PRODUCT AND ENGINEERING SPECIFICATION

Torsionfield Runtime

Capability, authority, evidence, verification, and acceptance

EDITION
Version 3.0

DATE
1 August 2026

STATUS
Normative integrated rewrite

SOURCE
Complete v2.4 specification

This edition rewrites the document for navigability, implementation traceability, accessible screen reading, and compact paged-media output. All 37 source sections are retained and mapped. Normative clauses receive stable requirement identifiers without changing their force.

WHAT THIS SYSTEM IS

A programmable browser runtime with explicit authority and evidence.

Torsionfield begins as a ScriptCat-derived extension and a local Torsion Node. It turns browser pages, AI sessions, userscripts, local tools, and optional peers into typed capability providers connected by durable operations.

The first release is not a browser fork, a token economy, or a promise of autonomous correctness. It is a working browser runtime whose effects are attributable, observable, recoverable, and accepted only under an explicit policy.

GOVERNING RULE

Nothing is silently prohibited. Nothing is silently trusted. Nothing is silently verified. Nothing is silently accepted.

THE DIRECT IMPLEMENTATION PATH

Build one complete ScriptCat-first vertical slice.

01

Extension

Router, Mother Script, provider adapters, target selection, mutation, installation, and protected agent tabs.

02

Torsion Node

Authenticated local IPC, durable journal, files, processes, artifacts, identity, and bounded runners.

03

Work and verification

Signed tasks, participant-owned providers, encrypted bundles, quarantine, receipts, independent review, and rollback.

04

Optional network

Peer discovery, revocable capabilities, project coordination, and first-party domain integration.

DO NOT COLLAPSE THESE CONCEPTS

Four independent decisions govern every consequential result.

Permission

May this operation execute under the current authority profile?

Participant trust

How reliable is this participant for this task, role, consequence, device, and context?

Assurance

What properties of identity, execution, observation, isolation, recovery, and attestation are actually supported?

Acceptance

Who is willing to rely on this exact result, for which purpose, despite what remaining uncertainty?

IMPLEMENTATION INTERPRETATION

Every section is labelled by delivery status.

FIRST RELEASE	Required by the direct ScriptCat-first implementation or its release gates.
OPTIONAL NATIVE	Stronger browser-native guarantees that are not on the first-release critical path.
STAGED TARGET	Architecturally defined future work that must not delay the complete vertical slice.
REFERENCE	Source material, interoperability targets, and implementation references.

NORMATIVE LANGUAGE

Requirement force is preserved.

MUST and **MUST NOT** define acceptance requirements. **SHOULD** identifies the preferred implementation unless a documented constraint requires another choice. **MAY** identifies an optional capability.

This edition identified **94** normative clauses. Each receives a stable identifier such as `TF-S07-R004` and an exact SHA-256 digest in the downloadable requirements manifest.

DOCUMENT MAP

Contents

PART I

Mandate and architecture

§1

FIRST RELEASE

Purpose and delivery target

> **Purpose.** Defines the product that must be delivered and the evidence standard for calling it complete.

The system turns AI chat webpages into programmable browser runtimes. A modified ScriptCat extension, a Mother userscript, provider adapters, protected agent tabs, a local Torsion Node daemon, and optional peer/cloud services cooperate to generate, install, execute, repair, and share userscripts.

TF-S01-R001 The delivered system **MUST** provide, as one integrated build:

1. streamed assistant-output observation and command extraction;
2. transactional self-mutation of the Mother runtime;
3. native ScriptCat userscript installation, update, execution, and rollback;
4. operation-scoped routing between coding tabs, agent tabs, target tabs, the daemon, peers, and projects;
5. user-guided target-tab and DOM-element selection;
6. provider-specific prompt insertion, submission, attachment, output extraction, and completion detection;
7. recovery from service-worker suspension, tab reload, SPA navigation, frontend changes, daemon restart, and network interruption;
8. configurable local and remote capabilities, including an explicitly unrestricted owner profile;
9. protected and restorable AI agent tabs;
10. optional NetworkScript exposure and peer task delegation;
11. structured observability, provenance, and acceptance tests.

The target is not a demonstration or a collection of disconnected scripts. Acceptance requires a working installation in which every effect has a durable operation identity, every nontrivial action returns evidence, and uncertain effects are observed before retry.

§2

FIRST RELEASE

Architectural invariants

> **Purpose.** States the rules that remain true across every implementation: durable identity, observable effects, explicit authority, recoverability, and local credentials.

2.1 Chat webpages are first-class runtimes

A supported AI site is represented by a ProviderAdapter. The adapter owns the complete web interaction contract for that site:

- match the correct host and route;
- identify or create a conversation;
- locate and verify the composer;
- insert and verify prompt text;
- submit and verify the submission effect;
- attach files and verify attachment state;
- identify user and assistant message roots;
- observe streaming output incrementally;
- detect generation state and completion;
- extract final text, code blocks, tagged commands, and diagnostics.

No provider is implemented as a URL plus one hard-coded selector.

2.2 Self-mutation remains a real capability

TF-S02-R001 The Mother Script **MUST** be able to replace its active operational core from model output. Staging, validation, versioning, leases, and rollback protect continuity; they **MUST NOT** reduce self-mutation to manual copy-and-paste.

2.3 Generated userscripts are real ScriptCat scripts

TF-S02-R002 A generated userscript **MUST** be installable into ScriptCat's canonical script registry with its full metadata block, source, grants, matches, resources, version, enabled state, and durable identity. A downloaded .user.js file is only a compatibility fallback.

2.4 The background router is the canonical runtime bus

TF-S02-R003 GM storage **MAY** hold durable state, snapshots, deduplication records, and recovery markers. It **MUST NOT** be the primary request/response bus. Runtime messages pass through the extension service worker and, where needed, an offscreen runtime, using explicit source, destination, operation, attempt, generation, epoch, and reply information.

2.5 Durable identities are separate from browser transport addresses

Raw Chrome tabId and windowId values are ephemeral hints. Routing and recovery use stable logical identities and resolve them to current browser IDs at execution time.

2.6 Effects are evidence-based

TF-S02-R004 A dispatched click, synthetic event, or API call is not itself success. Prompt insertion, submission, attachment, mutation, installation, and remote invocation **MUST** return postcondition evidence. Boolean-only success results are insufficient for routed effects.

2.7 User authority is expressible, not silently narrowed

TF-S02-R005 The policy engine **MUST** support restrictive profiles and explicitly unrestricted profiles. The software records and displays the selected authority; it does not substitute a narrower policy without the user's decision.

2.8 Agent-tab protection is operational

TF-S02-R006 Protection includes ownership, routing identity, group/workspace placement, focus and input policy, navigation policy, anti-discard behavior, close handling, and restoration. A stock extension cannot make a tab literally uncloseable; it restores a protected tab after closure. A Chromium-derived build **MAY** enforce native close inhibition.

2.9 Participant credentials remain local

TF-S02-R007 Peer and project tasks run through each participant's own configured provider sessions and capabilities. The architecture **MUST NOT** require centralized possession of participants' model-account credentials.

§3

FIRST RELEASE

Verified ScriptCat baseline and required extensions

> **Purpose.** Fixes the verified ScriptCat foundation and separates existing CAT capabilities from additions required by Torsionfield.

TF-S03-R001 Implementation **MUST** be based on the ScriptCat source and documented interfaces present at build time, not on signatures inferred from earlier design discussion.

The baseline verified on 31 July 2026 includes:

- ScriptCat v1.4 testing-phase CAT.agent.* APIs;
- @grant CAT.agent.dom;
- DOM operations including listTabs, navigate, readPage, screenshot, click, fill, scroll, waitFor, executeScript, startMonitor, stopMonitor, and peekMonitor;
- CAT.agent.dom.executeScript(code, options?);
- MAIN and ISOLATED execution worlds;
- background userscripts that run without direct DOM access;
- CAT.agent.conversation, CAT.agent.opfs, CAT.agent.model, CAT.agent.task, CAT.agent.skills, and CAT.agent.mcp;
- an existing architecture spanning userscript sandbox, offscreen document, service worker, provider, tool registry, MCP client, and task scheduler.

TF-S03-R002 The implementation **SHOULD** reuse an existing CAT.agent.* operation when it already meets the requirement. The following are explicit extension additions:

- Router Core;
- Agent Tab Manager;
- Provider Adapter Host and adapter configuration service;
- Script Mutation Manager;
- direct userscript install/update/rollback APIs;
- NetworkScript Manager;
- Daemon Bridge;
- Identity and Capability Manager;
- protected-tab lifecycle and restoration;
- adapter health, hotfix, fingerprint, and rollback services.

TF-S03-R003 A background userscript **MUST NOT** be treated as if it directly owns unrestricted chrome.tabs, chrome.tabGroups, chrome.scripting, or native-messaging objects. Privileged browser functions belong to the extension and are exposed through typed, granted CAT APIs.

§4

FIRST RELEASE

System architecture

> **Purpose.** Explains the runtime roles, execution planes, and the canonical path of a delegated browser operation.

4.1 Runtime roles

Coding Tab An AI conversation in which a user or orchestrator describes work and receives code, commands, diagnostics, or results.

Agent Tab An AI conversation owned by the Agent Tab Manager for delegated execution. It has a durable agentId independent of its current tabId.

Target Tab A webpage being inspected, selected, modified, tested, or exposed as an endpoint.

Mother Script The userscript runtime in coding and agent tabs. It observes assistant output, parses commands, invokes CAT APIs, manages its mutable core, and routes results.

Target Probe A minimal content runtime used for DOM selection, element fingerprinting, observation, trace capture, and target-side execution.

4.2 Execution planes

Chat Runtime Plane Coding tabs, agent tabs, provider adapters, Mother Script, and output observers.

ScriptCat Plane Userscript sandbox, script registry, OPFS, offscreen runtime, service worker, Router Core, mutation manager, adapter host, tab manager, identity/capability manager, and daemon bridge.

Local Node Plane The resident Torsion Node daemon, which supplies authenticated local IPC, process and filesystem endpoints, identity, a durable journal, peer networking, artifact storage, and update/install services.

Peer Data Plane Encrypted libp2p sessions for discovery, capability exchange, invocation, result transport, and artifact transfer.

Cloud Control Plane Optional Cloudflare Workers, Durable Objects, R2, D1/SQLite, Queues, and Tunnel services for authenticated rendezvous, presence, project coordination, task leases, release distribution, and optional stable local exposure.

Optional Browser-Native Plane A Chromium-derived implementation that deepens tab protection, trusted input, daemon IPC, adapter lifecycle, anti-discard policy, and operation tracing.

Observability Plane A common event stream correlated by operationId across browser, userscripts, daemon, peer transport, and project services.

4.3 Canonical execution flow

A typical delegated prompt follows this sequence:

1. The origin creates an operationId and sends agent.prompt to the Router.
2. The Router resolves the agentId to a current tab binding and validates the capability profile.
3. The adapter host resolves the ProviderAdapter for the current URL and document epoch.
4. The adapter locates and fingerprints the composer.
5. The adapter writes the prompt and verifies the resulting composer state.
6. The adapter submits and waits for a submission postcondition.
7. The output observer binds only to the new conversation turn or relevant conversation subtree.
8. Partial output events are routed to the origin.
9. Complete command envelopes are parsed once and deduplicated.
10. Completion is established from provider state plus quiescence.
11. Final output, commands, artifacts, adapter evidence, and diagnostics are returned.

12. The operation journal records the terminal or unknown_outcome state.

§5

FIRST RELEASE

Repository and deliverable layout

> **Purpose.** Defines the repository shape, release artifacts, and source boundaries an implementation must actually produce.

TF-S05-R001 The integrated repository **SHOULD** use this layout:

```
scriptcat-transductive/
  extension/
    src/background/router/
    src/background/agent-tabs/
    src/background/script-mutation/
    src/background/network-scripts/
    src/background/daemon/
    src/background/identity/
    src/background/capabilities/
    src/background/adapters/
    src/offscreen/
    src/content/selection/
    src/content/protection/
    src/content/provider-adapters/
    src/ui/agent-workspace/
    src/ui/permissions/
    src/types/cat-transductive.d.ts
  userscripts/
    mother/ScriptCat-Mother.user.js
    target-probe/ScriptCat-Target-Probe.user.js
    network-bridge/Transductive-Network-Bridge.user.js
    examples/
  daemon/
    crates/torsion-node/
    crates/torsion-protocol/
    crates/torsion-policy/
    crates/torsion-libp2p/
    crates/torsion-cloud/
    crates/torsion-keystore/
    crates/torsion-runner/
  cloud/
    workers/control-plane/
    durable-objects/node-presence/
    durable-objects/project-room/
    durable-objects/task-room/
  shared/
    schemas/
    protocol/
    test-vectors/
  tests/
    extension/
    userscripts/
    providers/
    daemon/
    network/
    end-to-end/
  packaging/
    windows/
    linux/
    macos/
    extension/
  docs/
    implementation-spec.md
    protocol.md
    provider-adapters.md
    permission-model.md
    browser-fork.md
```

Required release artifacts:

- installable modified ScriptCat extension;
- Mother Script and Target Probe;
- Torsion Node binaries and installers for Windows, Linux, and macOS;
- provider adapter descriptors, fixtures, and test vectors;
- shared JSON Schemas and TypeScript/Rust protocol types;

- optional Cloudflare control-plane project;
- automated acceptance runner;
- signed extension, daemon, adapter-pack, and release metadata.

§6

FIRST RELEASE

Canonical identities and data model

> **Purpose.** Names the stable identities used for browsers, tabs, agents, operations, projects, peers, capabilities, and artifacts.

nodeId Long-lived identity of a Torsion Node installation.

browserInstanceId Identity of one browser profile and ScriptCat installation.

scriptId Durable ScriptCat script identity.

tabInstanceId Stable orchestrator identity assigned to a browser tab. tabId is only a current transport hint.

conversationId Provider conversation identity when discoverable.

agentId Durable logical agent identity that can be rebound to a restored tab.

endpointId A callable provider, tab, NetworkScript, daemon command, or composite endpoint.

operationId Globally unique identity of one requested effect.

attemptId Identity of one execution attempt under an operation.

generation Monotonically increasing binding generation. Responses from older generations are rejected.

epoch Identity of the currently loaded document and observation context.

projectId Transductive project identity.

peerId libp2p identity of a remote node.

capabilityProfileId Policy profile authorizing an operation.

artifactId Immutable content-addressed script, trace, prompt, result, screenshot, adapter pack, or file.

Every routed message resolves logical identities to current transport bindings. No durable state is keyed only by tabId.

PART II

Browser runtime and operations

§7

FIRST RELEASE

Router core and operation semantics

> **Purpose.** Defines routing, delivery, deduplication, recovery, and unknown-outcome semantics for every requested effect.

