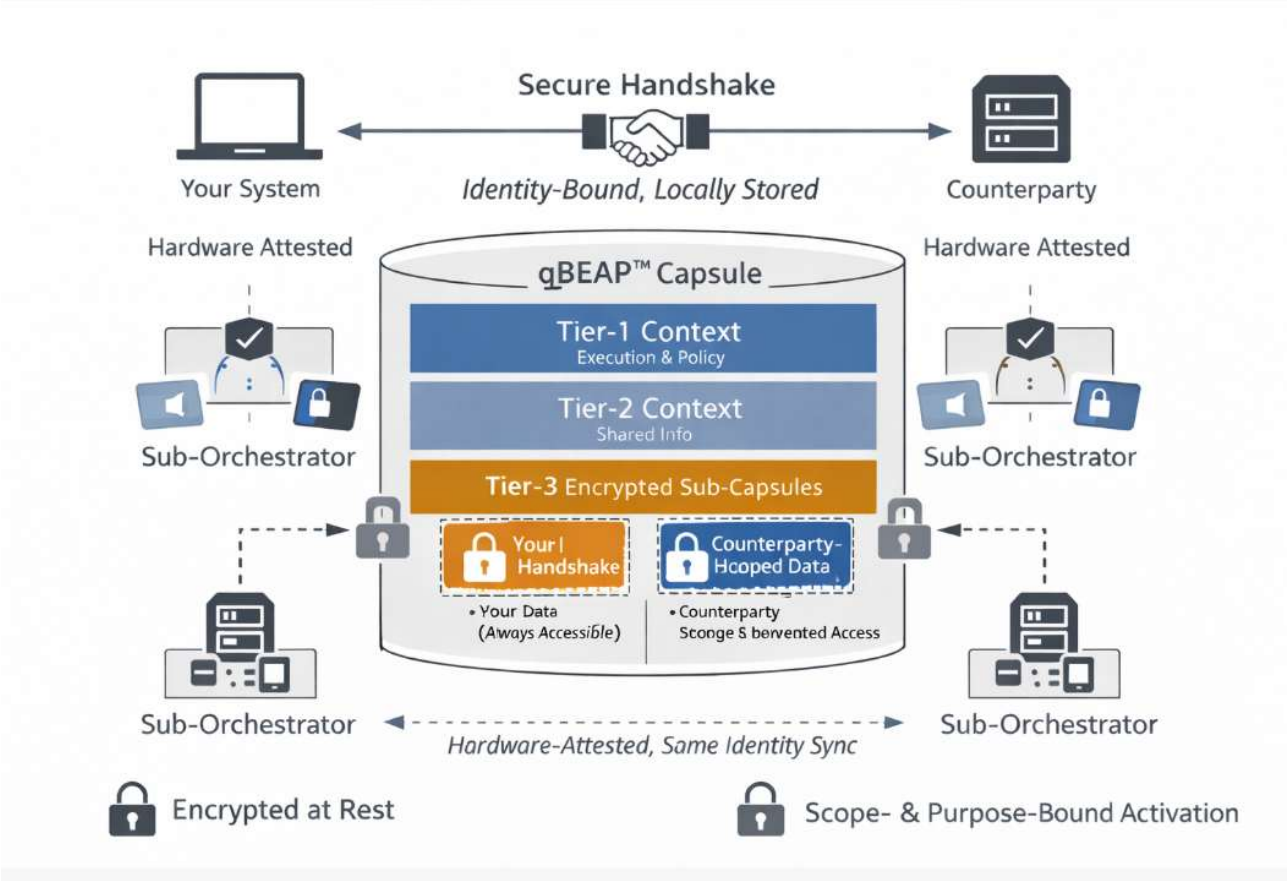

Annex A1 (Normative)

Expectation-Driven Handshake Synchronization, Attested Execution Environments, and Purpose-Bound Data Activation in BEAP™

A1.1 Status and Scope

This annex is normative.
It defines mandatory handshake semantics, storage rules, access boundaries, and synchronization constraints for BEAP™-compliant implementations that support expectation-driven handshakes and handshake-scoped data.

This annex does not modify or extend:

- wire formats,
- capsule serialization formats,
- cryptographic primitives,
- transport mechanisms, or
- PoAE™ execution semantics

defined in the BEAP™, qBEAP™, and PoAE™ specifications.

This annex defines behavioral, architectural, and security requirements that MUST be honored by conforming implementations.

A1.2 Applicability and Environment Constraints

The handshake model defined in this annex SHALL impose hardware-attestation requirements only on environments that request, activate, process, or receive sensitive effects under a handshake-scoped cooperation state.

An implementation MUST NOT enable processing, activation, or access to handshake-scoped sensitive data unless:

- the requesting or processing environment is hardware-attested, and
- the attestation is bound to a verifiable identity recognized by the BEAP™ trust model.

Hardware attestation:

- SHALL be mandatory for environments that request, activate, or process:
    - PII,
    - sensitive data,
    - financial or other irreversible effects.
- SHALL NOT be mandatory for environments that:
    - initiate first contact,
    - negotiate expectations,
    - express consent,
    - participate in handshake establishment without processing sensitive data.

Non-attested environments MAY participate in BEAP™ interactions and MAY be parties to a handshake but SHALL NOT request, activate, process, or receive handshake-scoped sensitive data.

A1.3 Handshake as an Operational Boundary

In BEAP™, a handshake SHALL represent a bounded, identity-anchored cooperation state shared between two parties.

A handshake state SHALL be:

- identity-bound,
- explicitly consented,
- cryptographically verifiable,
- revocable,
- and confined to attestation-enforced execution boundaries for sensitive operations.

The handshake SHALL function as a temporary operational contract governing:

- which data classes MAY exist within exchanged capsules,
- which expectations apply,

- which encrypted data regions MAY be accessed,
- and under which declared purposes activation MAY occur.

All shared context SHALL be strictly scoped to the lifetime and declared scope of the handshake.

A1.4 Local Handshake Persistence

Each participating party SHALL store the handshake state locally.

Local handshake persistence:

- SHALL be encrypted at rest,
- SHALL be cryptographically bound to the handshake identifier,
- SHALL be bound to the local execution environment and identity,
- SHALL NOT constitute a global session state,
- SHALL NOT require continuous online availability of the counterparty.

For handshake-scoped sensitive data, local persistence and any subsequent activation SHALL require a hardware-attested execution environment.

Local persistence SHALL NOT relax capsule-local determinism or PoAE™ policy enforcement.

A1.5 Expectation-Driven Synchronization

A1.5.1 Expectations as Declarative Signals

An expectation SHALL be a declarative statement describing information that MAY be required for a handshake to transition into an active operational state.

