates. Where a plan is based on a source, source integrity must be preserved between plan and apply. Force may only override narrowly defined thresholds.

Aspect	Contract
Precondition	Valid plan_type, target profile determined; for apply, confirmed=true and matching plan/source data.
Flow	Evaluate the plan, or check apply against a pending/approved plan; write target files under protection.
Postcondition	For plan, only a preview; for apply, a precisely defined mutation plus audit.
Deny/Error	unconfirmed_apply_denied, source_changed_replan_required, non_target, unsupported_plan_type.
Review question	Is there any apply path that bypasses confirmed or source checking?

M.6 plwc_sandbox_run Contract

plwc_sandbox_run allows bounded execution. The contract stands or falls with Docker-only. Execution must never land on the host, not even as debug or fallback behavior.

The sandbox call must bound container resources and ignore Docker parameters from the model. lang=node is additionally narrower: code denotes a workspace-relative .js path. This reduces inline JS and dynamic package installation.

Aspect	Contract
Precondition	Docker available, image present, workspace mount validated, lang supported.
Flow	PolicyIntent EXECUTE_SANDBOXED, audit preflight, start Docker with

	server-owned args, bound output.
Postcondition	Exit code, bounded stdout/stderr, or a structured Safe Mode deny.
Deny/Error	docker_unavailable, image_missing, unsupported_lang, timeout, mount_policy_denied.
Review question	Can a prompt force network access, a Docker socket, privileged mode, or additional mounts?

M.7 plwc_workspace_operation Contract

plwc_workspace_operation is the working-file contract. It must be useful enough for real project work while staying narrow enough not to become a general host file manager.

The most important precondition is a safe, workspace-relative path. Every operation must be checked against allowed roots and protected paths before execution. For write operations, it must additionally be clear whether rewrite, append, copy, or exact_replace is intended.

Aspect	Contract
Precondition	Operation supported, paths are workspace-relative, target is not protected, size limits observed.
Flow	Normalize the path, check the intent, audit, run the adapter, bound the result.
Postcondition	File read/written/moved, or a structured deny.
Deny/Error	path_traversal_denied, protected_path_denied, unsupported_operation, size_limit_exceeded.
Review question	Are symlink and broad-root cases tested in a real environment?

M.8 plwc_document_operation Contract

plwc_document_operation bundles parser and document work. The contract must explicitly describe input formats, output formats, and non-goals, because document functions are otherwise quickly over-interpreted.

The Document Worker must not be a free conversion engine. Every operation needs a controlled specification: create_docx/xlsx/pptx/pdf, edit_docx, inspect/extract, bounded PDF operations, bounded ZIP operations, and image read. Everything else must be validation_error or unsupported_operation.

Aspect	Contract
Precondition	input_path/output_path are workspace-relative, format supported, worker ready, limits observed.
Flow	Validate parameters, call the worker container/adapter, produce output or extraction, bound the result.
Postcondition	New artifact, or structured extraction; no in-place destruction.
Deny/Error	worker_missing, unsupported_format, overwrite_denied, macro_denied, zip_slip_denied.
Review question	Are OCR, redaction, signature, macro execution, and raw XML really refused?

Appendix N. Governance Examples as Decision Logic

The following examples show how governance decisions are meant to be read functionally. They illustrate that PLwC does not just protect files, but controls durable behavioral changes.

N.1 Harmless Working Request versus Persona Change

The user says: "Answer more concisely in this reply." That is a harmless working request for the current answer and needs no persona promotion. If the user says repeatedly and explicitly: "From now on always give a brief version first, then details," that can, after evidence and confirmation, become a persona or temperament candidate.

Danger: writing every temporary style request straight into PERSONA.md would quickly clutter the profile. PLwC needs the distinction between a session wish and a durable rule.

N.2 Memory Fact versus Project Fact

The user says: "PLwC rc18.dev9 has Qdrant optional, off by default." That is a project fact in the current document, not automatically a personal memory fact. The user says: "Remember for PLwC documents: always name risks openly." That can be a memory/persona candidate if it is durably relevant for future PLwC work.

Danger: mixing project content with personal working preferences makes memory imprecise. PLwC must check the purpose and the reuse value.

N.3 Reflection Candidate with Weak Evidence

A single observation can be noted as a reflection but be too weak for memory. The plan can report insufficient_evidence. Force might be administratively possible but should be used only deliberately and remain auditable.

Danger: force becomes a shortcut. The documentation must state that force does not replace confirmation and semantics.

N.4 Conflict with Existing Memory

A new candidate contradicts an active memory entry. The plan should not blindly append but should report the conflict. Possible resolution: retire the old entry, promote the new entry, or give both a validity context.

Danger: a contradictory compiled_layer leads to unstable behavior. Conflict management is product-relevant.

N.5 CORE.md as a Non-Goal

An observation feels fundamental and the user is enthusiastic. Even so, CORE.md is not an auto-promotion target. Changes to CORE are identity and security changes and do not belong in normal reflection/governor automation.

Danger: CORE promotion would be the fastest path to personality and governance erosion.

Appendix O. Extended Non-Goals by Domain

Non-goals are not negative in the sense of a lack of capability. They are part of the security design. The following domain-specific breakdown helps to correctly classify later feature requests.

Domain	Non-goals
Filesystem	No host file manager, no home-root exposure, no recursive delete, no bypass of protected paths, no access to profiles via normal workspace reads.
Process execution	No host-shell fallback, no process manager, no service starter, no Docker Compose orchestrator, no access to the Docker socket or host network.
Documents	No OCR, no legally sound redaction, no digital signature, no form engine, no macro execution, no free LibreOffice/Pandoc conversion within this contract.
Profile and memory	No autonomous self-rewriting, no CORE auto-promotion, no direct memory-file editing, no promotion without evidence/confirmation, no silent conflict resolution.
Qdrant	No canonical store, no automatic background reindex, no silent retrieval from a stale index, no memory truth from vector hits.
Audit	No raw-data archive, no SIEM replacement, no storage of secrets, code, stdout/stderr, or full profile contents.
Security in general	No guarantee against model hallucinations, no control over parallel MCPs, no enterprise sign-off, no malware sandbox with certification claims.

Appendix P. Acceptance Recommendation for the Next Version

This Dev9 edition incorporates the verified package facts and the recorded package, Claude Desktop, and Odysseus smoke conclusions. Future document versions can add selected sanitized test excerpts and additional clean-system installation evidence without changing the distinction between description and proof.

The most important improvement would be a traceability matrix from requirement ID to code path, test file, smoke report, and documentation section. This would make it possible, for example, to trace RC18-ONBOARDING-001 directly from the manifest description through first-run status to the smoke result. For external third parties, this traceability is often more convincing than long prose alone.

It would also make sense to add a separate operator appendix with concrete Windows paths, Docker build commands, and screenshots from Claude Desktop. This software description deliberately remains more technical and general. Beta testers additionally need a shorter, action-oriented installation guide.