7.1 Responsibilities

TF-S07-R001 The Router Core **MUST**:

- register and lease endpoints;
- assign and reconcile tabInstanceId values;
- validate command and RPC envelopes;
- resolve logical destinations;
- evaluate capabilities;
- create operation and attempt records;
- route requests, responses, events, cancellation, and acknowledgements;
- reject stale generation or epoch messages;
- correlate concurrent work without global storage keys;
- record minimal durable recovery state;
- emit structured trace events;
- observe uncertain effects before retry.

7.2 RPC envelope

```
{
  "protocol": "mf-router/1",
  "messageId": "msg_ … ",
  "operationId": "op_ … ",
  "attemptId": "attempt_ … ",
  "kind": "request|response|event|cancel|ack",
  "method": "agent.prompt",
  "source": {
    "nodeId": " … ",
    "browserInstanceId": " … ",
    "tabInstanceId": " … ",
    "endpointId": " … "
  },
  "destination": {
    "nodeId": " … ",
    "browserInstanceId": " … ",
    "tabInstanceId": " … ",
    "endpointId": " … "
  },
  "generation": 4,
  "epoch": " … ",
  "capabilityProfileId": " … ",
  "deadline": " … ",
  "payload": {},
  "signature": " … "
}
```

7.3 Operation states

created awaiting_permission accepted routing awaiting_user_selection awaiting_agent_ready running awaiting_result succeeded failed cancelled timed_out unknown_outcome

TF-S07-R002 unknown_outcome means the effect may have occurred but confirmation was lost. A non-idempotent operation in this state **MUST** be observed before any retry.

7.4 Delivery guarantees

Transport delivery is at least once. Target effects are effectively once through operationId deduplication and effect observation.

TF-S07-R003 Read-only operations **MAY** be retried automatically. Non-idempotent operations **MAY** be retried only after the target state proves that the effect did not occur.

Responses are rejected when the operation, attempt, generation, epoch, or endpoint lease no longer matches.

§8

FIRST RELEASE

Mother script and command protocol

> **Purpose.** Defines the Mother Script, command envelopes, compatibility tags, acknowledgements, and incremental output parsing.

8.1 Grants

The Mother declares the grants actually exposed by the selected ScriptCat build, including:

```
@grant GM_getValue
@grant GM_setValue
@grant GM_deleteValue
@grant GM_notification
@grant CAT.agent.dom
@grant CAT.router
@grant CAT.agent.tabs
@grant CAT.agent.adapters
@grant CAT.script.mutation
@grant CAT.identity
@grant CAT.daemon
@grant CAT.network
```

TF-S08-R001 The implementation **MUST NOT** invent undocumented GM_sendMessage or GM_addMessageListener signatures. The fork's source and type definitions are authoritative.

8.2 Boot sequence

BOOTSTRAP Load configuration, identity, adapter host, Router connection, and active core descriptor.

RECOVER Ask the Router for operations and endpoint leases associated with the current tabInstanceId and conversationId.

EXECUTE_CORE Resolve the active immutable core artifact and run it through the mutation manager or CAT.agent.dom.executeScript(code, options).

OBSERVE Bind the active provider's output observer to the current conversation and document epoch.

READY Register endpoint capabilities and accept routed work.

8.3 Legacy command compatibility

The parser continues to recognize:

```
[[MUTATION_START]] ... [[MUTATION_END]]
[[MUTATE_CORE]] ... [[END_MUTATE]]
[[SPAWN_USERSCRIPT]] ... [[END_SPAWN]]
[[SPAWN_AGENT: ChatGPT]]
[[SPAWN_AGENT: Gemini]]
[[SPAWN_AGENT: Claude]]
[[MF:USER SELECT TARGET TAB AND WITHIN TARGET TAB CONTENTEDITABLE]]
```

Legacy tags are translated into the canonical command envelope before execution.

8.4 Canonical command envelope

```
[[MF_COMMAND/1]]
{
  "operationId": "op_ … ",
  "command": "agent.spawn",
  "arguments": {},
  "replyMode": "composer",
  "dedupeKey": "sha256: … "
}
[[/MF_COMMAND]]
```

A command is executable only when:

- both delimiters are present;
- the JSON parses and validates;
- the enclosing assistant-message identity is known;
- the command or dedupe digest has not already been acknowledged;
- any declared payload hash matches;
- the selected capability profile authorizes it.

8.5 Canonical command families

mother.mutateCore

script.install script.update script.enable script.disable script.run script.rollback

agent.spawn agent.attach agent.prompt agent.cancel agent.release

tab.select element.select tab.read tab.screenshot tab.execute

trace.start trace.stop

daemon.invoke

network.expose network.unexpose network.invoke network.delegate network.publishResult

project.join project.leave

permission.request operation.status

8.6 Acknowledgement envelope

```
[[MF_ACK/1]]
{
  "operationId": "op_ … ",
  "status": "accepted|running|succeeded|failed|unknown_outcome",
  "resultRef": "artifact_ … ",
  "error": null
}
[[/MF_ACK]]
```

Acknowledgements are routed or stored; the Mother does not rewrite assistant messages. The acknowledgement digest prevents DOM rehydration, virtualization, scrolling, or reload from executing the same command twice.

8.7 Incremental parser

TF-S08-R002 The parser **MUST NOT** rescan document.body.textContent on every mutation. The ProviderAdapter identifies assistant-message roots. A scoped MutationObserver tracks:

- newly created assistant turns;
- text and code descendants of the current streaming turn;
- generation and finalization state.

For each message, the parser stores the accumulated buffer, last processed offset, command ranges, message identity, epoch, and finalization state. Whole-document scanning is a degraded fallback only.

§9

FIRST RELEASE

Script mutation and installation

> **Purpose.** Makes self-mutation and userscript installation transactional, versioned, testable, and reversible.

9.1 Mother core storage

```
mother/
  active.json
  candidates/{artifactId}.json
  artifacts/{sha256}.js
  history.jsonl
  traces/{operationId}.jsonl
```

active.json contains the current artifactId, semantic version, activation time, previousArtifactId, source operationId, and boot generation.

9.2 Mutation transaction

TF-S09-R001 mother.mutateCore **MUST**:

1. capture and hash the exact payload;
2. store an immutable candidate;
3. parse JavaScript and metadata;
4. run a syntax and isolated-load probe;
5. run an exported self-test when present;
6. atomically point active.json at the candidate;
7. restart only the Mother runtime;
8. require a boot-success lease from the new generation;
9. restore previousArtifactId if the lease is not acquired;
10. return the candidate, activation, boot, and rollback evidence.

Validation and rollback do not require human approval unless the active capability profile requests it.

9.3 Userscript installation transaction

script.install accepts:

```
{
  "source": "…full userscript…",
  "enabled": true,
  "installMode": "native|installer-ui|download",
  "conflictMode": "update|fork|replace|error",
  "runAfterInstall": true
}
```

The Script Mutation Manager parses metadata, resolves durable identity, validates grants and match patterns, stores source in the canonical registry, records provenance and hashes, refreshes runtime caches, activates the script, optionally runs it against a target, and returns scriptId, version, grants, matches, and execution evidence.

The .user.js Blob installer remains a fallback when native registry installation is unavailable.

9.4 Debug feedback loop

Generated-script failures include scriptId, version, tabInstanceId, URL, operationId, stack, console records, rejected-promise details, recent relevant DOM mutations, and an optional screenshot reference.

The Router sends a compact diagnostic to the originating coding composer. Auto-submit of the repair prompt is an endpoint policy, not an unconditional behavior.

§10 FIRST RELEASE

User-guided target selection

> **Purpose.** Specifies the human-guided tab and element selection flow and the evidence returned to the requesting operation.

The selection macro is translated to element.select with an explicit element constraint.

Selection flow:

1. The Router records the requesting operation and origin.
2. Agent Workspace enters selection mode.
3. The user chooses a target tab or activates it directly.
4. Target Probe is injected or activated.
5. Eligible elements are highlighted.
6. Click selects; Escape cancels.
7. Target Probe creates a selector bundle and element fingerprint.
8. The Router returns it only to the requesting operation.
9. The Mother inserts the result into the origin composer.
10. Endpoint policy determines whether the prompt remains for review or is submitted.

Selector bundle:

```
{
  "tab": {
    "tabInstanceId": "…",
    "tabIdHint": 123,
    "windowIdHint": 8,
    "url": "…",
    "title": "…",
    "epoch": "…"
  },
  "element": {
    "primarySelector": "…",
    "selectorCandidates": [
      {"selector": "…", "score": 0.98, "strategy": "id"},
      {"selector": "…", "score": 0.91, "strategy": "aria"},
      {"selector": "…", "score": 0.72, "strategy": "structural"}
    ],
    "fingerprint": {},
    "tag": "div",
    "role": "textbox",
    "contenteditable": "true",
    "attributes": {},
    "boundingRect": {},
    "textPreview": "…",
    "outerHTMLPreview": "…",
    "framePath": [],
    "shadowPath": []
  }
}
```

Selector preference order:

1. unique stable ID;
2. explicit test or stable data attribute;
3. accessible role and name;
4. name, type, and placeholder combination;
5. stable class subset;
6. scoped structural selector;
7. text anchor as a last resort.

Every candidate is verified against the current document before return.

§11

FIRST RELEASE

Provider adapter subsystem

> **Purpose.** Defines provider adapters as complete interaction contracts rather than brittle collections of selectors.

11.1 Source boundary

The initial adapter seed is derived from srbhptl39/MCP-SuperAssistant, branch main, primarily:

```
pages/content/src/plugins/adapters/
pages/content/src/plugins/plugin-types.ts
pages/content/src/plugins/plugin-registry.ts
```

The source provides a plugin registry, BaseAdapterPlugin lifecycle, site adapters, a universal fallback, and remote adapter configuration. These mechanisms are integrated into ScriptCat rather than copied as isolated selectors into the Mother Script.

MCP-SuperAssistant's adapter registry supplies first-class seeds for:

- ChatGPT;
- Gemini;
- GitHub Copilot;
- DeepSeek;
- Grok;
- Perplexity;
- Google AI Studio;
- OpenRouter;
- T3 Chat;
- Mistral Le Chat;
- Kimi;
- Z.ai;
- Qwen Chat;
- a generic fallback.

TF-S11-R001 It does not contain a dedicated Claude adapter in this registry. Claude remains a required extensible provider target, but its implementation **MUST** be independently sourced or verified and **MUST NOT** be described as repository-derived until that is true.

SidebarAdapter and RemoteConfigAdapter are infrastructure components, not provider endpoints.

11.2 Adapter capabilities and lifecycle

```
type AdapterCapability =
  | "text-insertion"
  | "form-submission"
  | "file-attachment"
  | "url-navigation"
  | "element-selection"
  | "screenshot-capture"
  | "dom-manipulation"
  | "output-observation";
```

```
interface AdapterPlugin {
  readonly name: string;
  readonly version: string;
  readonly hostnames: readonly string[];
  readonly capabilities: readonly AdapterCapability[];
```

```
initialize(context: PluginContext): Promise<void>;
  activate(): Promise<void>;
  deactivate(): Promise<void>;
  cleanup(): Promise<void>;
}
```

BaseAdapterPlugin owns idempotent lifecycle transitions, PluginContext, event-bus integration, trace correlation, normalized errors, cleanup registration, and Router integration.

11.3 ProviderAdapter contract

```
type EditorKind =
  | "input"
  | "textarea"
  | "contenteditable"
  | "prosemirror"
  | "lexical"
  | "quill"
  | "unknown";
```

```
type SelectorPurpose =
  | "composer"
  | "submit"
  | "stop"
  | "fileButton"
  | "fileInput"
  | "mainPanel"
  | "dropZone"
  | "filePreview"
  | "buttonInsertion"
  | "userMessage"
  | "assistantMessage"
  | "assistantMessageContent"
  | "generationIndicator";
```

```
interface SelectorCandidate {
  id: string;
  selector: string;
  purpose: SelectorPurpose;
  confidence: number;
  source: "repository" | "built-in" | "remote" | "learned" | "manual";
  expectedTag?: string[];
  expectedRole?: string[];
  expectedEditorKind?: EditorKind[];
  mustBeVisible?: boolean;
  mustBeEnabled?: boolean;
  mustBeUnique?: boolean;
  semanticChecks?: string[];
  fingerprint?: ElementFingerprint;
  introducedAt?: string;
  expiresAt?: string | null;
}
```

```
interface SiteAdapterDescriptor {
  id: string;
  version: string;
  hostMatchers: string[];
  pathMatchers?: string[];
  priority?: number;
  capabilities: AdapterCapability[];
  editorKinds: EditorKind[];
  selectors: Partial<Record<SelectorPurpose, SelectorCandidate[]>>;
  insertionDriver:
    | "native-value"
    | "contenteditable-range"
    | "prosemirror"
    | "lexical"
    | "quill"
    | "auto";
  submissionDrivers: (
    | "button-click"
    | "form-request-submit"
    | "enter"
    | "ctrl-enter"
    | "meta-enter"
    | "trusted-input"
  )[];
  attachmentDrivers: (
    | "file-input"
    | "data-transfer"
    | "drag-drop"
    | "open-file-menu"
    | "trusted-attachment"
  )[];
  routePolicy?: {
    newConversationPaths?: string[];
    conversationIdPatterns?: string[];
  };
  featureFlags?: Record<string, boolean>;
}
```

```
interface ProviderAdapter extends AdapterPlugin {
  readonly descriptor: SiteAdapterDescriptor;
```

```
identifyConversation(): Promise<ConversationIdentity>;
  locateComposer(options?: LocateOptions): Promise<ResolvedElement>;
  setComposerText(
    text: string,
    options?: ComposerWriteOptions
  ): Promise<WriteResult>;
  verifyComposerText(expected: string): Promise<VerificationResult>;
```

```
locateSubmitControl(): Promise<ResolvedElement | null>;
  submitPrompt(options?: SubmitOptions): Promise<SubmitResult>;
  verifySubmission(
    before: SubmissionSnapshot
  ): Promise<SubmissionVerification>;
```

```
attachFiles?(
    files: File[],
    options?: AttachOptions
  ): Promise<AttachmentResult>;
```

```
locateAssistantMessages(): Promise<MessageHandle[]>;
  observeAssistantStream(handler: StreamHandler): Unsubscribe;
  extractMessage(handle: MessageHandle): Promise<ExtractedMessage>;
  isGenerating(): Promise<GenerationState>;
  stopGeneration?(): Promise<boolean>;
```

```
healthCheck(
    options?: HealthCheckOptions
  ): Promise<AdapterHealth>;
  discoverFallback?(
    purpose: SelectorPurpose
  ): Promise<DiscoveryResult>;
}
```

Every action result includes operationId, attemptId, adapter ID and revision, candidate ID, generation, epoch, element fingerprint, before/after observations, fallback stages, and postcondition result.

11.4 Registry and activation

TF-S11-R002 The factory-based registry **MUST**:

1. register built-in descriptors and lazy factories;
2. match hostname and optional path;
3. prefer exact host and path matches over wildcards;
4. resolve ties by configured priority and compatible version;
5. activate one provider adapter plus explicitly enabled infrastructure adapters;
6. deactivate and clean up the previous adapter when SPA navigation changes context;
7. expose the active adapter and health state through CAT.agent.adapters;
8. use GenericAdapter only when no specific adapter is healthy;
9. never map an unknown site to an unrelated provider default.