An expectation:

- SHALL NOT mandate immediate disclosure,
- SHALL NOT imply processing,
- SHALL define requirements, not actions.

An expectation MAY specify:

- attribute identifiers,
- data class (e.g. PII, business-critical, informational),
- purpose identifiers,
- requirement level,
- additional constraints (e.g. jurisdiction, format, validation state).

A1.5.2 Iterative Capsule Synchronization

Handshake completion SHALL occur through iterative capsule exchange.

Capsules exchanged during synchronization MAY contain:

- capsule-local deterministic execution context,
- shared informational references,
- encrypted handshake-scoped data regions.

Sensitive data SHALL NOT be required to be processed solely to satisfy expectation negotiation.

A1.6 qBEAP™ Capsule Structure and Encrypted Regions

A qBEAP™ capsule used under an active handshake MAY contain multiple encrypted regions, each with distinct access semantics.

A1.6.1 Capsule Regions

A capsule MAY conceptually contain:

Region 1 — Tier-1 Context
Capsule-local deterministic execution context and policy references.
This region SHALL be the sole authority for execution and automation decisions.

Region 2 — Tier-2 Context
Shared informational and non-sensitive material, including augmented overlay annotations and support context.

Region 3 — Handshake-Scoped Encrypted Sub-Capsules
One or more encrypted sub-capsules containing handshake-scoped data such as PII, sensitive data, or augmented overlay support for sensitive operations.

A1.6.2 Handshake-Scoped Encrypted Sub-Capsules

Handshake-scoped data SHALL be stored exclusively within encrypted qBEAP™ sub-capsules.

Each such sub-capsule:

- SHALL be encrypted at rest and in transit,
- SHALL be cryptographically bound to the handshake identifier,
- SHALL be isolated from execution context,
- SHALL NOT influence execution logic,
- SHALL be inaccessible without explicit activation.

Multiple encrypted sub-capsules MAY coexist within a single qBEAP™ capsule.

A1.7 Access Semantics and Purpose-Bound Activation

Processing or access to handshake-scoped sensitive data SHALL be permitted only when all of the following conditions are met:

- a matching expectation exists,
- a declared purpose identifier applies,
- policy conditions are satisfied,

- valid consent is present,
- the requesting or processing environment is hardware-attested.

Requesting or activating access to PII, sensitive data, or irreversible effects without hardware attestation SHALL NOT be permitted, regardless of consent or handshake existence.

Each party:

- SHALL be able to decrypt and access its own contributed handshake-scoped data at any time,
- SHALL NOT gain unrestricted access to counterparty-contributed data.

Access to counterparty-provided handshake data:

- SHALL be scope-bound,
- SHALL be purpose-bound,
- SHALL be denied by default.

Transport or storage of encrypted handshake-scoped data SHALL NOT constitute processing.

A1.8 Sub-Orchestrator Synchronization

Handshake-scoped data MAY be synchronized between sub-orchestrators operating under the same identity, provided that:

- each sub-orchestrator that may activate or process sensitive data executes in a hardware-attested environment,
- attestation confirms equivalence of trust level,
- the identity binding is verifiable and unchanged.

Synchronization MAY include:

- an empty qBEAP™ capsule,
- the handshake identifier,
- encrypted handshake-scoped sub-capsules.

Sub-orchestrators that are not hardware-attested MAY receive handshake metadata but SHALL NOT receive, activate, or process handshake-scoped sensitive data.

Synchronization SHALL NOT relax access controls or activation rules defined in this annex.

A1.9 Permitted Data Introduction Paths

BEAP™ SHALL permit two normative mechanisms for introducing sensitive data into a handshake-scoped cooperation state:

1. Expectation fulfillment via explicit data request
2. Handshake-anchored encrypted data presence

In both cases, data SHALL remain inactive until purpose-bound activation occurs.

A1.10 Security, Privacy, and Determinism Properties

The model defined in this annex:

- restricts processing to explicitly declared purposes,
- prevents implicit or ambient data access,
- preserves auditability and traceability,
- enforces role- and operation-bound attestation requirements,
- maintains strict determinism.

Handshake revocation SHALL immediately affect future access and synchronization without altering historical integrity.

A1.12 First Contact and Handshake Eligibility

A BEAP™ interaction MAY begin without an existing handshake.

First contact MAY occur via:

- public WR Codes,
- websites or applications,
- BEAP™ Packages containing pBEAP™ capsules,
- other BEAP™-compatible interaction mechanisms.

The absence of a handshake SHALL NOT prevent:

- expectation declaration,
- message exchange,
- automation execution,
- data requests,
- consent negotiation,
- or handshake establishment.

qBEAP™ capsules SHALL NOT be used for first contact. qBEAP™ capsules SHALL be permitted only for enterprise handshakes in which both parties execute in hardware-attested environments.

A1.13 Handshake Creation and Promotion

A handshake MAY be created as a result of an interaction that includes:

- a declared expectation,
- an explicit data request,

- and valid user consent.

When these conditions are met, an implementation MAY promote the interaction into a handshake-scoped cooperation state.

Handshake creation:

- SHALL require explicit, informed consent,
- SHALL be identity-bound,
- SHALL be cryptographically verifiable,
- SHALL be locally persisted,
- SHALL be revocable at any time.

No prior authentication or handshake SHALL be required for handshake creation itself.

Handshake promotion SHALL NOT:

- imply immediate processing of sensitive data,
- bypass purpose or policy constraints,
- activate sensitive data without satisfying the requirements defined in this annex.

A1.14 Handshake Scope Anchoring and Reuse

Each handshake SHALL be anchored to a declared cooperation scope.

A handshake scope SHALL define:

- the cooperating counterparty identity,
- the permitted interaction domain or service context,
- the allowed purpose identifiers,
- the permitted data classes.

A handshake SHALL NOT be implicitly bound to a specific WR Code.

Multiple interaction entry points, including multiple public WR Codes and BEAP™ Packages, MAY reference the same handshake, provided their declared scope requirements are compatible with the handshake scope.

Scope compatibility SHALL be evaluated deterministically at execution time.

A1.15 Capsule Formats, Automation, and Attestation Requirements

BEAP™ defines multiple capsule formats with distinct security and deployment properties.

A pBEAP™ capsule:

- SHALL be unencrypted,
- MAY carry messages, expectations, and automation instructions,

- SHALL be used for first contact, public WR Code interactions, and general messaging,
- SHALL NOT require hardware attestation on both parties.

For public publishers of WR Codes, the publishing environment:

- SHALL be hardware-attested,
- SHALL be identity-bound within the BEAP™ trust model,
- SHALL NOT require the scanning or consuming party to be hardware-attested at first contact; however, additional use cases MAY require hardware attestation of the scanning or consuming party after handshake establishment.

A qBEAP™ capsule:

- SHALL be encrypted,
- MAY carry messages and automation instructions,
- SHALL be used exclusively for enterprise handshakes,
- SHALL require both parties to execute in hardware-attested environments.

Automation capability is not limited to qBEAP™ and MAY be executed within pBEAP™ capsules, subject to policy and trust constraints.

A1.16 Encrypted Handshake-Scoped Sensitive Data Containers

In addition to capsule formats, an encrypted handshake-scoped sensitive data container MAY be used as an edge case for sensitive data handling.

Such encrypted handshake context:

- SHALL contain sensitive data only,
- SHALL NOT contain automation templates or execution logic,
- SHALL NOT constitute a qBEAP™ capsule,
- MAY be embedded within a BEAP™ Package containing a pBEAP™ capsule,
- SHALL require hardware attestation only for the requesting or processing environment,
- SHALL be encrypted using the same cryptographic construction, key management model, and post-quantum–ready primitives as defined for qBEAP™ capsules.

The existence of such encrypted containers SHALL NOT alter the classification of the enclosing capsule format.

A1.17 Sensitive Data Introduction During Handshake Establishment

Sensitive data MAY be introduced during or after handshake establishment only when:

- a matching expectation exists,
- a declared purpose identifier applies,

- explicit consent is granted,
- the requesting or processing environment satisfies the hardware-attestation requirements defined in this annex.

Sensitive data introduced under a handshake:

- SHALL be stored exclusively within encrypted handshake-scoped containers,
- SHALL remain inactive until purpose-bound activation occurs,
- SHALL NOT influence execution logic.

Requesting or activating access to PII or sensitive data without hardware attestation SHALL NOT be permitted, regardless of consent or handshake existence.

A1.18 Public WR Codes and Handshake Interaction

Public WR Codes:

- SHALL remain static and unencrypted,
- SHALL resolve to BEAP™ Packages containing pBEAP™ capsules,
- SHALL NOT contain sensitive data,
- SHALL NOT embed handshake identifiers or user-specific state.

Personalized behavior resulting from scanning a public WR Code:

- SHALL derive exclusively from an existing handshake,
- SHALL be enforced through scope and purpose evaluation,
- SHALL NOT alter the semantic meaning of the WR Code.

A1.19 Handshake Revocation Effects

A handshake MAY be revoked at any time by either party.

Upon revocation:

- future access to encrypted handshake-scoped containers SHALL be denied,
- synchronization of handshake-scoped sensitive data SHALL cease,
- activation of encrypted handshake-scoped data SHALL be prevented.

Revocation SHALL NOT alter historical BEAP™ Package or capsule integrity or auditability.

A1.20 Handshake Establishment via Public WR Code®

Public WR Code® instances SHALL function as static, unencrypted, replay-safe entry points into BEAP™ interactions.

They SHALL NOT embed handshake state, encrypted material, user-specific information, dynamic identifiers, or mutable execution logic.

Public WR Code® interactions SHALL require network access for:

- WR Code® resolution,
- retrieval of referenced templates and BEAP™ Packages,
- integrity verification of retrieved artefacts,
- verification of externally anchored integrity commitments as defined by the WR Code® protocol.

Resolution of a public WR Code® SHALL always result in a pBEAP™-based interaction, independent of whether a handshake exists.

A1.20.1 Resolution and Initial Execution Context

Upon resolution of a public WR Code®, the orchestrator SHALL:

- resolve the publisher identity associated with the WR Code®,
- retrieve the referenced BEAP™ Package containing a pBEAP™ capsule,
- perform integrity verification of retrieved artefacts against externally anchored commitments,
- establish a deterministic execution context.

At this stage:

- no handshake SHALL be assumed to exist,
- no encrypted handshake-scoped context SHALL be active,
- no authentication or hardware attestation SHALL be implied for the consuming party.

The publishing environment of public WR Code® instances SHALL be hardware-attested and identity-bound within the BEAP™ trust model.

A1.20.2 Handshake Evaluation and Verifiable Continuity

Once the required artefacts and verification material are available, the orchestrator SHALL evaluate whether a locally stored handshake exists that:

- is identity-bound to the resolved publisher,
- declares a cooperation scope compatible with the current interaction.

This evaluation:

- SHALL be performed within the orchestrator,
- SHALL be deterministic with respect to the retrieved artefacts and local handshake state,

- MAY involve verification of externally anchored commitments to ensure that the handshake state and scope have not been altered retroactively.

Handshake evaluation and establishment MAY rely on external verification mechanisms but SHALL NOT delegate execution authority, policy decisions, or access control to external systems.

If no compatible handshake exists, execution SHALL proceed in generic, non-personalized mode.

A1.20.3 Expectation Declaration and Handshake Eligibility

A pBEAP™ capsule resolved from a public WR Code® MAY declare expectations indicating:

- intended cooperation scope,
- data classes that MAY be requested,
- purposes that would apply if cooperation were elevated.

Such expectations:

- SHALL remain declarative,
- SHALL NOT imply processing,
- SHALL NOT activate sensitive data,
- SHALL NOT require hardware attestation.

Expectations MAY designate the interaction as handshake-eligible but SHALL NOT, by themselves, establish a handshake.

A1.20.4 Handshake Establishment as a Verifiable State Transition

A handshake SHALL be established only when:

- explicit user consent is provided,
- the cooperating publisher identity is known,
- the declared scope and purposes are accepted.

Handshake establishment:

- SHALL be implemented as a local, deterministic state transition within the orchestrator,
- SHALL create an identity-bound and scope-bound handshake record,
- SHALL NOT require modification of the WR Code®,
- SHALL NOT require modification of the capsule format,
- SHALL NOT require mutual hardware attestation.

To prevent repudiation, silent alteration, or post-hoc denial, the establishment of a handshake and its declared scope SHALL be cryptographically committed and externally anchored in a privacy-preserving manner.

Such anchoring:

- SHALL record only cryptographic commitments (e.g. salted hashes or aggregated roots),
- SHALL NOT expose identities, scopes, purposes, or user-specific metadata,
- SHALL serve as non-authoritative proof of existence and integrity.

The handshake SHALL become the continuity anchor for subsequent interactions with the same publisher and compatible scope.

A1.20.5 Post-Handshake Capability Escalation

After handshake establishment, additional capabilities MAY become available, including:

- personalization of subsequent pBEAP™ interactions,
- storage and retrieval of encrypted handshake-scoped sensitive data containers,
- escalation of hardware attestation requirements for specific purposes or operations.

Such escalation:

- SHALL be strictly scope- and purpose-bound,
- SHALL require hardware attestation only for environments requesting or processing sensitive or irreversible actions,
- SHALL be verifiably attributable to the established handshake,
- SHALL NOT retroactively affect prior interactions.