TF-S11-R003 Navigation is detected through history.pushState, history.replaceState, popstate, hashchange, and extension tab URL events. A low-frequency reconciliation probe **MAY** exist, but permanent high-frequency URL polling **MUST NOT** be duplicated in every adapter.

11.5 Shared interaction drivers

Native value driver Uses the native HTMLInputElement or HTMLTextAreaElement value setter when frameworks wrap the instance property. It focuses, writes, updates selection, emits applicable beforeinput/input/change/selectionchange events, and verifies the value. It supports replace, append, prepend, and insert-at-selection.

TF-S11-R004 Contenteditable driver Uses Selection and Range without destroying editor-owned root attributes. It preserves provider-appropriate line breaks, emits beforeinput/input, and verifies accessible and DOM text. It **MUST NOT** replace innerHTML indiscriminately.

ProseMirror driver Targets a visible .ProseMirror[contenteditable="true"], preserves the root, uses editor-compatible input events, and verifies DOM text and observable transaction state. Direct paragraph replacement is a fallback only.

Lexical driver Recognizes data-lexical-editor="true", performs selection-aware insertion, and escalates to trusted input when synthetic events are ignored.

Quill driver Targets the .ql-editor root, preserves required paragraph structure, emits input, and verifies editor text.

Submission driver Attempts configured methods in order: valid send control, form.requestSubmit(), provider keyboard shortcut, then trusted input. Success requires a postcondition such as composer clearing, creation of a user message, transition to stop/cancel state, generation beginning, or a conversation identity change. A dispatched click without a postcondition returns SUBMIT_NOT_CONFIRMED.

Attachment driver Attempts a matching file input with DataTransfer, opens an attachment menu when needed, performs validated drag/drop, then uses trusted attachment if available. Success requires file-input state, a matching preview, upload progress, or provider attachment-model evidence. Opening a file picker is not success.

11.6 Repository-derived provider descriptors

Selectors below are ordered seed candidates. Runtime validation, confidence, visibility, editor kind, semantics, and fingerprint determine whether a candidate may be used.

11.6.1 ChatGPT

Adapter: chatgpt-web Host: chatgpt.com Editor: ProseMirror/contenteditable Composer candidates: - #prompt-textarea - .ProseMirror[contenteditable="true"] - div[contenteditable="true"][data-id*="prompt"]

Submit candidates: - button[data-testid="send-button"] - button[aria-label*="Send"] - button[data-testid="fruitjuice-send-button"]

Attachment candidates: - #upload-file-btn - button[aria-label*="Add photos"] - button[data-testid="composer-action-file-upload"] button - input[type="file"][multiple]

The adapter prefers #prompt-textarea when it is the visible ProseMirror root, verifies text after insertion, uses DataTransfer before drag/drop, and scopes output observation to conversation-turn roots.

11.6.2 Gemini

Adapter: gemini-web Host: gemini.google.com Editor: Quill/contenteditable Composer candidates: - div.ql-editor.textarea.new-input-ui - .ql-editor - div[contenteditable="true"]

Submit candidates: - button.mat-mdc-icon-button.send-button - button[aria-label*="Send"] - button[data-testid="send-button"]

Attachment candidates: - button[aria-label="Add files"] - button[aria-label*="attach"] - input[type="file"] - div[xapfileselectordropzone]

The adapter binds the Quill root, not only its current paragraph, and re-resolves the active editor after navigation or reconstruction. Remote configuration updates invalidate cached bindings while retaining valid built-in defaults.

11.6.3 GitHub Copilot

Adapter: github-copilot-web Host/path: github.com/copilot Editor: textarea Composer candidates: - #copilot-chat-textarea - textarea[placeholder*="How can I help"] - .ChatInput-module__input--iApWs

Submit candidates: - button[aria-labelledby*="Send"] - button:has(.octicon-paper-airplane) - the validated toolbar send button

Attachment candidates: - button[data-testid="attachment-menu-button"] - button[aria-label*="Attach"] - button:has(.octicon-paperclip) - #image-uploader - matching hidden or multiple file inputs

Path matching is mandatory. Stable IDs, test attributes, aria labels, and Octicon semantics outrank CSS-module classes.

11.6.4 DeepSeek

Adapter: deepseek-web Host: chat.deepseek.com Editor: textarea/contenteditable Composer candidates: - textarea[spellcheck="false"] - textarea[data-gramm="false"] - textarea[placeholder*="Message DeepSeek"] - textarea[placeholder*="Ask"] - textarea.chat-input - a validated contenteditable

Submit candidates: - button[aria-label*="Send"] - button[data-testid="send-button"] - button.send-button

Generated hash classes found in the source, including .ec4f5d61, ._24fad49, .bf38813a, and .aaff8b8f, are low-confidence, expiry-bound fallbacks and never outrank semantic candidates.

11.6.5 Grok

Adapter: grok-web Hosts: grok.com and Grok-specific routes on x.com Editor: textarea/contenteditable Composer candidates: - textarea[aria-label="Ask Grok anything"] - textarea[placeholder="Ask anything"] - validated Grok-specific textarea markers

Submit candidates: - button[aria-label="Submit"] - button[aria-label="Send message"] - button[data-testid="send-button"] - validated send-button semantics

TF-S11-R005 On x.com, the adapter requires a Grok route or surrounding Grok chat fingerprint and **MUST NOT** bind to ordinary X compose boxes. Placeholder-only selectors receive low confidence.

11.6.6 Perplexity

Adapter: perplexity-web Hosts: perplexity.ai and www.perplexity.ai Editor: Lexical/contenteditable Composer candidates: - #ask-input[contenteditable="true"] - #ask-input[role="textbox"] - div[role="textbox"][contenteditable="true"] - div[contenteditable="true"][data-lexical-editor="true"]

Submit methods: - Enter after composer verification; - button[aria-label="Submit"]; - button[aria-label="Send"]; - button[type="submit"].

Attachment candidates: - button[aria-label="Add files or tools"] - button[aria-label*="Attach"] - input[type="file"][multiple]

11.6.7 Google AI Studio

Adapter: google-aistudio-web Host: aistudio.google.com Editor: textarea Composer candidates: - textarea.textarea[placeholder="Start typing a prompt"] - textarea.textarea[aria-label="Enter a prompt"] - .prompt-box-container textarea.textarea

TF-S11-R006 The repository delegates part of submission and attachment behavior to legacy chatInputHandler code. The port **MUST** trace and move that behavior into the descriptor and shared drivers. Until verified postconditions exist, text insertion may be healthy while form submission or attachment remains DEGRADED.

11.6.8 OpenRouter

Adapter: openrouter-web Host: openrouter.ai Editor: textarea/contenteditable Composer candidates: - textarea[data-testid="composer-input"] - textarea[placeholder="Start a new message..."] - validated contenteditable

Submit candidates: - button[data-testid="send-button"] - button[aria-label="Send message"] - button[aria-label="Send prompt"]

Attachment candidates: - button[aria-label="Add attachment"] - button[aria-label="Attach file"] - matching file input

Test attributes and aria semantics outrank Tailwind-composition selectors.

11.6.9 T3 Chat

Adapter: t3chat-web Host: t3.chat Editor: textarea/contenteditable Composer candidates: - textarea#chat-input - textarea[placeholder*="Type your message"] - textarea.resize-none - validated contenteditable

Submit candidates: - button[type="submit"] - button[aria-label*="Submit"] - button[aria-label*="Send"] - button.send-button

Attachment candidates: - button[aria-label="Add files"] - button[aria-label*="attach"] - input[type="file"]

Both textarea and contenteditable UI variants require fixtures.

11.6.10 Mistral Le Chat

Adapter: mistral-web Host: chat.mistral.ai Editor: textarea/ProseMirror Composer candidates: - textarea[name="message.text"] - textarea[placeholder="Ask Le Chat anything"] - div.ProseMirror[contenteditable="true"][data-placeholder="Ask Le Chat anything"] - div.ProseMirror[contenteditable="true"]

Submit candidates: - .ms-auto .flex.gap-2 button[type="submit"] - button.bg-state-primary

Attachment candidates: - button[data-testid="attach-file-button"] - button[aria-label="Add files"] - input[name="file-upload"] - input[type="file"][multiple]

The repository's button[aria-label="Dictation"] submission candidate is quarantined unless runtime evidence proves that it submits.

11.6.11 Kimi

Adapter: kimi-web Host: kimi.com Editor: Lexical/contenteditable/textarea Composer candidates: - .chat-input-editor[contenteditable="true"] - div[contenteditable="true"][data-lexical-editor="true"] - .chat-input-editor - textarea[placeholder*="Ask"]

Submit candidates: - .send-button-container:not(.disabled) .send-button - button[aria-label*="Send"] - .send-button

Attachment candidates: - .attachment-button - label.attachment-button - input[type="file"]

11.6.12 Z.ai

Adapter: z-web Hosts: z.ai and chat.z.ai Editor: #chat-input Composer: - #chat-input

Submit: - #send-message-button - #send-message-button[type="submit"]

Attachment candidates: - button[aria-label*="More"] - a multiple file input matching accepted document/image/code types

A long position-dependent file-preview selector from the source is observation-only and low confidence. File name, accessible preview semantics, upload state, or learned fingerprints determine attachment success.

11.6.13 Qwen Chat

Adapter: qwen-web Hosts: qwen.ai and chat.qwen.ai Editor: textarea Composer candidates: - textarea.message-input-textarea - #chat-input - textarea.chat-input

Submit candidates: - button.omni-button-content-btn - div.message-input-right-button-send button - div.chat-prompt-send-button button - #send-message-button

Attachment candidates: - div.mode-select .ant-dropdown-trigger - div.mode-select-open - button.chat-prompt-upload-group-btn - input#filesUpload - input[type="file"][multiple]

The implementation includes fixtures for both the refreshed message-input UI and earlier prompt-input variants.

11.7 GenericAdapter and configured providers

GenericAdapter is a scored discovery engine, not “first editable element, first form.”

Composer candidates are collected from text inputs, textareas, contenteditable roots, role=textbox, ProseMirror, Lexical, Quill, and user-supplied profiles. Candidates are rejected when detached, hidden, disabled, readonly, inert, zero-sized, covered by an unrelated modal, inside login/search/navigation regions, or inside ScriptCat/another extension UI.

Positive signals include stable configured selectors, textbox semantics, visibility, known editor markers, lower-main-content position, nearby enabled send controls, composer-like containers, prompt/message/ask/chat names, successful fingerprints, and explicit user selection.

Negative signals include search/login/comment/filter semantics, disabled state, small single-line fields without multiline semantics, ambiguous duplicates, generated-class-only identity, and extension-owned UI.

`TF-S11-R007` The winning candidate **MUST** exceed a threshold and lead the runner-up by a minimum margin. Otherwise the adapter enters AWAITING_USER_SELECTION.

`TF-S11-R008` Send controls are searched first within the owning form and nearest composer container. GenericAdapter **MUST NOT** click an arbitrary last button.

A user may store a custom provider profile containing host/path matchers, selector candidates, drivers, completion rules, and output selectors. Such a profile participates in the normal registry, versioning, sharing, and capability system.

11.8 Output observation and completion

Composer control and output observation are separate adapter facets.

Each provider supplies or learns:

- conversation root;
- user-message roots;
- assistant-message roots and content;
- stop/cancel controls;
- generation indicators;
- message identity;
- code-block boundaries;
- provider-specific completion state.

The observer snapshots existing messages before submission, waits for a new user turn or generation transition, scopes a MutationObserver to the new assistant turn or conversation subtree, emits partial stream events, parses complete command envelopes, and returns final text, commands, artifacts, and evidence.

Completion requires:

- provider completed state or no active generation/stop control;
- no relevant text mutation for the configured quiescence period;
- a non-empty assistant message;
- no pending attachment or tool-result region;
- the same message root and document epoch throughout finalization.

Stable text alone is not completion.

11.9 Adapter configuration and frontend-change recovery

Effective configuration precedence, lowest to highest:

1. built-in repository-derived defaults;
2. signed release adapter pack;
3. signed remote hotfix pack;
4. locally learned candidates with successful postconditions;

5. user-defined provider profile;
6. operation-scoped manual selection.

Remote configuration is fetched once through the extension background, schema-validated, signature-checked when applicable, cached by adapter ID and revision, merged only into allowed fields, invalidated by update events, and rolled back to built-in or last-known-good state on failure.

```
interface AdapterPack {
  protocol: "scriptcat-adapters/1";
  packId: string;
  revision: number;
  issuedAt: string;
  expiresAt: string | null;
  minExtensionVersion: string;
  adapters: Record<string, SiteAdapterDescriptorPatch>;
  testVectors: AdapterTestVector[];
  rollbackRevision?: number;
  signer: string;
  signature: string;
}
```

TF-S11-R009 An adapter pack activates only after schema validation, compatibility checks, signature verification under the configured trust policy, fixture tests, and dry-run health checks. Users **MAY** authorize unsigned local packs and additional signing keys.

Health states:

HEALTHY All required capabilities pass and recent actions satisfy postconditions.

DEGRADED One or more advertised capabilities fail or fall below confidence threshold.

DISCOVERING Known candidates failed and semantic/local discovery is evaluating alternatives.

VALIDATING_CANDIDATE A candidate is tested through non-destructive probes and, when authorized, a reversible canary.

HOTFIXED A newer configuration passes and becomes active.

ROLLED_BACK The latest configuration failed and last-known-good is restored.

AWAITING_USER_SELECTION Automatic discovery is ambiguous.

QUARANTINED Repeated failures make unattended dispatch unreliable.

TF-S11-R010 Health is capability-scoped. A site may remain healthy for insertion while attachment is degraded. The user **MAY** override quarantine; the choice and resulting evidence are recorded.

Element fingerprints include tag, namespace, role, accessible-name hash, stable attributes, editor markers, ancestor path, send-control relation, bounding-box class, owning-form signature, frame/shadow path, class-token stability, and a text-free subtree shape hash. A selector that still matches but violates its fingerprint is rejected.

Per-adapter storage:

```
adapters/{adapterId}/active.json
adapters/{adapterId}/revisions/{revision}.json
adapters/{adapterId}/health/{timestamp}.json
adapters/{adapterId}/learned/{fingerprint}.json
adapters/{adapterId}/fixtures/{uiVersion}/
adapters/{adapterId}/traces/{operationId}.jsonl
```

During an active operation, a lost or replaced composer causes the adapter to pause without losing operationId, increment generation, record epoch, re-resolve provider and conversation, test known selectors, apply valid hotfixes, run discovery, request selection if needed, observe whether insertion or submission already occurred, and resume only from the observed effect state.

TF-S11-R011 A lost acknowledgement after submission remains UNKNOWN_OUTCOME until the user message or assistant response is observed. The prompt **MUST NOT** be blindly resubmitted.