A1.20.6 Integrity Anchoring and Non-Repudiation Guarantees

Externally anchored integrity commitments SHALL be used to ensure that:

- publishers cannot deny or retroactively alter offered templates, scopes, or handshake terms,
- consumers can prove the origin, scope, and integrity of interactions they accepted,
- service providers and integrators can demonstrate non-manipulation of execution logic.

Such anchoring mechanisms:

- SHALL be used solely for verification and auditability,
- SHALL NOT become authoritative for execution, authorization, or policy enforcement,
- SHALL preserve unlinkability between independent interactions,
- SHALL prevent both unilateral repudiation and fabricated claims.

This anchoring model is intended to eliminate ambiguity, abuse, and post-hoc denial across all parties, thereby preventing unbounded or adversarial behavior in public WR Code®–based interactions.

A1.20.7 Purpose Binding, Consent Continuity, and Responsibility Allocation (Normative)

Any access to handshake-scoped data **SHALL** be bound to an explicitly declared purpose provided by the data-processing party.

Consent for such purposes:

- **SHALL** be obtained explicitly at handshake establishment or handshake scope extension,
- **MAY** authorize repeated access to the declared data classes for the declared purpose,
- **SHALL** remain valid until explicitly revoked,
- **SHALL NOT** require repeated user interaction for each subsequent use within the same scope and purpose.

Each party that processes data under a handshake **SHALL** attach verifiable references to its applicable **privacy policy** and, where applicable, **terms of service**, as part of the handshake state.

Such policy and terms references:

- **SHALL** be provided and maintained by the data-processing party,
- **SHALL** be bound to the handshake identity, scope, and purpose,
- **SHALL** be immutable for the lifetime of the handshake,
- **SHALL** define responsibility, conditions, and limitations for lawful data processing and service execution.

WR Desk™:

- **SHALL** act solely as a technical enabler and orchestration platform,
- **SHALL NOT** act as a data controller, joint controller, or data processor,
- **SHALL NOT** define, validate, or interpret declared purposes, privacy policies, or terms of service,
- **SHALL** enforce only the existence, binding, and immutability of declared purposes and policy references within the handshake.

Responsibility for purpose limitation, data minimization, lawful processing, contractual obligations, and compliance with applicable data protection and consumer protection regulations remains exclusively with the operating entities participating in the handshake.

A1.11 Conclusion (Final, Corrected)

This annex defines a deliberate architectural shift for **BEAP™ communications and automation** away from implicit trust, transient sessions, and unverifiable exchanges toward a **handshake-anchored cooperation model** that is explicit, auditable, and resistant to repudiation by any party.

The principles defined herein apply uniformly across **all BEAP™ interaction mechanisms**, including but not limited to public **WR Code®**–initiated interactions, application-based exchanges, service-to-service communication, and automated workflows.
WR Code® is treated as a prominent and security-critical entry mechanism, but not as a limitation of the handshake or context model itself.

The mechanisms specified in this annex intentionally separate:

- **interaction initiation** (which may occur through multiple BEAP™-compatible channels),
- **cooperation continuity** (established through identity- and scope-bound handshakes),
- **capability escalation** (controlled by purpose-, role-, and attestation-bound activation),
- and **accountability** (provided through cryptographic integrity anchoring).

By anchoring context, expectations, and sensitive cooperation state to explicit handshakes rather than to transport artefacts, session constructs, or user interface entry points, the model enables persistent, verifiable cooperation without embedding mutable or user-specific state into messages, capsules, or automation logic.

Beyond its security properties, this model is designed to **reduce friction in everyday operational routines**.
Data required to satisfy handshake expectations is maintained locally within the **WRVault™** and does not need to be re-entered or reconfigured for each handshake.
Handshake establishment can therefore draw from an existing, locally controlled data base, while access to that data remains strictly scope- and purpose-bound.

Knowledge bases, references, and previously consented data can be queried efficiently once a handshake exists, without repeated disclosure or negotiation.
Authentication and authorization can occur quickly and deterministically, as all required information is already locally available, yet remains inaccessible outside the declared scope and purpose.

A central design goal of this annex is to eliminate ambiguity and post-hoc denial in distributed and cross-organizational environments.
Both parties to a BEAP™ interaction are protected against false claims, silent modification, and retroactive reinterpretation of scope, consent, or permitted actions, while integrators and platform operators retain a defensible, non-authoritative audit trail.

The use of externally anchored cryptographic commitments is intentional and normative.
Such anchoring provides **proof of existence and integrity**, not authorization or control, and serves as a neutral witness for handshake-scoped cooperation states that would otherwise be unverifiable in adversarial, regulated, or high-risk contexts.

This annex defines the **target architecture** of the WR Desk™ and BEAP™ ecosystem.
It is expected that early implementations of WR Desk™ may not yet implement all primitives, enforcement points, or anchoring mechanisms described herein.
Such partial implementations do not weaken or reinterpret this specification; rather, they reflect a staged evolution toward the fully verifiable, non-repudiable cooperation model defined by this annex.

Conforming implementations are expected to evolve toward full compliance over time, without diluting the guarantees, boundaries, or intent established herein.

Annex B — Trust Domains, Provenance Context, and Sandbox-Scoped Execution

B.1 Canonical (Normative)

B.1.1 Trust Domains

The system SHALL support multiple **trust domains** defining execution boundaries for BEAP capsules.

Trust domains SHALL differ in permitted interaction surfaces and policy strictness, but SHALL NOT alter BEAP integrity, confidentiality, or correctness guarantees.

At minimum, the system SHALL distinguish:

- a primary (host) trust domain, and
- one or more lower-trust sandbox domains.

The primary trust domain SHALL be authoritative.

B.1.2 BEAP Security Invariance

All BEAP capsules, regardless of origin, provenance, or trust domain placement:

- SHALL remain valid BEAP capsules,
- SHALL retain full BEAP security guarantees,
- SHALL NOT be downgraded in security due to origin or transformation history.

BEAP security properties are invariant across trust domains.

B.1.3 Provenance Context

Every BEAP capsule MUST include provenance metadata describing the **context of its creation**.

Provenance metadata SHALL:

- identify the origin mechanism by which the capsule was produced,
- identify the origin channel from which the payload was derived,
- identify the **handshake trust class** under which the capsule was created.

The handshake trust class SHALL be one of at least the following:

- enterprise handshake,
- professional (pro) handshake,

- standard handshake,
- none / depackaged.

Provenance metadata:

- is informational but **normative for trust classification**,
- provides recipient context,
- SHALL NOT modify BEAP validity or cryptographic security guarantees,
- SHALL be used to determine trust-domain policies and execution constraints.