TF-S11-R012 Default remote diagnostics exclude prompts, responses, file contents, credentials, and full DOM. They **MAY** include adapter ID, selector candidate ID, failure code, UI fingerprint hashes, extension version, and postcondition results. Broader sharing is user-configurable.

11.10 Adapter implementation layout

```
extension/src/content/provider-adapters/
  core/
    adapter-types.ts
    base-adapter.ts
    registry.ts
    context.ts
    event-bridge.ts
    selector-engine.ts
    selector-validation.ts
    element-fingerprint.ts
    postconditions.ts
    health-state-machine.ts
    config-manager.ts
    adapter-pack.ts
  drivers/
    native-value-driver.ts
    contenteditable-driver.ts
    prosemirror-driver.ts
    lexical-driver.ts
    quill-driver.ts
    submit-driver.ts
    attachment-driver.ts
    trusted-input-driver.ts
  adapters/
    chatgpt.adapter.ts
    gemini.adapter.ts
    github-copilot.adapter.ts
    deepseek.adapter.ts
    grok.adapter.ts
    perplexity.adapter.ts
    google-aistudio.adapter.ts
    openrouter.adapter.ts
    t3chat.adapter.ts
    mistral.adapter.ts
    kimi.adapter.ts
    z.adapter.ts
    qwen.adapter.ts
    claude.adapter.ts
    generic.adapter.ts
    configured.adapter.ts
  configs/
    built-in/
    remote/
    local/
    learned/
  output/
    stream-observer.ts
    message-extractor.ts
    completion-detector.ts
    command-parser-bridge.ts
  recovery/
    hotfix-manager.ts
    rollback-manager.ts
    manual-selection.ts
    quarantine.ts
  fixtures/
  tests/
```

`TF-S11-R013` The source adapter code **SHOULD** be ported with license attribution and normalized into descriptors and shared drivers. Thirteen copies of URL polling, observer setup, event setup, file transfer, button insertion, and cleanup **MUST NOT** be retained when one tested shared implementation suffices.

§12 FIRST RELEASE

Agent tab manager

> **Purpose.** Defines protected agent tabs, pool scheduling, restoration, and the difference between extension and native enforcement.

12.1 States

requested creating loading binding ready busy waiting returning idle quarantined restoring released closed

12.2 Spawn request

```
CAT.agent.tabs.spawn({
  provider: "chatgpt-web",
  conversation: "new|reuse",
  prompt: "…",
  active: false,
  workspace: "Transductive Agents",
  group: "AGENT POOL",
  pinned: true,
  protection: "hard",
  discardPolicy: "never-while-busy",
  restorePolicy: "always",
  capabilityProfileId: "…"
})
```

12.3 Protection modes

none Normal browser behavior.

visual Badge, group, title/status indicator, and management controls.

input-shield Human pointer and keyboard input are intercepted while automation channels remain available.

hard input-shield plus ownership checks, navigation policy, close restoration, anti-discard while busy, and managed workspace placement.

native-lock Optional Chromium mode that rejects ordinary close, drag-out, manual navigation, and human editing until released.

12.4 Close and restoration

In the extension build, chrome.tabs.onRemoved detects closure. The binding enters restoring, a replacement tab opens with the same agentId and provider/conversation target, adapter health is re-established, protection is reapplied, and the operation resumes from observed state or becomes unknown_outcome.

TF-S12-R001 In the Chromium-derived build, protected tabs **MAY** carry a native close disposition. Force release remains available through explicit management UI or command.

12.5 Pool scheduling

The manager supports warm-agent minima/maxima per provider, per-project pools, provider-account affinity, serial or parallel dispatch, conversation-reuse policy, idle timeout, health probes, and quarantine.

Concurrency is limited by user configuration, provider policy, and system resources, not by an arbitrary built-in reluctance to open tabs.

§13

FIRST RELEASE

Proposed cat API extensions

> **Purpose.** Lists the typed CAT compatibility APIs that expose privileged extension capabilities to userscripts.

13.1 Router

```
@grant CAT.router
```

```
CAT.router.registerEndpoint(descriptor)
CAT.router.request(method, payload, options)
CAT.router.respond(operationId, payload)
CAT.router.emit(event, payload)
CAT.router.subscribe(filter, handler)
CAT.router.cancel(operationId)
CAT.router.status(operationId)
```

13.2 Agent tabs

```
@grant CAT.agent.tabs
```

```
CAT.agent.tabs.spawn(options)
CAT.agent.tabs.attach(tabIdOrInstanceId, options)
CAT.agent.tabs.prompt(agentId, prompt, options)
CAT.agent.tabs.protect(agentId, mode)
CAT.agent.tabs.release(agentId)
CAT.agent.tabs.list(filter)
CAT.agent.tabs.get(agentId)
CAT.agent.tabs.cancel(agentId, operationId)
```

13.3 Provider adapters

```
@grant CAT.agent.adapters
```

```
CAT.agent.adapters.getActive()
CAT.agent.adapters.getForUrl(url)
CAT.agent.adapters.list()
CAT.agent.adapters.health(adapterId?)
CAT.agent.adapters.reload(adapterId?)
CAT.agent.adapters.applyConfig(adapterId, config, options?)
CAT.agent.adapters.discover(purpose, options?)
CAT.agent.adapters.selectElement(purpose, options?)
CAT.agent.adapters.exportDiagnostics(adapterId, options?)
```

13.4 Script mutation

```
@grant CAT.script.mutation
```

```
CAT.script.mutation.installSource(source, options)
CAT.script.mutation.updateSource(scriptId, source, options)
CAT.script.mutation.getSource(scriptId, version?)
CAT.script.mutation.listVersions(scriptId)
CAT.script.mutation.activate(scriptId, version)
CAT.script.mutation.rollback(scriptId, version?)
CAT.script.mutation.run(scriptId, target, options)
```

13.5 Daemon bridge

```
@grant CAT.daemon
```

```
CAT.daemon.connect(profile?)
CAT.daemon.status()
CAT.daemon.listCommands()
CAT.daemon.invoke(command, arguments, options)
CAT.daemon.cancel(operationId)
CAT.daemon.subscribe(event, handler)
```

13.6 Identity

```
@grant CAT.identity
```

```
CAT.identity.getNodeIdentity()
CAT.identity.getBrowserIdentity()
CAT.identity.createIdentity(options)
CAT.identity.sign(data, options)
CAT.identity.verify(data, signature, identity)
CAT.identity.pairDaemon(options)
CAT.identity.pairPeer(invitation)
CAT.identity.rotate(options)
CAT.identity.export(options)
CAT.identity.import(bundle, options)
```

13.7 NetworkScripts

```
@grant CAT.network
```

```
CAT.network.joinProject(projectDescriptor)
CAT.network.leaveProject(projectId)
CAT.network.listPeers(filter)
CAT.network.expose(endpointDescriptor)
CAT.network.unexpose(endpointId)
CAT.network.invoke(peerId, endpointId, method, payload, options)
CAT.network.delegate(task, options)
CAT.network.subscribe(topicOrFilter, handler)
CAT.network.publish(topic, payload, options)
CAT.network.getPermissionRequest(requestId)
CAT.network.answerPermissionRequest(requestId, decision)
```

Every API evaluates the selected capability profile and returns operation-scoped evidence.

§14

FIRST RELEASE

NetworkScript model

> **Purpose.** Defines remotely callable NetworkScripts and revocable tab or script endpoints without hiding broad user-configured authority.

A NetworkScript is a ScriptCat userscript with one or more remotely callable endpoint descriptors.

Example metadata:

@network transductive/1 @expose research.answer @expose tab.read @capability provider.chat @capability tab.dom @capability daemon.command

Metadata declares intent. The active capability profile determines actual exposure.

Endpoint descriptor:

```
{
  "endpointId": "research.answer",
  "scriptId": "…",
  "version": "1.0.0",
  "methods": {
    "run": {
      "inputSchema": {},
      "outputSchema": {},
      "streaming": true,
      "idempotency": "operation"
    }
  },
  "scope": {
    "projects": ["project_…"],
    "peers": ["peer_…"]
  },
  "execution": {
    "target": "agent-pool",
    "provider": "chatgpt-web",
    "timeoutMs": 900000
  }
}
```

A remote invocation is bound to peer identity, project identity, endpoint/version, operationId, capability profile, method schema, resource scope, expiry, and signature.

A shared tab is a revocable endpoint:

```
{
  "endpointId": "tab:…",
  "tabInstanceId": "…",
  "allowedMethods": ["read", "screenshot", "execute", "fill", "click"],
  "selectorScope": "*",
  "urlScope": "*",
  "expiresAt": null
}
```

TF-S14-R001 Users **MAY** intentionally define broad endpoints and wildcard scopes. The platform records and enforces the explicit policy rather than imposing a hidden ceiling.

PART III

Node, network, authority, and recovery

§15 FIRST RELEASE

Torsion node daemon

> **Purpose.** Defines the Rust Torsion Node, its transports, services, command registry, invocation lifecycle, and service installation.

15.1 Implementation

TF-S15-R001 The production daemon **SHOULD** be a Rust workspace producing one signed executable per platform. Rust is preferred for resident footprint, strong protocol typing, service integration, and mature libp2p support. A Node.js reference client **MAY** remain for interoperability tests.

15.2 Local transport preference

1. authenticated native messaging;
2. authenticated loopback WebSocket;
3. Unix domain socket or Windows named pipe through the extension host;
4. browser-fork native IPC.

Loopback binds to 127.0.0.1 and ::1 unless the user explicitly configures another address.

15.3 Services

identity policy command-registry process-runner filesystem artifact-store operation-journal scriptcat-bridge libp2p cloud-control updater service-manager diagnostics

15.4 Command registry

Commands are configuration-defined rather than limited to a fixed list.

```
{
  "command": "shell.exec",
  "effect": "allow",
  "runner": "process",
  "executable": "*",
  "arguments": "*",
  "workingDirectories": ["*"],
  "environment": "*",
  "stdin": true,
  "network": true,
  "filesystem": {
    "read": ["*"],
    "write": ["*"]
  },
  "peers": ["*"],
  "projects": ["*"],
  "approval": "none"
}
```

A cautious profile may be narrow; an owner profile may be unrestricted.

15.5 Invocation lifecycle

1. receive signed request;
2. verify identity, expiry, sequence, and replay constraints;
3. resolve capability profile;
4. request a user decision when configured;
5. journal the operation;
6. start the runner;
7. stream stdout, stderr, and structured events;
8. collect exit state and artifacts;

9. sign the result;
10. return it to ScriptCat or the remote peer;
11. retain journal data according to policy.

15.6 Service installation

Windows: per-user service or scheduled service by default; optional system service. Linux: systemd user service by default; optional system service. macOS: LaunchAgent by default; optional LaunchDaemon.

Pairing occurs after installation.

§16

FIRST RELEASE

Identity, pairing, and message protection

> **Purpose.** Defines identity hierarchy, pairing, key handling, signatures, encryption, and replay protection.

16.1 Identity hierarchy

Root user identity Signs device identities.

Node identity Identifies the Torsion Node and libp2p peer.

Browser identity Identifies one ScriptCat/browser profile.

Project membership credential Binds an identity to a project and role.

Endpoint credential Delegates a defined capability subset to a script, provider, daemon command, or tab.

16.2 Recommended algorithms

- Ed25519 signatures where supported;
- ECDSA P-256 as WebCrypto compatibility fallback;
- X25519 or ECDH P-256 for key agreement;
- HKDF-SHA-256 for derivation;
- AES-GCM or ChaCha20-Poly1305 for application envelopes;
- libp2p Noise or TLS 1.3 for peer transport.

TF-S16-R001 Browser private keys **SHOULD** be non-extractable CryptoKeys where possible. Daemon keys **SHOULD** use the OS key store or encrypted local storage. Exportable mode remains available when the user enables it.

16.3 Pairing

1. Browser and daemon create identities.
2. The daemon presents a one-time challenge over loopback.
3. The browser signs the challenge.
4. The daemon signs the transcript.
5. The user confirms matching short authentication strings, unless an explicitly configured automated local mode applies.
6. Both sides pin identities and derive a session key.
7. Subsequent connections use mutual authentication.

TF-S16-R002 TOFU-only pairing **MAY** be selected for automated local VM deployments.

16.4 Signed messages

Signed messages include protocol, messageId, operationId, sender, recipient, timestamp, expiry, monotonic sequence, nonce, payload digest, capability profile, and signature.

Receivers maintain replay windows and reject duplicate messageId/sequence combinations.

§17

FIRST RELEASE

Peer networking and cloud control

> **Purpose.** Defines peer connectivity, project tasks, domain responsibilities, and the optional Cloudflare coordination plane.

17.1 libp2p stack

Torsion Node uses supported combinations of TCP, QUIC, WebSocket, WebTransport, or WebRTC; Noise or TLS; Yamux; Identify; Ping; AutoNAT; Circuit Relay v2; DCUtR; optional Kademlia DHT; optional GossipSub; and request/response protocols.

Mplex is retained only for required interoperability.

Protocol IDs:

```
/transductive/hello/1
/transductive/capabilities/1
/transductive/invoke/1
/transductive/result/1
/transductive/artifact/1
/transductive/presence/1
/transductive/project/1
```

Connectivity strategy:

1. connect to configured bootstrap peers;
2. obtain relay reservations when not publicly dialable;
3. advertise relay addresses;
4. establish relayed sessions;
5. attempt DCUtR direct upgrade;
6. retain relay fallback;
7. use authenticated Cloudflare rendezvous when configured;
8. do not require inbound port forwarding for ordinary participation.

17.2 Project task

```
{
  "taskId": "…",
  "projectId": "…",
  "issuer": "…",
  "requirements": {
    "endpoint": "research.answer",
    "provider": ["chatgpt-web", "gemini-web"],
    "capabilities": ["provider.chat", "tab.dom"]
  },
  "prompt": "…",
  "artifacts": [],
  "deadline": "…",
  "resultSchema": {},
  "replication": 3,
  "aggregation": "project-defined"
}
```

The local node matches requirements to exposed capabilities, obtains configured consent, dispatches to agent tabs, and returns signed results.

17.3 Domain responsibilities

torsionfield.de Node downloads, signed release and adapter-pack manifests, bootstrap peers, relay registry, pairing invitations, node administration, and optional Tunnel enrollment.

transductive.org Participant identity, collaboration, general NetworkScript projects, project rooms, and endpoint discovery.

transductive.science Scientific project definitions, task queues, prompt packages, evidence requirements, result assembly, and publication provenance.

17.4 Cloudflare components

Worker Validates HTTPS/WebSocket requests, identity, origin, and routing.

NodePresence Durable Object Coordinates authenticated node sessions.

ProjectRoom Durable Object Coordinates participants, offers, acknowledgements, and result announcements.

TaskRoom Durable Object Owns one task's leases, replicas, deadlines, and result references.

R2 Stores signed release packages and optional encrypted large artifacts.

D1 or Durable Object SQLite Stores project metadata and audit records.

Queues Handles control-plane fanout and result-processing jobs.