B.1.4 Depackaged BEAP Capsules

A **Depackaged BEAP Capsule** is a BEAP capsule created by transforming an external payload into BEAP format through a depackaging process.

A depackaged BEAP capsule:

- SHALL remain a fully valid BEAP capsule,
- SHALL be subject to all BEAP guarantees,
- SHALL differ from other BEAP capsules only by provenance metadata.

Depackaging SHALL be understood as a format and context transformation, not a trust reduction.

B.1.5 Execution Scope and Sandbox Domains

Execution scope SHALL be explicitly defined and SHALL be independent of provenance and handshake-derived trust levels.

A BEAP capsule MAY be assigned a **sandbox execution scope**.

The sandbox execution scope:

- is an **optional flag**,
- MAY be applied at BEAP capsule creation time or during inbox handling,
- SHALL NOT be implied by provenance, handshake class, or trust level.

The sandbox execution scope is a **routing flag** indicating that a **logical clone of the BEAP capsule** SHALL be synchronized to a sandbox trust domain.

The sandbox execution scope:

- SHALL NOT alter execution semantics in the primary trust domain,
- SHALL NOT imply mandatory sandbox execution of the BEAP capsule itself, except where explicitly restricted or overridden by policy,
- SHALL solely control clone-based synchronization behavior.

Interaction with **original external artefacts or external links**, even when referenced by a BEAP capsule, SHALL always occur in a sandbox trust domain, independent of whether the BEAP capsule carries a sandbox execution scope.

B.1.6 Clone-Based Sandbox Synchronization

Sandbox synchronization SHALL operate exclusively on BEAP capsules explicitly marked for sandbox execution.

Synchronization MUST occur only when:

1. the BEAP capsule carries a sandbox execution scope, and
2. the capsule exists in both the primary and sandbox trust domains.

Synchronization SHALL result in a **logical clone** within the sandbox domain.

The primary-domain capsule:

- SHALL remain authoritative,
- SHALL NOT be modified by sandbox execution,
- SHALL NOT receive implicit state or artifact propagation.

B.1.7 Authority and Promotion Rules

Artifacts, state, or outputs produced in sandbox trust domains:

- SHALL NOT be authoritative,
- SHALL NOT implicitly propagate to higher-trust domains.

There SHALL be no backchannel from the sandbox domain to higher-trust domains. The original BEAP capsule SHALL always remain on the primary (host) trust domain. Any explicit reuse of sandbox-derived results MUST occur via creation of a new BEAP capsule.

B.1.8 Trust Levels

The system SHALL enforce **handshake-derived trust levels** based on provenance metadata.

At minimum, the following trust levels SHALL be supported:

- enterprise handshake (highest trust),
- professional (pro) handshake,
- standard handshake,
- depackaged / no-handshake (lowest trust).

Trust levels SHALL:

- be derived deterministically from handshake provenance,
- influence execution policy, sandbox requirements, and promotion rules,

- be immutable for a given BEAP capsule.

Enterprise trust levels SHALL enforce the strictest execution and promotion policies and MAY prohibit sandbox interaction entirely.

B.2 Non-Canonical (Informative)

B.2.1 Immediate Trust Differentiation

Implementations typically expose handshake-derived trust levels in a manner that allows recipients to **immediately distinguish** between:

- enterprise handshake capsules,
- professional handshake capsules,
- standard handshake capsules,
- depackaged capsules.

The concrete representation (e.g. visual indicators) is implementation-specific but SHALL preserve unambiguous differentiation.

B.2.2 Purpose of Sandbox Domains

Sandbox trust domains enable controlled interaction with external systems and artefacts while preserving the primary domain as the system of record.

They also enable exploratory or experimental automation without authority escalation.

B.2.3 Physical Separation Requirement

Sandbox orchestrators SHALL be **physically separated** from the host system.

Physical separation means that the sandbox orchestrator operates on distinct hardware and does not share a common execution environment with the host.

Virtualization or containerization alone SHALL NOT be considered sufficient isolation for a sandbox orchestrator.

Implementation choices MAY vary, but physical separation is a mandatory property and does not affect the canonical semantics defined above.

B.2.4 Rationale

By separating:

- BEAP security guarantees,
- handshake-derived trust classification,
- provenance context,

- execution scope,

The architecture enables strict enterprise controls while retaining flexibility for lower-trust interaction and automation.

B.2.5 Sandbox-Prepared Archival Media (Informative)

Sandbox environments MAY support the use of **sandbox-prepared removable storage** for archiving original artefacts outside of BEAP.

Such removable storage:

- SHALL be prepared and initialized exclusively by a sandbox orchestrator,
- SHALL be cryptographically or structurally bound to that sandbox orchestrator,
- SHALL NOT be usable by the primary (host) trust domain or other systems.

Sandbox-prepared archival media enables:

- long-term archiving of original artefacts outside BEAP,
- operation of RAM-only or ephemeral sandbox environments,
- optional persistence of session state and artefacts when explicitly configured.

Use of sandbox-prepared archival media is optional and SHALL require explicit setup and user intent.

B.2.6 Ingestor and Validator Separation (Normative)

The system SHALL implement a mandatory, non-bypassable ingestion pipeline consisting of at least two distinct stages:

1. **Ingestor** (acquisition and depackaging), and
2. **Validator** (BEAP protocol conformance and integrity validation).

The ingestor and validator SHALL be logically separated components with an explicit interface.

B.2.6.1 Ingestor

The **Ingestor** SHALL act as the exclusive entry point for all external inputs and SHALL operate under the **identity of the host**.

All external inputs, regardless of channel or source, SHALL be processed exclusively by the ingestor prior to becoming BEAP capsules.

The ingestor SHALL:

- ingest messages and data from heterogeneous channels,
- depackage external inputs into **candidate BEAP capsules**,
- attach or preserve provenance metadata.

The ingestor SHALL NOT:

- validate BEAP protocol conformance beyond minimal framing required to produce a candidate capsule,
- render content,
- execute payloads,
- interact with original artefacts beyond depackaging,
- expose user-facing interfaces,
- distribute capsules to any trust domain.

B.2.6.2 Validator

The **Validator** SHALL be the sole authority for accepting candidate capsules as BEAP-conformant.

The validator SHALL:

- perform strict BEAP protocol conformance validation,
- validate structural integrity and required metadata presence,
- fail closed on any validation error,
- mark capsules as **validated** only upon successful verification.

A capsule that has not been marked validated SHALL NOT be treated as BEAP by any other component and SHALL NOT be distributed.

B.2.6.3 Distribution Gate

Distribution to any trust domain SHALL occur only for capsules marked validated by the validator.

The interface between ingestor and validator SHALL be non-bypassable. No component other than the validator SHALL be able to mark a capsule as validated.

B.2.6.4 Isolation Requirements

The ingestor and validator SHALL run in **strict isolation**.

Isolation SHALL be enforced by executing the ingestion pipeline:

- either on the host in a strictly isolated, non-interactive environment, or
- on a physically detached container or virtual machine with root control.

The ingestion environment MUST be configured for RAM-only operation, and all volatile memory used during depackaging or validation SHALL be securely cleared immediately after each depackaging process completes.

Isolation of the ingestion pipeline is mandatory and SHALL be treated as a non-bypassable security boundary. The specific isolation mechanism is implementation-dependent but MUST provide strong separation from both the primary (host) and sandbox orchestrators.

Annex X – Deterministic Client Execution, Extension Governance, and Off-Band Routing (Informative)

In **High-Assurance Mode**, WRDesk operates the primary orchestrator under strictly deterministic conditions. All executable components, including browser extensions and client-side execution units, MUST be **cryptographically WRStamped, version-pinned, and identity-bound**. Dynamic runtime code, uncontrolled auto-updates, and non-verifiable execution paths are excluded from the authoritative execution domain.

To support this model, WRDesk plans a **WRStamped extension distribution framework**, enabling deterministic deployment, controlled updates, revocation, and lifecycle management of verified execution units in high-assurance environments.

The orchestrator enforces a **single authoritative browser context** configured exclusively with WRStamped extensions and a static execution profile. This context represents the **trusted execution cage**. Any content, action, or workflow that cannot be conclusively verified, is classified as risk-bearing, or does not meet deterministic assurance requirements is **automatically routed off-band**.

Original artefacts extracted from emails (including attachments, embedded content, or referenced payloads) are automatically redirected to the off-band execution path. If configured, the orchestrator MAY automatically trigger a **sandbox computation wake-up** to process or inspect these artefacts in an isolated environment. If no automatic off-band switch to the sanbox environment is provisioned, an **explicit manual context switch via hotkey** is the recommended and enforced interaction pattern.

Off-band execution targets MAY include isolated RAM-only systems, mini-PCs, tablets, or equivalent sandboxed devices. **Explicit separation between observation and effect is enforced via dedicated hotkeys or programmable KVM switches. Special-purpose KVM devices with RS232 interfaces MAY be used for deterministic, scriptable routing and are supported for direct control and integration within the orchestrator.**

Extensions and tools executed off-band MAY persist depending on the operating system and deployment profile. RAM-only configurations are RECOMMENDED. Regardless of persistence, the system guarantees a **return to a known-good baseline**, either manually or through automatically triggered reset intervals, preventing long-term state drift and accumulation of untrusted changes.

This architecture preserves unrestricted exploration while ensuring that all authoritative execution remains deterministic, auditable, revocable, and provably governed.

Normative Section: Agent-Scoped Declarative Capability Enforcement

C1. Agent-Centric Execution Model

C1.1

All execution within the system MUST be performed by explicitly defined agents.

C1.2

Tools MUST be bound to agents and MUST NOT be executed independently of an agent context.

C1.3

Each agent MUST operate exclusively within the authority defined by its assigned Capability Tokens.

C2. Declarative Tool Capability Requirements

C2.1

Every tool MUST declaratively specify the Capability Classes required for its invocation.

C2.2

Declared tool requirements MUST be machine-readable and MUST be evaluated prior to execution.

C2.3

A tool MUST NOT be executable unless the invoking agent possesses Capability Tokens that fully satisfy the tool’s declared requirements.

C2.4

Users MUST NOT be required to manually assign capabilities when constructing agents. Capability assignment MUST be automatically derived from the tools bound to the agent.

C3. Capability Tokens

C3.1

Capability Tokens MUST be:

- bound to a single agent identity,
- bound to a specific execution context or session,
- non-transferable and non-delegable.

C3.2

Capability Tokens MUST NOT be shared, inherited, or reused across agents.

C3.3

Capability Tokens MUST be cryptographically verifiable and validated in the BEAP validator pipeline prior to execution.

C3.4

The absence, invalidity, or revocation of a required Capability Token MUST result in execution denial.

C4. Credential and Access Mediation

C4.1

Credentials MUST NOT be transferred directly to agents.

C4.2

All credential handling MUST be performed by the system through controlled access mechanisms.

C4.3

Agents MUST receive only declarative access permissions, not raw secrets, keys, or credentials.

C4.4

Such access permissions MAY be secured and audited using Proof-of-Authoritative-Execution (PoAE™) depending on the execution environment policy.

C5. Data Access as a Capability Property

C5.1

Access to data MUST be governed by Capability Tokens.

C5.2

Capability Tokens MUST explicitly define permitted data classes and access scopes.

C5.3

An agent MUST NOT access data beyond the scope defined in its Capability Tokens.

C5.4

Different agents within the same session MAY have access to different data sets or data classes.

C6. Risk Classification and Policy-Driven Consent

C6.1

All actions MUST be assigned to a risk class determined by policy.

C6.2

Risk classification MUST be evaluated deterministically prior to execution.

C6.3

The assigned risk class MUST determine whether consent is required and the form that consent takes, including but not limited to:

- explicit user confirmation,
- multi-factor authentication,
- Proof-of-Human or Proof-of-Authoritative-Execution mechanisms.

C6.4

If required consent cannot be established, execution MUST be denied.

C7. Authorization Semantics

C7.1

Authorization MUST be derived exclusively from:

- validated Capability Tokens,
- policy evaluation,
- risk classification outcomes.

C7.2

Agent reasoning, internal logic, or tool behavior MUST NOT influence authorization decisions.

C7.3

Authorization MUST be enforced prior to execution and MUST be fail-closed.

C8.1

The system MAY provide a catalog of reference templates to reduce setup time.

C8.2

Templates MUST serve only as co

Informative Section: Isolated Execution, Watchdog Oversight, and Post-Execution Verification

I. Isolated Agent Execution via BEAP Repackaging

In addition to the normative execution model, the system may support an isolation mechanism for selected agents or agent sub-flows. In such cases, the orchestrator or builder process may extract a subset of agents, actions, and their associated capabilities and repackage them into an isolated BEAP package.

This isolated BEAP package may then be transferred to a sandboxed execution environment. The sandbox environment is intended to operate with restricted authority, reduced attack surface, and limited interaction with the host system. Within the sandbox, agents execute strictly according to the capabilities and policies embedded in the packaged context.

Only the resulting outputs of the isolated execution, such as computed results, messages, or status signals, are returned to the originating environment. Return channels may include controlled APIs such as email delivery, messaging systems, or other predefined backchannels. Direct access to internal sandbox state, intermediate data, or execution context is not required.