TF-S17-R001 Durable Object WebSocket Hibernation **SHOULD** maintain idle sessions without pinning active compute.

TF-S17-R002 Cloudflare Tunnel is optional. It **MAY** expose a user-authorized local endpoint under a stable hostname, with or without Access. It is not the mandatory peer data path.

§18

FIRST RELEASE

Permission and capability model

> **Purpose.** Defines the permission model, including narrow profiles, explicit unrestricted profiles, and signed delegation.

18.1 Decision values

deny ask allow_once allow_operation allow_session allow_project allow_peer allow_endpoint always_allow

18.2 Match dimensions

subject Script, browser profile, local user, peer, project, site, provider, or daemon module.

action Read, write, execute, install, mutate, spawn, navigate, prompt, submit, share, invoke, command, filesystem, network, or identity.

resource Tab, selector, URL, script, file path, command, process, provider account, or endpoint.

context Local/remote, interactive/unattended, time, network, project, device, and operation ancestry.

Rules are evaluated from most specific to least specific. The UI shows the effective rule and why it matched.

18.3 Unrestricted profiles

TF-S18-R001 The system **MUST** support an explicit wildcard profile, for example:

```
{
  "effect": "allow",
  "subjects": ["script:*", "peer:*", "project:*"],
  "actions": ["*"],
  "resources": ["*"],
  "duration": "persistent"
}
```

This profile may authorize unattended remote commands, provider prompts, script installation, filesystem/process work, and peer invocation. The active mode must remain visible.

18.4 Delegation

Capabilities may be delegated to one operation, agent, script version, peer, project, endpoint, signed invitation, or a wildcard group. Delegations are signed and revocable.

§19

FIRST RELEASE

State, recovery, failure, and observability

> **Purpose.** Defines persistent state, restart reconciliation, failure envelopes, traces, and user-visible diagnostic surfaces.

19.1 ScriptCat OPFS

```
agents/workspace/transductive/
  identities/
  operations/
  scripts/
  mutations/
  adapters/
  traces/
  screenshots/
  provider-fixtures/
  network/
  cache/
```

19.2 Daemon state

SQLite tables:

operations attempts messages peers projects permissions endpoints artifacts agent_bindings audit_events

Content-addressed artifacts:

artifacts/sha256/ab/cd/<digest>

19.3 Extension restart

On restart, the extension:

1. restores browserInstanceId;
2. enumerates tabs and reconciles tabInstanceId markers;
3. restores agent bindings;
4. reloads adapter last-known-good revisions and health;
5. queries the daemon for in-flight work;
6. reattaches provider observers;
7. observes effects before retries;
8. recreates protected tabs when configured;
9. reports unresolved work as unknown_outcome.

19.4 Daemon restart

The daemon restores node identity, peer/project sessions, operation journal, ScriptCat connection, presence, and operation reconciliation.

19.5 Failure envelope

Every failure includes:

```
{
  "code": "…",
  "message": "…",
  "operationId": "op_…",
  "attemptId": "attempt_…",
  "component": "…",
  "recoverability": "…",
  "effectState": "…",
  "observations": [],
  "traceRef": "artifact_…",
  "recommendedNextAction": "…"
}
```

Required error families include:

PROVIDER_ADAPTER_UNHEALTHY COMPOSER_NOT_FOUND SUBMIT_NOT_CONFIRMED STREAM_TIMEOUT TARGET_TAB_GONE DOCUMENT_EPOCH_CHANGED SELECTOR_NOT_UNIQUE STALE_GENERATION PERMISSION_DENIED PERMISSION_PENDING DAEMON_UNAVAILABLE PEER_UNREACHABLE ENDPOINT_NOT_EXPOSED SCRIPT_PARSE_FAILED SCRIPT_BOOT_FAILED MUTATION_ROLLED_BACK UNKNOWN_OUTCOME

Failures are routable to the originating coding tab for repair or additional context.

19.6 Event schema

```
{
  "time": "…",
  "level": "debug|info|warn|error",
  "component": "…",
  "event": "…",
  "operationId": "…",
  "attemptId": "…",
  "agentId": "…",
  "tabInstanceId": "…",
  "generation": 1,
  "epoch": "…",
  "data": {}
}
```

User-visible views:

Agent Workspace Live agents, provider, task, health, protection, generation, and release controls.

Operation Inspector Causal event timeline across origin, Router, adapter, target, daemon, peer, and project.

Permission Inspector Request, matching rule, effective authority, and rule-creation controls.

Script Mutation History Versions, hashes, activation, rollback, origin, and tests.

Adapter Inspector Active descriptor revision, capability health, selector evidence, fingerprints, hotfixes, learned candidates, rollback, and quarantine.

Network Dashboard Connections, path type, exposed endpoints, projects, and remote operations.

§20 OPTIONAL NATIVE

Optional chromium-derived integration

> **Purpose.** Defines optional Chromium-native guarantees while keeping the extension and daemon as the complete first implementation.

The extension and daemon deliver the complete functional architecture. A Chromium-derived build adds guarantees unavailable to ordinary extension APIs.

AgentWorkspaceService Browser-owned agent/workspace registry.

ProtectedTabController Native close, drag, focus, input, navigation, and discard policy.

TransductiveIpcService Authenticated daemon IPC bound to browser-profile identity.

AutomationInputService Trusted input without a competing user-visible debugger attachment.

ProviderAdapterService Native adapter lifecycle and DOM bridge.

OperationTraceService Browser-process trace correlation by operationId.

NetworkScriptService Direct bridge from ScriptCat APIs to browser/daemon routing.

A native-lock tab cannot be closed through ordinary UI, detached, manually navigated outside policy, edited by human input while locked, or discarded while owning active work. It remains force-releasable through explicit management controls.

TF-S20-R001 Any nonstandard web-platform behavior **MUST** be behind an explicit build or runtime feature flag. Undocumented flags **MUST NOT** be required where an extension or daemon bridge provides the capability.

PART IV

Implementation, productization, and evolution

§21

FIRST RELEASE

Implementation work packet

> **Purpose.** Turns the architecture into one integrated engineering work packet rather than a sequence of disconnected demonstrations.

The implementation is delivered as one integrated work packet. Internal scaffolding is allowed, but the delivered artifact must represent the final architecture rather than staged mock behavior.

`TF-S21-R001` The engineering pass **MUST**:

1. fork the current ScriptCat source;
2. add typed CAT APIs and grants;
3. implement Router Core and operation journal;
4. implement Mother Script and command parsers;
5. implement transactional core mutation;
6. implement native userscript install/update/run/rollback;
7. implement Target Probe and selection mode;
8. port and normalize all repository-derived adapters;
9. implement an independently verified Claude adapter or clearly mark it unavailable rather than fabricating selectors;
10. implement shared editor, submission, attachment, output, health, hotfix, and rollback modules;
11. implement GenericAdapter and ConfiguredAdapter;
12. implement Agent Tab Manager and protection/restoration;
13. implement error/result feedback into coding composers;
14. implement Torsion Node and local pairing;
15. implement signed identities and envelopes;
16. implement libp2p protocols;
17. implement optional Cloudflare services;
18. implement NetworkScript exposure and invocation;
19. implement permission UI, including unrestricted profiles;
20. implement crash and frontend-change recovery;
21. produce installers and signed packages;
22. execute the complete acceptance matrix.

§22

FIRST RELEASE

Acceptance criteria

> **Purpose.** Defines the acceptance matrix for mutation, routing, adapters, recovery, the daemon, and the network.

22.1 Mother mutation

- legacy and canonical mutation tags are parsed once;
- a valid candidate is stored, activated, boot-leased, and executed;
- a broken candidate rolls back automatically;
- DOM rehydration does not repeat mutation.

22.2 Userscript installation

- a complete .user.js payload installs with exact metadata and grants;
- the script runs in a selected target tab;
- a second version updates the same script identity;
- rollback restores the prior source;
- the Blob installer fallback works.

22.3 Target selection

- the exact selection macro starts an operation-scoped workflow;
- the user selects a tab and eligible element;
- the selector bundle returns only to the origin;
- insertion uses the active provider adapter;
- optional auto-submit obeys endpoint policy.

22.4 Routing and recovery

- at least 20 simultaneous operations across at least 20 tabs complete without cross-talk;
- stale generation and epoch responses are rejected;
- duplicate transport requests do not duplicate effects;
- lost non-idempotent acknowledgements become unknown_outcome and are observed;
- service-worker suspension preserves operation identity;
- browser restart reconciles tabs and bindings;
- daemon restart restores journal and presence;
- network interruption reconnects without duplicated effects.

22.5 Agent pool

- multiple configured provider tabs can be opened and scheduled;
- group/workspace and protection policies are applied;
- input-shield/hard modes block human input while automation remains functional;
- closing a busy stock-browser agent causes restoration;
- native-lock prevents ordinary close in the Chromium build;
- concurrency is governed by configured limits and resources.

22.6 Provider adapters

For every repository-derived provider:

- registry matching selects the correct host/path and no unrelated site;

- composer resolution succeeds from built-in or valid updated descriptors;
- replace and append modes work;
- inserted text is verified;
- submission produces an observed user-message or generation transition;
- assistant streaming is captured without whole-body rescanning;
- final output and command ranges are extracted;
- command envelopes execute once;
- SPA navigation rebinds the adapter;
- composer replacement is recovered;
- stale candidates fail fingerprint validation;
- a valid signed hotfix activates without reinstall;
- an invalid hotfix rolls back;
- ambiguous discovery asks for selection;
- user-selected profiles persist and are reusable;
- errors return to the origin.

Provider-specific cases:

ChatGPT ProseMirror insertion, send-button variants, multiple-file DataTransfer, and drag/drop fallback.

Gemini Quill-root rebinding and remote-config invalidation/merge.

GitHub Copilot github.com route isolation and semantic selectors outranking CSS modules.

DeepSeek Generated hash selectors cannot outrank stable semantic candidates.

Grok Ordinary x.com compose boxes never activate GrokAdapter.

Perplexity Lexical insertion and Enter-first verified submission.

Google AI Studio Text insertion health is independent from still-unverified legacy submit/attachment behavior.

OpenRouter data-testid and aria semantics outrank Tailwind compositions.

T3 Chat Textarea and contenteditable variants are fixture-tested.

Mistral Textarea and ProseMirror variants work; Dictation is rejected as submit without evidence.

Kimi Lexical/contenteditable insertion and send-state validation.

Z.ai Primary ID paths work; brittle preview failure does not invalidate text submission.

Qwen Refreshed and earlier composer/attachment UI variants work.

Generic The correct chat editor is selected on a page containing search, login, comment, and chat fields; low confidence causes no action; manual selection creates a reusable profile.

Claude The adapter passes the same contract and fixtures once independently implemented. Until then, provider selection fails explicitly instead of falling through to an unrelated adapter.

22.7 Adapter recovery

- built-in selector failure enters discovery;
- valid remote hotfix succeeds;
- schema or signature failure is rejected;
- last-known-good rollback succeeds;
- a local learned candidate can recover a capability;
- document epoch change during streaming is handled;
- tab reload and frontend A/B variants rebind;
- an operation resumes without duplicate prompt submission;
- repeated failure causes quarantine;
- explicit user override is honored and traced.

22.8 Daemon and network

- extension and daemon pair cryptographically;
- reconnect survives browser and daemon restart;
- a narrow profile permits only its configured command;
- an unrestricted profile can invoke an authorized arbitrary process/filesystem action;
- stdout, stderr, exit state, and artifacts stream back;
- two nodes discover and connect directly or through relay;
- one node exposes and another invokes a NetworkScript;
- policy decisions and revocation are honored;
- results are signed and correlated;
- replicated project results carry node, provider, prompt, operation, and artifact provenance.

§23

FIRST RELEASE

Implementation rules

> **Purpose.** Collects the non-negotiable implementation rules that prevent silent narrowing, brittle routing, and false success.

- Reuse existing CAT.agent.* functions when they already satisfy the requirement.
- Add privileged extension APIs instead of pretending background userscripts own Chrome internals.
- Preserve legacy command compatibility.
- Use operation identity for every effect.
- Never use one global storage key as a runtime bus.
- Never bind durable state only to tabId.
- Separate provider matching, DOM location, action driver, output observation, and recovery.
- Treat selectors as candidates with evidence, not timeless facts.
- Prefer stable semantic selectors over generated classes and deep structure.
- Do not claim a stock extension can make a tab literally uncloseable.
- Do not silently narrow an explicitly unrestricted policy.
- Do not centralize participant model credentials.
- Do not make Cloudflare the only data path.
- Do not blindly retry an uncertain effect.
- Do not report action success without a postcondition.
- Preserve user choice over local, remote, interactive, and unattended authority.
- Keep provider registration extensible at runtime rather than imposing a fixed allowlist ceiling.

§24

FIRST RELEASE

Product completeness and implementation closure

> **Purpose.** States what remains missing before the specification is implementation-closed and production-ready.