This mechanism allows higher-risk or untrusted agent logic to be executed without exposing the primary system environment, while preserving deterministic authorization and auditability.

II. Watchdog Processes and Behavioral Oversight

The system may deploy local monitoring agents operating in parallel to execution agents. These watchdog processes are conceptually similar to host-based protection or integrity monitoring systems and are intended to observe agent behavior, execution patterns, and policy compliance.

Watchdog processes may analyze signals such as:

- agent activity patterns,
- capability usage consistency,
- boundary crossings,
- integrity signals from runtime environments.

Findings may be classified and escalated according to system policy. Lower-severity findings may be recorded locally, while higher-severity findings may be escalated for further analysis.

Where permitted by policy, abstracted and sanitized findings may be forwarded to more capable external analysis systems, including cloud-based AI services. Prior to any such escalation, personally identifiable information, credentials, and other sensitive data must be removed or irreversibly transformed. External analysis remains advisory and does not replace deterministic local enforcement.

III. Post-Execution Verification and PoAE™ Confirmation

Following execution, Proof-of-Authoritative-Execution (PoAE™) logs may be used to confirm that all actions were performed in accordance with declared intent, assigned capabilities, and enforced policies.

PoAE™ logs enable post-hoc verification that:

- execution paths matched authorized plans,
- no unauthorized capability usage occurred,
- required consent and policy gates were respected,
- sandboxed or isolated execution followed prescribed boundaries.

Verification results may be summarized in a structured outcome report, including pass and fail indicators for individual execution stages or policy checks. This report provides a reproducible and auditable record demonstrating that execution was completed as intended.

Informative Concept: Verified Runtime Environments and Client-Side Isolation

The use of isolated browser execution surfaces for autonomous agents provides a practical and scalable mechanism for enforcing capability-bound behavior within interactive environments. By constraining navigation, storage access, and input/output paths at runtime, such browser-based execution contexts can effectively prevent unintended privilege expansion and accidental data exposure in non-high-assurance scenarios.

However, when autonomous agents operate continuously and without direct human supervision, the execution surface itself becomes part of the overall trust model. In such settings, the correctness of policy enforcement depends not only on declared capabilities and server-side controls, but also on the integrity of the client-side runtime responsible for enforcing those constraints. Commodity browser environments, while highly hardened, inherently permit a degree of user-driven or environment-driven modification, including changes to extensions, profiles, runtime configuration, and update state.

As assurance requirements increase, this residual mutability of the execution surface becomes a limiting factor. While policy violations can be mitigated through off-band redirection and external isolation, the in-band execution environment remains partially dependent on procedural controls rather than technically unavoidable enforcement. This is particularly relevant for autonomous agents executing long-lived tasks, interacting with multiple systems, or operating across mixed-trust domains.

In later iterations, this limitation may be addressed through the use of a dedicated runtime environment in which the browser execution surface itself is subject to integrity constraints. In such an environment, extensions, configuration artifacts, and execution components may be cryptographically stamped and verified prior to use, ensuring that isolation boundaries and runtime constraints cannot be weakened through client-side modification. This approach does not introduce

new trust assumptions, but rather removes residual ones by aligning client-side enforcement with the same declarative and verifiable principles applied elsewhere in the system.

Importantly, the introduction of a verified or policy-enforced runtime environment does not alter the underlying capability or policy model. Instead, it increases the assurance with which that model can be upheld under adversarial or highly regulated conditions. For baseline and non-high-assurance operation, commodity browser environments combined with capability enforcement and off-band execution remain sufficient. The tailored runtime environment therefore represents a logical extension of the isolation model, rather than a prerequisite for its correctness.

Deterministic LLM-Governed App Ecosystem in WRDesk

Abstract

WRDesk introduces a governed application ecosystem designed for high-assurance automation environments. Unlike traditional app stores that rely primarily on human moderation or opaque automated checks, WRDesk establishes a deterministic, identity-bound, and cryptographically verifiable lifecycle for applications and browser extensions. All executable artifacts—whether first-party or third-party—are sandboxed by default, distributed as stamped code capsules, and evaluated through deterministic large language model (LLM)–based processes. Human input is permitted but intentionally subordinate to reproducible, machine-enforced evaluation. This article presents the architecture, governance model, agent instruction framework, sandbox-centric verification market, and feasibility considerations of this approach.

---

1. Design Goals and Non-Goals

The WRDesk ecosystem is designed around the following goals:

- Deterministic execution and evaluation
- Fail-closed security semantics
- Cryptographic provenance of code and updates
- Reproducible, auditable trust decisions
- Resistance against silent updates, social manipulation, and supply-chain ambiguity

Explicit non-goals include:

- Formal verification of arbitrary application logic
- Guaranteed detection of all vulnerabilities or zero-day exploits
- Fully autonomous trust escalation without human oversight

The system prioritizes risk reduction and transparency by requiring every app or extension to be created through a **normed, deterministic process** in which Zero Trust and security principles are

baked in from the outset. Compliance with these principles is not assumed at runtime, but is first required to pass a high-assurance initial security gate before entering the sandbox environment, and is later verified through distributed, deterministic scrutiny by the network.

---

2. Sandboxed-by-Default Application Model

Every application and WR-stamped browser extension in WRDesk begins its lifecycle with a mandatory `sandbox` label. Sandboxed artifacts:

- execute as normal apps or extensions within the WRDesk sandbox runtime, (e.g. mini pc)
- operate under explicitly declared and user-visible capability, API, and data-access controls,
- and are prevented from affecting other apps or extensions beyond user-approved and policy-defined interaction boundaries.

No artifact may opt out of sandboxing at creation time. Trust escalation is a separate, explicit process and never implicit.

---

3. Stamped Code Capsules and Identity Binding

Applications, extensions, and updates are distributed as **stamped code capsules**, a specialized BEAP-derived format. Each capsule cryptographically binds:

- executable code and assets,
- dependency graphs and version locks,
- declared capabilities and policies,
- producing identity (publisher, agent, organization),
- and lineage metadata linking updates to prior versions.

Updates are never applied in place. Every modification results in a new stamped capsule with an explicit parent reference. This ensures immutable history and prevents silent or partial upgrades.

---

4. Agent-Assisted Creation with Human-in-the-Loop

Artifact creation in WRDesk is agent-assisted but not agent-autonomous. Each app or extension is produced through a workflow that combines:

- a constrained, system-approved agent,
- mandatory human oversight,
- and deterministic validation before acceptance.

The agent is considered **part of the artifact itself** and is embedded into the stamped capsule as a governed component.

4.1 Dual Instruction Layers

Each agent operates under two instruction layers:

1. **Functional Instructions**
   Define purpose, workflows, inputs, outputs, and user-visible behavior.
2. **System-Verified Meta Instructions**
   Deterministic, machine-checkable instructions defining:
   - security expectations and forbidden behaviors,
   - performance and resource constraints,
   - determinism requirements,
   - declared scope and non-goals,
   - acceptable interaction patterns.