TF-S24-R001 24.1 Completeness assessment This specification is architecturally complete enough to define the intended product, its trust boundaries, its principal runtime components, and its major execution flows. It is not yet implementation-complete in the stronger sense that an engineering team could build every component without making additional product, protocol, storage, user-interface, packaging, and operational decisions. For planning purposes, the current state **SHOULD** be treated as approximately complete at three different levels: - product and architectural intent: substantially complete; - build specification: partially complete and still requiring closure artifacts; - production and operational specification: incomplete until deployment, update, support, security, and service procedures are fixed. These percentages are engineering judgement rather than measured facts. The important distinction is that architecture answers what the system is, while implementation closure fixes exactly how each component is patched, configured, tested, shipped, upgraded, recovered, and operated. 24.2 Required implementation-closure artifacts Before the specification may be declared final for production, the repository **MUST** contain: 1. an exact ScriptCat baseline commit, fork branch, and patch manifest identifying every modified file, exported symbol, database migration, permission, build flag, and user-interface route; 2. complete JSON Schemas plus generated TypeScript and Rust types for every CAT API, Router message, operation state, adapter pack, prompt commit, incorporation record, network message, first-party website message, and daemon command; 3. extension UI wireframes and state models for the popup, side panel, script list, editor, visual recorder, workflow builder, Agent Workspace, NetworkScript Manager, Adapter Inspector, Permission Inspector, Incorporation Dashboard, update center, and onboarding flow; 4. a browser-permission matrix covering required, optional, requested-at-use, host, native-messaging, downloads, debugger, scripting, tab, storage, offscreen, and browser-specific permissions; 5. storage schemas, migration rules, retention policy, backup/export, corruption recovery, and version compatibility for extension storage, OPFS, daemon SQLite, local artifacts, and cloud control-plane state; 6. a security and abuse model covering secret storage, identity recovery, key rotation, lost devices, malicious scripts, malicious peers, compromised upstream repositories, prompt-injection boundaries, supply-chain attacks, denial of service, and revocation; 7. provider fixtures, golden traces, replayable frontend snapshots, browser/version matrices, and automated canary accounts sufficient to test provider adapters without relying only on live manual inspection; 8. installers, signing keys, update channels, rollback channels, release manifests, browser-store submission plans, unattended update policy, and reproducible-build instructions; 9. import and migration specifications for ScriptCat, Tampermonkey, Violentmonkey, exported userscript archives, settings, values, update URLs, and compatibility modes; 10. the torsionfield.de registration, pairing, membership, device, node, project, and revocation protocols described in machine-readable form; 11. service-level objectives, privacy defaults, telemetry fields, support bundles, incident response, status reporting, and compatibility support windows; 12. contribution, plugin, adapter, capability-pack, and incorporated-extension SDK documentation; 13. an acceptance manifest that maps every normative requirement to one or more automated tests and release gates. A feature is specification-complete only when its contract, user surface, state model, storage, permissions, failure behavior, migration, observability, tests, rollback, documentation, and ownership are all defined. A prose description of desired behavior alone is not implementation closure. 24.3 Exact ScriptCat modification dossier The fork **MUST** include docs/scriptcat-patch-dossier.md and a machine-readable patch-dossier.json. Each patch entry contains: - upstream ScriptCat commit and file path; - upstream symbol or component; - reason for modification; - added or changed behavior; - public CAT API or internal service affected; - storage or migration impact; - permission impact; - UI impact; - tests and acceptance IDs; - upstream-conflict risk; - rebase strategy; - rollback strategy. No requirement may remain expressed only as “modify ScriptCat to support X.” The dossier **MUST** identify the concrete extension source boundary where X is implemented. 24.4 Product modes The final product has three progressively enabled modes: Browser Runtime The extension works as a superior userscript manager without an account or daemon. It provides userscript compatibility, import, editing, visual recording, AI-assisted generation, script diagnostics, self-healing selectors, local workflow execution, session snapshots, and provider adapters. Local Node Pairing the Torsion Node daemon adds filesystem and process tools, local models, MCP servers, artifact storage, durable background work, trusted input fallbacks, desktop automation bridges, and stronger recovery. Network Node Registration through torsionfield.de adds signed membership, rendezvous, relay discovery, projects, peer invocation, NetworkScript exposure, replicated work, capability exchange, and participation in transductive.science, transductive.art, and transductive.org. Users **MUST** gain substantial value in Browser Runtime mode. Registration and daemon installation unlock capabilities; they are not prerequisites for ordinary userscript management.

§25

STAGED TARGET

Extension incorporation program and killer features

> **Purpose.** Defines how useful extension capabilities may be incorporated without uncontrolled copying or architectural fragmentation.

TF-S25-R001 25.1 Incorporation principle The product **MUST** be able to absorb useful capabilities from external browser extensions without becoming a pile of copied repositories. “Incorporation” means one of four explicitly recorded strategies: 1. direct source integration when the license, architecture, provenance, and maintenance burden permit it; 2. clean behavioral reimplementation from documented behavior and black-box tests; 3. protocol, file-format, metadata, import/export, or API compatibility without source copying; 4. an optional isolated bridge or plugin that keeps the external component separable. Every incorporated capability **MUST** retain upstream attribution, license analysis, source commit or release identity, extracted capability contracts, transformation history, tests, and a future update path. Proprietary code and license-incompatible code **MUST NOT** be copied. Copyleft projects may be used only under a distribution strategy that satisfies their complete license obligations. Researching an extension does not authorize source incorporation. 25.2 Priority source projects and capability targets The initial incorporation registry **SHOULD** evaluate at least the following projects: Violentmonkey and Tampermonkey compatibility Target capabilities: mature userscript metadata and GM API compatibility, import/export, update checks, value migration, script search, per-site enablement, sync concepts, and low-friction migration. Violentmonkey is the preferred open-source compatibility reference. Tampermonkey behavior may be targeted through interoperability tests rather than current-source incorporation. Nanobrowser, AIPex, and BrowserOS Target capabilities: multi-agent side panels, conversation history, provider switching, bring-your-own-key and local-model routing, MCP tools, browser-agent task views, accessibility-tree and optimized-DOM representations, agent CLI concepts, and evaluation harnesses. Architecture and license determine whether each capability is reimplemented, bridged, or directly reused. MCPMonkey and Algonius Browser Target capabilities: typed MCP exposure, extension-to-native-host bridges, stdio/WebSocket transport, schema validation, browser tools, fine-grained tool permissions, and integration-test patterns. Automa Target capabilities: visual block workflows, scheduling, reusable blocks, workflow sharing, parameterized runs, standalone workflow export, and marketplace ergonomics. Its visual model **SHOULD** compile to the same typed operations used by scripts and agents rather than creating a second runtime. UI.Vision RPA, Skyvern, and Magnitude Target capabilities: visual recording, macro import/export, OCR and computer-vision fallback, visual assertions, screenshot-based element recovery, Playwright/Selenium interoperability, and resilient validation when DOM selectors are insufficient. SingleFile Target capabilities: reproducible Web Capsules containing page state, resources, selected scripts, prompt commits, adapter revision, traces, screenshots, and evidence sufficient to replay or audit an operation. Vimium and related keyboard-first extensions Target capabilities: universal command palette, link hints, keyboard navigation, site-specific mappings, operation search, and one-keystroke execution of scripts, workflows, and NetworkScripts. Tab Session Manager and session-management extensions Target capabilities: named workspace snapshots, automatic recovery, task-bound tab groups, agent-pool restoration, versioned session state, and shareable project workspaces. 25.3 Killer-feature portfolio The final extension **SHOULD** make the following capabilities first-class product features: One-click Agentify This Site From the page context menu or command palette, the user selects a goal. The extension records page structure and interaction evidence, opens a coding agent, generates a maintainable userscript plus a typed workflow and optional NetworkScript endpoint, executes tests, and installs the result with rollback. Record Once, Compile Three Ways A visual recorder produces three synchronized artifacts: a replayable workflow, readable userscript source, and a callable endpoint contract. Editing any representation regenerates the others through a shared intermediate operation graph. Self-Healing Userscripts Scripts may declare semantic targets and postconditions. When selectors drift, the runtime compares DOM, accessibility, visual, and historical fingerprints; validates alternatives through canaries; proposes or applies a signed repair; and preserves the original script identity and provenance. Universal Command Palette and Link Hints Scripts, workflows, agents, tabs, peers, projects, tools, and recent operations are searchable from one keyboard interface. Link-hint mode allows deterministic keyboard selection of visible elements and can feed the recorder or element selector. Time-Travel Workspaces The user can capture, name, diff, restore, and share a browser workspace containing tabs, groups, selected form state, installed scripts, active versions, agent bindings, operations, and local artifacts. Sensitive fields are excluded by policy. Reproducible Web Capsules A capsule packages sufficient page and runtime context to reproduce a bug, demonstrate a workflow, publish research evidence, or transfer an automation. Capsules are content-addressed, inspectable, redactable, and signed. Multi-Provider Agent Console A side panel manages web-session providers, API providers, local models, MCP servers, and peer endpoints through one routing policy. Tasks can fail over without losing operation identity, but provider changes and cost/privacy implications remain visible. Visual Trust Ledger Every autonomous task exposes a concise Perceive–Plan–Act–Verify timeline backed by the operation trace. Users can inspect the target selected, evidence observed, action attempted, postcondition obtained, model/provider used, and any recovery or rollback. Script and Workflow Marketplace torsionfield.de hosts signed metadata and discovery, while packages may be downloaded from their declared source. Listings expose permissions, capabilities, compatible sites, provenance, reproducibility evidence, health across browser versions, selector-drift status, update source, license, and peer

reviews. Zero-Friction Migration On first run, the extension detects supported managers and archives, previews the import, preserves enabled state and values where possible, flags incompatible grants, and leaves the original manager untouched until the user completes verification. Local and Desktop Tool Bridge The daemon exposes configurable local commands, files, MCP servers, OCR, image tools, browser drivers, and desktop automation through the same evidence-based operation model. Browser-only workflows degrade explicitly when a local capability is unavailable. Capability Contribution Users may opt in to expose idle local capabilities or authenticated provider tabs to selected peers or projects. Contribution is disabled by default, bounded by explicit profiles, rate limits, budgets, schedules, revocation, and visible activity. The core product **MUST NOT** depend on speculative tokens or hidden resource leasing. 25.4 Unified intermediate representation Visual workflows, recorded macros, generated userscripts, agent plans, imported Selenium/Playwright steps, and NetworkScript endpoints **SHOULD** compile to a common Operation Graph. Nodes describe typed reads, writes, navigation, waits, assertions, scripts, model calls, daemon calls, branches, loops, artifacts, and subflows. Edges describe control, data, compensation, and evidence dependencies. The Operation Graph allows one execution engine, debugger, trace model, permission system, test runner, and marketplace format to serve every authoring surface.

§26

STAGED TARGET

Superfluid prompt-commit workflow

> **Purpose.** Defines Prompt Commits and the replayable, licensed, tested process for keeping incorporated capabilities synchronized.

26.1 Purpose
The Superfluid workflow keeps incorporated capabilities synchronized with evolving upstream extensions. It combines ordinary Git commits with immutable Prompt Commits: replayable transformation specifications that explain how an upstream capability is identified, extracted, normalized, integrated, tested, and maintained.
A Prompt Commit is not a commit message and is not an instruction to copy whatever changed upstream. It is a versioned engineering object with source boundaries, architectural intent, license constraints, invariants, transformation instructions, deterministic tooling, expected outputs, tests, and rollback evidence.
26.2 Core records
ExtensionIncorporation
Identifies an upstream project, repository, license, allowed incorporation strategy, pinned baseline, watched branches/releases, extracted capabilities, target modules, attribution, maintainers, update policy, and current health.
CapabilityContract
Defines observable behavior independent of implementation. It includes inputs, outputs, permissions, state, failure cases, postconditions, compatibility fixtures, performance bounds, privacy rules, and acceptance tests.
PromptCommit
Defines one replayable source-to-product transformation.
ExtractionRun
Records one execution of one or more Prompt Commits against a specific upstream transition and stores tool/model versions, inputs, outputs, diffs, tests, evidence, decisions, and signatures.
26.3 Prompt Commit schema
A Prompt Commit MUST contain at least:

```
{
  "protocol": "torsionfield-prompt-commit/1",
  "promptCommitId": "pc_sha256:…",
  "title": "Extract visual workflow compiler changes",
  "upstream": {
    "incorporationId": "automa",
    "fromRevision": "…",
    "toRevision": "…",
    "sourceGlobs": ["src/workflow/**"],
    "semanticQueries": ["tree-sitter-or-ast-grep-query-id"]
  },
  "capabilityContracts": ["cap.visual-workflow.compiler/1"],
  "strategy": "direct|reimplementation|compatibility|bridge",
  "targetModules": ["extension/src/workflows/compiler"],
  "invariants": [],
  "nonGoals": [],
  "licensePolicy": {},
  "transformationPrompt": "…",
  "toolchain": {
    "parsers": [],
    "rewriters": [],
    "modelPolicy": {},
    "formatters": []
  },
  "expectedOutputs": [],
  "tests": [],
  "rollback": {},
  "signer": "…",
  "signature": "…"
}
```

The transformation prompt MUST reference capability contracts and architecture boundaries rather than asking a model to “merge the new upstream code.”
26.4 Update-extraction pipeline
For every watched upstream transition, the incorporation service MUST:
1. detect a release, tag, security advisory, or relevant commit;
2. mirror the exact source into a quarantined, read-only vendor workspace;
3. verify repository identity, commit signatures when available, license files, dependency changes, and known security findings;
4. compute textual, manifest, dependency, API, UI, and semantic AST diffs using Git plus incremental parsers and structural search rules;
5. map changed symbols and behavior to Capability Contracts;
6. classify changes as irrelevant, compatible, behavior-changing, security-critical, license-changing, or architecture-breaking;
7. replay the relevant Prompt Commits in an isolated branch with recorded tool and model versions;
8. generate the smallest architecture-conforming patch, attribution changes, migrations, and updated tests;
9. run formatting, type checking, linting, static analysis, dependency policy, license checks, and secret scanning;
10. run unit, contract, fixture, migration, browser, visual, security, and end-to-end tests;
11. compare the resulting behavior with the pinned upstream version using golden traces and side-by-side black-box scenarios;
12. create an Incorporation Pull Request containing the upstream diff, capability impact report, generated patch, prompt-commit replay record, license decision, test evidence, screenshots, risks, and rollback plan;
13. merge automatically only when repository policy, branch protection, review requirements, and every acceptance gate permit it;
14. sign the new incorporation ledger, capability pack, and release artifacts;
15. deploy to canary users or test nodes before broad release;
16. roll back automatically if field health falls below the configured threshold.
Dependency and upstream update detection MAY use Renovate-style pull requests and shared configuration, but semantic capability extraction remains a separate layer. Tree-sitter-style incremental parsing and ast-grep-style structural search SHOULD be used to avoid fragile line-based transformations.
26.5 Repository layout

```
incorporated/
```

```
  registry/
  sources/
  capability-contracts/
  prompt-commits/
  extraction-rules/
  license-decisions/
  golden-traces/
  compatibility-fixtures/
  provenance/
tools/incorporation/
  watcher/
  mirror/
  diff-engine/
  semantic-index/
  prompt-commit-runner/
  behavior-comparator/
  license-gate/
  pull-request-generator/
  canary-monitor/
extension/src/incorporated/
  compatibility/
  workflows/
  command-interface/
  snapshots/
  capsules/
  migration/
26.6 Incorporation APIs
@grant CAT.incorporation
CAT.incorporation.list()
CAT.incorporation.inspect(incorporationId)
CAT.incorporation.checkUpdates(incorporationId?)
CAT.incorporation.planUpdate(incorporationId, revision)
CAT.incorporation.applyPromptCommit(promptCommitId, options)
CAT.incorporation.runAcceptance(extractionRunId)
CAT.incorporation.activate(extractionRunId)
CAT.incorporation.rollback(incorporationId, revision?)
CAT.incorporation.exportProvenance(incorporationId, options?)
Ordinary users receive signed, tested incorporation updates. Maintainer-only APIs that inspect source or execute
transformations are disabled unless developer mode and the required capability profile are active.
26.7 Superfluid acceptance criteria
- a new upstream release is detected and pinned without mutating the active product;
- unchanged upstream behavior produces no unnecessary product patch;
- a behavior change is mapped to the correct Capability Contract;
- relevant Prompt Commits replay with all inputs, tools, and outputs recorded;
- an incompatible or changed license blocks unauthorized direct integration;
- generated changes cannot activate before static, contract, fixture, and end-to-end tests pass;
- the incorporation pull request explains every generated change and preserves attribution;
- a failed canary rolls back to the last-known-good incorporation revision;
- field diagnostics identify incorporation revision and Prompt Commit provenance;
- the system can reproduce the accepted patch from the recorded source revision and transformation environment
within documented reproducibility limits.
```

§27

STAGED TARGET

Web product, domain, and network architecture

Purpose. Defines product naming, domain responsibilities, registration, first-party integration, and background services.

TF-S27-R001 27.1 Product naming The public product **SHOULD** use a coherent stack: Torsionfield Runtime The complete local-first browser automation platform. Torsionfield Extension The evolved ScriptCat-compatible userscript manager, agent runtime, recorder, workflow studio, and browser control surface. Torsion Node The optional local daemon providing durable execution, local tools, models, artifacts, identity, and peer networking. Torsionfield Network The signed P2P membership, rendezvous, relay, capability, and project coordination layer. Transductive Workspaces The science, art, and general collaboration applications built on the Runtime and Network. The product **MUST NOT** be presented primarily as "a modified ScriptCat." ScriptCat is the compatible foundation and an acknowledged upstream; the product is a local-first userscript and browser-agent runtime with reproducible automation, self-healing, local tools, and optional peer collaboration. 27.2 torsionfield.de responsibilities torsionfield.de is the canonical technology, distribution, identity, and network-administration site. It **MUST** provide: - product presentation and interactive demonstrations; - extension and daemon downloads; - signed release, update, adapter-pack, capability-pack, and incorporation manifests; - documentation, API references, compatibility information, and migration guides; - account and public-identity registration for users who choose Network Node mode; - browser, daemon, and device pairing; - P2P node registration, membership issuance, relay and rendezvous configuration, invitations, and revocation; - node, device, endpoint, project, budget, permission, and activity dashboards; - script, workflow, capsule, adapter, and capability discovery; - security advisories, status, privacy policy, transparency reports, and changelog; - developer portal, SDKs, Prompt Commit provenance, upstream attribution, and incorporation health. Local browser use **MUST** remain possible without registration. The site stores public identity and membership metadata required for network operation; it **MUST NOT** receive browser cookies, provider session credentials, local model secrets, or private keys. 27.3 Network registration and pairing The standard Network Node registration flow is: 1. the user installs the extension and may import existing scripts; 2. the user optionally installs Torsion Node and completes local browser-daemon pairing; 3. the browser or daemon generates local root/device identities; 4. torsionfield.de/pair creates a short-lived challenge, QR code, and signed deep link; 5. the extension and daemon sign the challenge and return public identity claims; 6. the user selects display identity, discoverability, relay use, project memberships, contribution policy, budgets, and capability profile; 7. the server issues a signed membership credential and bootstrap/relay configuration; 8. the node completes direct and relayed connectivity health tests; 9. the dashboard displays the node, active devices, advertised endpoints, and current health; 10. credentials can be rotated or revoked from either the local product or the site. Account recovery **MUST NOT** silently replace a lost cryptographic identity. Recovery creates or authorizes a new device identity according to the configured root-identity and social/administrative recovery policy. 27.4 Transductive domains transductive.org General collaboration, community, public projects, workflow and NetworkScript discovery, participatory development, documentation, governance, and cross-domain identity. It is the broad workspace layer rather than the infrastructure control plane. transductive.science Research projects, task queues, evidence contracts, replicated execution, reproducible Web Capsules, datasets, provenance, result comparison, review, and publication assembly. Scientific tasks can request specific providers, tools, replicas, evidence, and conflict-resolution procedures. transductive.art Creative workflows, media-generation pipelines, prompt/script compositions, collaborative productions, asset provenance, reusable project packs, galleries, and optional peer rendering or tool contribution. All three sites integrate with the extension, local daemon, and background services through signed first-party protocols. They **MUST NOT** depend on fragile DOM scraping for core first-party functions. 27.5 First-party site integration contract The extension **MUST** provide a FirstPartySiteAdapter for approved Torsionfield and Transductive origins. The adapter supports: - authenticated site-to-extension discovery; - signed challenge and pairing exchange; - project join/leave; - task preview, acceptance, refusal, progress, and result submission; - artifact and capsule transfer with explicit user policy; - endpoint and capability publication; - local health and compatibility summaries; - opening extension views, agent workspaces, or local dashboards; - revocation and sign-out. Communication uses a versioned postMessage or MessageChannel bridge bound to exact origins and nonces, plus extension runtime messaging. Large or durable transfers use the daemon or signed upload URLs. Custom URL schemes or extension deep links **MAY** be used for installation and pairing, but every request is authenticated, replay-protected, user-attributable, and capability-checked. 27.6 Background services The background control plane **SHOULD** expose separate services for identity/membership, device registration, node presence, relay/rendezvous, projects, tasks, package manifests, provenance, marketplace metadata, notifications, and status. Durable Objects or equivalent stateful actors **MAY** own node, project, and task sessions; object storage **MAY** hold signed packages and encrypted artifacts; relational storage **MAY** hold accounts, memberships, metadata, and audit events. The P2P network remains the preferred data path for peer execution and artifact exchange when available. The web control plane coordinates identity, discovery, invitations, leases, and fallback transport without becoming the owner of participant model credentials or all task data.

§28

STAGED TARGET

Product presentation, adoption, and user experience

> **Purpose.** Defines how the product must present immediate browser-only value, progressive installation, and evidence-backed claims.

TF-S28-R001 28.1 Positioning The primary promise is: Turn any website into a reliable personal tool—record it, script it, repair it with AI, run it locally, and share it as a verified workflow when you choose. The website and extension **SHOULD** demonstrate a complete transformation rather than lead with architectural terminology: 1. record a repetitive browser task; 2. compile it into a readable workflow and userscript; 3. show the generated tests and permissions; 4. break a selector and demonstrate self-repair; 5. run it locally, through another model, or on an authorized peer; 6. inspect the evidence ledger and reproducible capsule. 28.2 No-brainer installation criteria Installation is compelling only when the product is immediately useful before the user learns the network architecture. The release **MUST** provide: - a three-minute guided setup; - one-click import from supported userscript managers; - a familiar script-manager mode for ordinary users; - a recorder that produces a successful first automation without code; - an AI-assisted editor that explains changes as diffs and tests; - transparent, progressive permission requests rather than an unexplained maximum-permission wall; - browser-only operation with clear optional daemon and network upgrades; - a sample workflow gallery and an interactive demo before account creation; - local-first defaults, visible data paths, and plain-language security explanations; - signed downloads, reproducible-build information, open-source provenance, and public issue tracking; - graceful uninstall/export so users never fear lock-in. 28.3 Principal product surfaces Extension popup Fast per-site script control, recent workflows, command palette entry, recording, Agentify, health, and active operation status. Side panel Agent console, workflow runner, recorder timeline, task queue, model/provider routing, evidence ledger, and first-party project interaction. Script and Workflow Studio Code editor, visual graph, synchronized recorder steps, metadata, values, permissions, tests, versions, provenance, and deployment targets. Agent Workspace Protected agent tabs, pools, provider health, current work, budgets, recovery, and release controls. Incorporation Dashboard Tracked upstream extensions, pinned revisions, license strategy, extracted capabilities, available updates, Prompt Commit runs, pull-request evidence, canary health, and rollback. Torsion Node Dashboard Local services, models, MCP servers, commands, files, peers, projects, endpoints, artifacts, logs, budgets, updates, and identity. Web dashboards Account, devices, node membership, projects, marketplace, invitations, releases, provenance, security, and status. 28.4 Professional presentation requirements The public site **MUST** include architecture diagrams, permission and data-flow diagrams, a live or simulated operation trace, a compatibility matrix, concrete productivity benchmarks, video demonstrations, security documentation, API documentation, changelog, roadmap, and source/provenance links. Claims such as “self-healing,” “private,” “decentralized,” “reproducible,” or “provider-independent” require visible qualification and test evidence. The presentation **MUST** distinguish local-only, daemon-enabled, and network-enabled behavior. 28.5 Additional implementation and acceptance gates Sections 21 and 22 are extended with the following mandatory work: - implement the unified Operation Graph and converters for recorder, visual workflow, userscript, imported macro, and NetworkScript representations; - implement the incorporation registry, Capability Contracts, Prompt Commit runner, semantic diff pipeline, provenance ledger, and canary rollback; - implement zero-friction migration and compatibility tests against supported managers; - implement command palette, link hints, recorder, visual workflow studio, Web Capsules, workspace snapshots, self-healing scripts, and evidence ledger; - implement FirstPartySiteAdapter and torsionfield.de registration/pairing protocols; - implement the torsionfield.de, transductive.org, transductive.science, and transductive.art integration contracts and background services; - produce the ScriptCat patch dossier and full implementation-closure artifact set. Release acceptance additionally requires: - import of a representative ScriptCat, Violentmonkey, and Tampermonkey archive without silent data loss; - recording one task and compiling it into synchronized workflow, userscript, and endpoint artifacts; - repair of a deliberately drifted target through semantic/fingerprint recovery with evidence and rollback; - creation, restoration, export, and import of a workspace snapshot and Web Capsule; - discovery and replay of an upstream incorporation update through a Prompt Commit with license and test gates; - browser-only onboarding without account or daemon; - daemon pairing and local-tool execution; - torsionfield.de network registration, membership issuance, relay health, revocation, and device rotation; - project/task exchange with each Transductive first-party site through the signed site bridge; - end-to-end evidence showing which component, identity, provider, script, capability revision, Prompt Commit, and artifact produced every result.

§29

STAGED TARGET

Configuration continuum and superfluid build matrix

> **Purpose.** Unifies compile-time and runtime configuration through signed profiles, capability graphs, and reproducible build variants.

29.1 Configuration is a single continuum

TF-S29-R001 Torsionfield **MUST** treat compile-time flags, link-time selection, packaging choices, installation profile, project policy, startup configuration, and runtime feature state as successive lowering stages of one declarative configuration model. Compile-time selection is configuration that is resolved earlier; it **MUST NOT** create an unrelated architecture.

A signed TFProfile is the single source of truth. It declares required capabilities, prohibited capabilities, target platforms, trust boundaries, privacy mode, browser mode, transports, storage backends, resource limits, user-interface surfaces, update channel, and acceptance suite. The profile compiler resolves the declaration into:

- Cargo features and crate inclusion;
- Chromium patch and feature bundles;
- extension manifest permissions and host permissions;
- userscript grants and TF API declarations;
- daemon services and runner backends;
- WASM modules;
- packaging and installer contents;
- runtime defaults and project policy;
- tests, audit probes, release metadata, and rollback information.

TF-S29-R002 The generated artifacts **MUST** retain the profile digest and the complete resolution explanation. A user, verifier, or governance process can therefore ask why a capability exists in a binary and receive the exact profile rule, dependency, source revision, and build evidence that caused its inclusion.

29.2 Constitutional kernel and capability modules

Minimum impact is achieved by keeping a small constitutional kernel stable and moving optional behavior into typed capability modules.

The constitutional kernel owns:

- canonical serialization and signature verification;
- identity ownership and key delegation;
- capability evaluation and isolation;
- operation identity, effect evidence, and replay protection;
- artifact addressing and provenance;
- profile resolution and compatibility negotiation;
- reproducible-build requirements;
- update verification, activation, rollback, export, and exit;
- the rule that compute contribution does not purchase governance power;
- the rule that self-modification is observable and reversible.

TF-S29-R003 Capability modules **MAY** provide provider adapters, browser personas, Tor/Arti transport, libp2p, local mesh, Blender execution, visual workflows, MCP bridges, remote rendering, fingerprint auditing, model routing, website services, or project-specific functions. Modules communicate through versioned contracts and **MUST NOT** reach across layers through undocumented global state.

TF-S29-R004 The dependency structure **MUST** be a capability graph, not a growing set of nested conditional branches. A profile resolver rejects incompatible capability combinations before compilation.

29.3 Build variants

TF-S29-R005 The same repository **MAY** produce multiple official variants, including:

- tf-node-minimal;
- tf-node-tor;
- tf-node-builder;
- tf-node-forge;
- tf-node-orkz-mesh;
- tf-browser-standard;
- tf-browser-persona;
- tf-browser-uniform;
- tf-browser-disposable;
- tf-browser-remote-hardened;
- tf-fingerprint-auditor;
- tf-verifier;
- tf-governance-builder.

TF-S29-R006 All variants expose the same canonical TF API namespace. Capability discovery reports which contracts are present, absent, degraded, or remotely supplied. Software **MUST** fail explicitly on an unavailable capability rather than silently substituting a semantically different one.

A build variant is content-addressed by source revision, TFProfile digest, toolchain lock, dependency graph, Prompt Commit set, environment manifest, and output hashes. Different compilations are therefore governed configurations of one system rather than independent products.

29.4 ChromiumFish-derived browser capability layer

TF-S29-R007 ChromiumFish-style functionality **SHOULD** be incorporated through separable capability contracts rather than by making every Torsionfield build depend on a permanent browser fork.

The initial browser capability modules are:

torsion-browser-persona A deterministic, internally coherent browser persona for stable pseudonymous accounts and agent sessions.

torsion-browser-uniform A release-cohort profile designed to make many users present the same restricted browser class rather than a unique synthetic identity.

torsion-browser-disposable A fresh persona, storage partition, Arti isolation context, and browser profile for a bounded operation.

torsion-render-bridge An authenticated optional renderer for canvas, WebGL, fonts, or other hardware-derived outputs when local rendering would expose an unwanted host signature.

torsion-fingerprint-auditor A continuous comparison and regression system covering JavaScript-visible APIs, property descriptors, canvas, WebGL, fonts, audio, WebRTC, Client Hints, TLS, HTTP protocol behavior, storage, automation artifacts, and reference-browser consistency.

TF-S29-R008 The persona engine, uniform profile, Arti transport, storage partition, and account/session policy **MUST** be resolved together. Changing only the User-Agent or only the network proxy is not a valid privacy profile.

29.5 Superfluid compilation

TF-S29-R009 Prompt Commits **MAY** transform not only source code but also TFProfiles, capability graphs, build recipes, migrations, test matrices, and packaging. An upstream browser or daemon change triggers only the affected profile resolutions and variants.

TF-S29-R010 The build service **MUST**:

1. resolve the requested TFProfile;
2. produce a machine-readable capability graph;
3. pin source, dependencies, compiler, linker, system image, and environment;

4. compile in an isolated workspace;
5. generate provenance and a software-bill-of-materials artifact;
6. run variant-specific tests and cross-variant contract tests;
7. compare reproducible outputs when the platform permits;
8. sign the build receipt;
9. publish only through threshold-authorized release metadata;
10. retain rollback to the last-known-good profile and artifact.

PART V

Participatory work, verification, and governance

§30

STAGED TARGET

Participatory development and work donation

> **Purpose.** Defines opt-in contribution of source generation, review, build, test, documentation, and compute work.

30.1 Torsionfield is the first participatory project

TF-S30-R001 Every installation **MUST** present Torsionfield itself as the first available project. Participation is not limited to donating idle CPU or GPU resources. Members can donate source generation, coding, compilation, testing, reproduction, static analysis, security review, documentation, benchmarking, adapter repair, build verification, world assets, moderation work, or governance review.

TF-S30-R002 When the product encounters a reproducible defect, missing adapter, failed build, untranslated text, documentation gap, degraded capability, or unmet test, it **MAY** create a local ProblemRecord and offer the user explicit choices:

- Help solve;
- Donate source generation;
- Donate compile or test capacity;
- Reproduce this failure;
- Verify another contribution;
- Schedule for tonight;
- Publish to the project queue;
- Keep private;
- Dismiss.