Only agents whose full instruction set passes system validation may generate sandboxed artifacts.

---

5. Deterministic and Distributed Agent Instruction Sets

Agent instruction sets are **deterministic and tamper-proof**, but intentionally **not uniform** across the ecosystem.

- Instruction sets are stamped and identity-bound.
- They are distributed across the network and selectively assigned.
- All instruction sets conform to strict schemas and validation rules.

Controlled variation is a design feature. In addition, **secure automation primitives**—including strict capability separation, time-bounded capabilities, scoped context usage, and explicit permission lifetimes—are deterministically provable and enforced during the build process. This ensures that a baseline for secure automation is structurally enforced rather than implicitly trusted. Different agents may emphasize:

- security hardening,
- performance stress analysis,
- side-effect discovery,
- determinism violations,
- ecosystem-level interaction risks.

This diversity broadens analytical bandwidth while remaining fully auditable and reproducible.

---

6. Sandbox as a Network-Enforced Analysis and Evaluation Market

The sandbox in WRDesk is not a lightweight preview environment. It is a **network-enforced analysis and evaluation market** in which sandboxed apps and extensions are examined before any production exposure is possible.

The WRDesk orchestrator always includes a **mandatory sandbox sub-orchestrator**. This sub-orchestrator is the only environment in which sandboxed artifacts can execute and enforces deterministic pre-usage checks automatically. These checks cannot be bypassed or deferred.

Before any user interaction, required test routines are executed by the sandbox sub-orchestrator. Unverified logic therefore never executes in trusted or production contexts.

---

7. Network-Enforced Swarm Verification for Sandboxed Artifacts

Mandatory distributed testing applies **exclusively to sandboxed artifacts**. Users interacting with sandboxed apps or extensions implicitly participate in structured verification.

Each interaction triggers execution of **deterministic test bundles** consisting of:

- a mandatory core test suite, and
- a rotating extension set drawn from an approved test corpus.

Test bundles are **hash-validated and whitelisted**. Only original, unmodified tests whose hashes match the approved corpus are accepted. Test results are collected as **tamper-proof, identity-bound artifacts**, cryptographically linked to both the artifact under test and the executed test bundle. Test artifacts themselves are treated as first-class objects: they undergo the same scrutiny, validation, and stamping processes as application artifacts and may be published independently for further swarm-based analysis and review.

Different users execute different, bounded test bundles. This controlled variation increases coverage while preserving determinism and comparability.

Human feedback and voting are permitted within the sandbox but are secondary and informational only. In addition to votes, **deterministic test summaries are automatically generated and posted as structured comments** for each sandboxed artifact. These summaries capture key findings, risk indicators, and coverage information derived from executed test bundles.

Reviewing and analyzing these test summaries is itself part of a defined verification scope. Dedicated evaluators—human and agent-based—may derive **improvement recommendations** from aggregated summaries where appropriate. Such recommendations are published as **flagged recommendation comments**, clearly distinguished from pass/fail results and non-binding with respect to trust progression.

Trust progression follows **fail-closed semantics**: any critical failure blocks advancement regardless of positive signals.

Marketplace eligibility requires sufficient volume of valid test artifacts, coverage across test bundle families, and participation from defined higher-trust identities. A **minimum percentage of all accepted test artifacts must originate from Pro or Publisher class identities**, ensuring that trust progression cannot be achieved through low-assurance participation alone.

---

8. Deterministic LLM-Based Evaluation

Formal trust decisions rely on deterministic evaluations executed by top-tier LLMs operating under strict constraints:

- pinned model versions of mandatory higher tiers,
- deterministic decoding parameters,
- schema-validated outputs,
- reproducible evaluation pipelines.

Each evaluation produces:

- hard **pass/fail gates**, and
- bounded numerical scores (1–100) used only for ranking and comparison.

Evaluation categories include:

1. Security and Capability Conformance
2. Performance and Resource Boundaries
3. Determinism and Reproducibility
4. Utility and Scope Alignment
5. Innovation and Side-Effect Analysis

Exploratory reasoning is permitted only within explicitly bounded sections and serves risk surfacing rather than guarantees.

---

9. Internal Pre-Marketplace Scrutiny and Admission

Sandboxed artifacts are not automatically eligible for marketplace listing. Before an app or extension may exit the sandbox, it undergoes **internal pre-admission scrutiny** by the WRDesk app team.

This step verifies:

- consistency across all collected test and evaluation artifacts,
- alignment between declared capabilities and observed behavior,
- absence of unresolved critical findings,
- conformance with current admission policies.

Internal scrutiny is confirmatory, not discretionary, and does not override deterministic failures. Artifacts that fail at any stage remain sandboxed or are rejected outright.

---

10. Feasibility and Risk Assessment

10.1 Feasible and Practical Elements

- Sandboxed execution and capability restriction
- Cryptographically stamped code distribution
- Deterministic LLM evaluation pipelines
- Network-enforced, hash-validated test execution
- Distributed, validated instruction diversity

These elements align with existing runtime, container, and cryptographic primitives.

10.2 Explicit Limitations

- Evaluation quality depends on the selected LLM class and version. Only the latest versions of frontier models are allowed to participate.
- Deterministic inference does not imply perfect correctness.
- Zero-day discovery is probabilistic and exploratory.

These limitations are explicit and integral to system credibility.

---

11. Conclusion

WRDesk does not attempt to eliminate risk through claims of perfect automation. Instead, it establishes a **sandbox-centric, deterministic, and identity-bound governance framework** for applications and browser extensions. By combining normed creation, stamped code capsules, mandatory sandbox orchestration, network-enforced swarm verification, and deterministic LLM-based evaluation, WRDesk offers a pragmatic alternative to socially moderated or opaque marketplaces.

The result is not a generic app store, but a governed code commons designed for environments where predictability, traceability, and controlled evolution matter more than unchecked speed or scale.

IV. Informative Scope

This section is informative and does not introduce additional normative requirements. It illustrates optional architectural patterns that complement the agent-scoped capability model and deterministic validation pipeline described elsewhere in this specification.

WRVault™ (Terminology Clarification — Single Sentence, Normative)

WRVault™ denotes a locally controlled, encrypted, multi-layered secure vault bound to the executing environment and identity, combining password management, PII storage, and sensitive data protection, in which all data **SHALL** be compartmentalized, scope- and purpose-restricted, cryptographically isolated by class, never directly accessible, and only exposed through controlled, augmented overlay contexts to prevent unauthorized extraction, misuse, or cross-domain leakage.

License Notice