No problem is published without the configured project and privacy policy. Prompts, responses, credentials, private file contents, and local paths are excluded or redacted unless the user deliberately authorizes their inclusion.

30.2 Donation types

A WorkOffer declares one or more contribution classes:

- source.generate;
- source.patch;
- source.review;
- build.compile;
- build.reproduce;
- test.unit;
- test.integration;
- test.browser;
- test.fuzz;
- security.static;
- security.dynamic;
- dependency.audit;
- license.review;
- result.verify;
- failure.reproduce;
- docs.write;
- translation.write;
- benchmark.run;
- world.primitive;
- blender.build;

- artifact.cache;
- relay.serve.

Each offer specifies capabilities, providers, toolchains, acceptable projects, privacy classes, resource budgets, schedules, electricity or thermal policy, network policy, storage limits, expiry, revocation, and whether human confirmation is required.

30.3 Problem and task model

ProblemRecord represents an observed need. It includes severity, impact radius, affected versions and profiles, reproduction evidence, privacy class, dependencies, known workarounds, requested capabilities, urgency, and ownership.

TaskManifest represents a bounded unit of work. It includes:

- taskId and projectId;
- immutable problem and source references;
- exact prompt or procedure digest;
- encrypted input bundle digest;
- expected output schema;
- allowed tools and network destinations;
- requested execution profile;
- deadline and lease duration;
- urgency and priority factors;
- required evidence;
- replication and verification policy;
- acceptance tests;
- privacy and disclosure policy;
- compensation or credit policy when applicable;
- issuer identity, signature, and expiry.

TF-S30-R003 Priority is computed transparently from configured factors such as safety severity, number of blocked users, dependency centrality, release deadline, age, available solvers, and project governance decisions. A high resource donor **MUST NOT** receive unilateral power to assign priority.

30.4 Queue and scheduling

The project queue supports interactive claims, scheduled commitments, and unattended contribution windows. A user may configure “leave Torsionfield on tonight” with bounded capabilities and a maximum time, cost, energy, temperature, storage, and network budget.

The scheduler operates through offers and leases:

problem → triage → signed task → publish → offer → claim → lease → execute → submit → scan → verify → accept or reject

A lease prevents uncontrolled duplicate execution but does not prohibit deliberate replication. Tasks may request multiple independent results, independent verification, or diversity across providers, platforms, toolchains, and geographic or organizational trust domains.

30.5 User interface

TF-S30-R004 The extension side panel and Torsion Node dashboard **MUST** make project contribution central but non-coercive. The interface displays:

- problems currently affecting the user;
- urgent Torsionfield tasks;
- tasks matching the user’s available capabilities;
- active and scheduled donations;
- resource budgets and kill controls;
- artifact and provenance status;

- requested verification work;
- accepted, rejected, quarantined, and superseded contributions;
- the exact effect of a contribution on releases or proposals.

TF-S30-R005 The interface **MUST** distinguish donation of model execution, local compute, code, review, and governance participation. Contribution remains opt-in and revocable.

§31

STAGED TARGET

Source generation pool and provider execution boundary

> **Purpose.** Defines the boundary around participant-owned model sessions and the limits of what provider execution can prove.

31.1 Canonical name and credential boundary

TF-S31-R001 The colloquial "Codex token pool" is implemented as the Source Generation Pool or Model Work Pool. It **MUST NOT** pool, transfer, lease, reveal, or centrally store provider account credentials, browser cookies, API keys, subscription access, or usage entitlements.

Each participant executes through that participant's own locally configured provider session, API account, local model, or approved organizational endpoint. The project control plane sees signed task manifests, capability offers, leases, receipts, and artifact references. It does not possess the participant's provider credential.

Provider-specific policy is part of capability matching. A task that cannot be performed under the participant's provider policy is refused or routed elsewhere.

31.2 Execution boundary

For a web-session provider, the standard path is:

1. the participant accepts or pre-authorizes a task class;
2. the local extension obtains the signed TaskManifest;
3. ScriptVault or Torsion Node releases the task decryption key only to the authorized local operation;
4. the provider adapter opens or selects an isolated agent tab;
5. the exact prompt and allowed attachments are inserted and observed;
6. the provider response is captured with message identity and operation evidence;
7. declared local tools, builds, and tests execute in the bounded runner;
8. outputs are packaged, hashed, scanned, and signed locally;
9. only the permitted result and provenance fields leave the device.

For API or local-model execution, the same TaskManifest and receipt model applies without a browser tab.

31.3 Provider result limits

A model response is an untrusted proposed artifact. It is not made trustworthy by a prompt instruction, a hidden phrase, a checksum printed by the model, or a base64 token included in the answer.

The system can prove that:

- a signed task manifest existed;
- a specific encrypted input bundle was released to a specific authorized executor;
- a particular prompt and result were observed by the local Torsionfield operation;
- the resulting artifacts have specific hashes;
- tests and scanners produced recorded evidence;
- a node identity signed the receipt;
- other nodes reproduced or reviewed the result.

TF-S31-R002 Without a provider-signed request/response receipt or a trustworthy attested execution path, the system cannot cryptographically prove that a remote proprietary model ran an exact unmodified prompt. The specification **MUST** state this limitation directly.

§32

STAGED TARGET

Signed task bundles, ScriptVault, MCP notary, and verification

> **Purpose.** Defines encrypted task bundles, ScriptVault, the MCP Notary, execution receipts, verification levels, and supply-chain hardening.

32.1 Bundle format

TF-S32-R001 Base64 is an encoding and provides no confidentiality or integrity. Large task inputs **SHOULD** be packaged as a deterministic archive, preferably tar plus Zstandard or a profile-approved ZIP form, and encrypted with a random per-task data-encryption key using a modern authenticated-encryption envelope.

The encrypted bundle contains only the files required by the task. Its manifest records paths, sizes, media types, plaintext hashes, normalization rules, source revision, and disclosure class. The public TaskManifest contains the ciphertext hash, not the decryption secret.

32.2 Key flow

The recommended key flow is:

1. the task issuer creates a random data-encryption key;
2. the input archive is encrypted and content-addressed;
3. the coordinator signs the TaskManifest containing taskId, source revision, prompt digest, ciphertext digest, expected outputs, nonce, expiry, and verification policy;
4. a participant claims a lease with a device or executor public key;
5. the coordinator or authorized project key service wraps the data-encryption key to that executor;
6. ScriptVault confirms local policy and releases the wrapped key to Torsion Node;
7. Torsion Node decrypts only inside the selected Debian VM, container, microVM, or local sandbox;
8. plaintext is destroyed or retained according to the task policy after execution;
9. outputs are independently encrypted when their disclosure class requires it.

TF-S32-R002 Passwords **MUST NOT** be inserted into model prompts. A password visible to the model or contributor cannot prove that the result was honestly generated.

32.3 ScriptVault

TF-S32-R003 ScriptVault is the local trust and secret boundary for browser-based execution. It **MAY** use non-extractable WebCrypto keys, WebAuthn, OS credential storage, TPM-backed keys, or paired Torsion Node storage.

ScriptVault exposes narrowly scoped operations:

```
TF.vault.inspectEnvelope(envelope)
TF.vault.authorizeTask(taskId, policy)
TF.vault.unwrapTaskKey(taskId, executorEvidence)
TF.vault.sealArtifact(artifact, recipients)
TF.vault.signReceipt(receipt)
TF.vault.revokeLease(leaseId)
TF.vault.destroyTaskMaterial(taskId)
```

Scripts and chat pages receive plaintext task material only when the capability profile explicitly permits it. Private keys never become prompt content.

32.4 MCP Notary

TF-S32-R004 The centrally hosted MCP service is a Notary and Coordinator, not an oracle that declares model output true. It **MAY**:

- issue signed short-lived task challenges;
- publish task manifests and encrypted bundle references;
- grant and revoke work leases;
- release wrapped task keys after policy checks;
- timestamp receipts;
- verify hashes, signatures, expiry, sequence, and replay constraints;
- append receipt digests to a transparency log;
- collect independent review attestations;
- evaluate quorum and acceptance policy;
- publish release candidates and rollback metadata.

TF-S32-R005 It **MUST NOT** claim that a valid receipt proves semantic correctness or proves proprietary model execution. Compromise of the Notary **MUST NOT** permit silent release replacement; threshold release keys, transparency, independent mirrors, and client-side verification protect the final activation path.

32.5 Execution receipt

ExecutionReceipt includes:

- taskId, leaseId, operationId, attemptId, and executor identity;
- TaskManifest digest and encrypted-input digest;
- resolved TFProfile and capability graph digest;
- provider or model descriptor;
- prompt digest and response digest;
- adapter, extension, daemon, browser, VM, and toolchain revisions;
- start, end, and monotonic timing evidence;
- network and filesystem policy;
- output artifact digests;
- stdout, stderr, trace, screenshot, and test references according to privacy policy;
- scan and analyzer results;
- resource use;
- attestation evidence when available;
- node signature.

TF-S32-R006 The raw prompt and response **MAY** remain private while their digests and permitted evidence are published.

32.6 Verification levels

Every donated result has an explicit trust level:

UNTRUSTED Received but not scanned.

SCANNED Passed configured archive, malware, secret, dependency, license, static-analysis, and policy checks.

REPRODUCED An independent executor reproduced declared outputs or test outcomes from the same manifest within stated limits.

INDEPENDENTLY_VERIFIED One or more reviewers or nodes inspected the patch, behavior, or evidence and signed attestations.

ATTESTED Execution includes verified hardware or measured-boot evidence tied to the executor profile.

QUORUM_VERIFIED The project's threshold policy over diverse independent verifiers has been satisfied.

Attestation proves properties of an execution environment and measured software. It does not by itself prove that generated source is correct, benevolent, license-compatible, or the unique output of a model.

32.7 Donation hardening pipeline

All source and binary donations enter quarantine. The default pipeline is:

1. reject malformed manifests, expired leases, replay, and hash mismatch;
2. unpack inside a non-privileged isolated workspace;
3. deny network by default and allow only declared destinations;
4. run archive, malware, secret, dependency, license, and prompt-injection scans;
5. parse and index source structurally;
6. compute source, dependency, manifest, generated-code, and binary diffs;
7. run formatting, type checking, linting, static analysis, policy checks, and fuzzing as configured;
8. execute tests in disposable runners;
9. build requested variants reproducibly;
10. compare outputs and behavior with reference and independent builds;
11. request human or peer review according to affected rights and risk;
12. canary the result;
13. activate only through signed release policy;
14. roll back on failed postconditions or field health.

Changes to the constitutional kernel, identity, update verification, key handling, capability enforcement, or governance activation always require a higher threshold and human review.

32.8 Supply-chain provenance

`TF-S32-R007` Torsionfield provenance **SHOULD** be compatible in spirit and structure with SLSA provenance, in-toto step attestations, reproducible-build records, Sigstore-style transparency, and TUF-style threshold update metadata. Compatibility **MAY** be implemented directly or through adapters, but the internal model remains provider-independent.

§33

FIRST RELEASE

TF API and ScriptCat compatibility

> **Purpose.** Makes TF.* the canonical API while retaining CAT and GM compatibility for the ScriptCat ecosystem.

33.1 Namespace transition

TF is the canonical product API. Existing CAT APIs remain as a compatibility layer during migration.

```
CAT.router maps to TF.router.
CAT.agent.tabs maps to TF.agents.
CAT.agent.adapters maps to TF.adapters.
CAT.script.mutation maps to TF.scripts.
CAT.daemon maps to TF.node.
CAT.identity maps to TF.identity.
CAT.network maps to TF.network.
CAT.incorporation maps to TF.incorporation.
```

TF-S33-R001 New code **SHOULD** use TF.*. Imported ScriptCat scripts continue to use supported GM and CAT names through generated compatibility shims and declared grants.

33.2 New canonical services

```
@grant TF.profile
TF.profile.getActive()
TF.profile.resolve(profile)
TF.profile.explain(capability)
TF.profile.listVariants()
TF.profile.activate(profileId)
TF.profile.diff(a, b)
```

```
@grant TF.capabilities
TF.capabilities.list(filter?)
TF.capabilities.get(capabilityId)
TF.capabilities.explain(capabilityId)
TF.capabilities.offer(descriptor)
TF.capabilities.revoke(offerId)
```

```
@grant TF.work
TF.work.publishProblem(problem)
TF.work.publishTask(task)
TF.work.list(filter)
TF.work.offer(taskId, offer)
TF.work.claim(taskId, options)
TF.work.schedule(taskId, schedule)
TF.work.complete(leaseId, result)
TF.work.requestVerification(resultId, policy)
TF.work.review(resultId, attestation)
TF.work.cancel(operationId)
```

```
@grant TF.build
TF.build.resolveProfile(profile)
TF.build.compileVariant(profile, options)
TF.build.reproduce(buildId, options)
TF.build.compare(buildIds)
TF.build.getReceipt(buildId)
TF.build.publishCandidate(buildId)
```

```
@grant TF.verify
TF.verify.scan(artifact, policy)
TF.verify.reproduce(resultId, options)
TF.verify.attest(executionId)
TF.verify.requestQuorum(resultId, policy)
TF.verify.status(resultId)
```

```
@grant TF.vault
TF.vault.inspectEnvelope(envelope)
TF.vault.authorizeTask(taskId, policy)
TF.vault.unwrapTaskKey(taskId, evidence)
TF.vault.sealArtifact(artifact, recipients)
TF.vault.signReceipt(receipt)
TF.vault.destroyTaskMaterial(taskId)
```

```
@grant TF.provenance
TF.provenance.createStatement(subjects, predicate)
TF.provenance.sign(statement)
TF.provenance.verify(statement)
TF.provenance.appendTransparency(statement)
TF.provenance.trace(artifactId)
```

```
@grant TF.browserPersona
TF.browserPersona.listProfiles()
TF.browserPersona.resolve(profile, networkContext)
TF.browserPersona.launch(profile, options)
TF.browserPersona.audit(sessionId)
TF.browserPersona.destroy(sessionId)
```

```
@grant TF.governance
TF.governance.proposeBuild(proposal)
TF.governance.simulate(proposalId, profiles)
TF.governance.review(proposalId, review)
TF.governance.approve(proposalId, credential)
TF.governance.activate(proposalId, epoch)
TF.governance.rollback(proposalId, reason)
```

Every method returns operation-scoped evidence and evaluates the active capability profile.

33.3 Shared intermediate objects

Visual workflows, userscripts, agent plans, provider prompts, build recipes, donated work, verification jobs, and governance builds use compatible Operation Graph and Artifact Graph primitives. This prevents the work-donation layer from becoming a second scheduler or security system.

A task node may invoke a browser adapter, model provider, source transformer, compiler, test runner, analyzer, verifier, or governance gate. The same routing, permission, cancellation, evidence, recovery, and unknown-outcome semantics apply.

§34

STAGED TARGET

Self-modification and democratic build governance

> **Purpose.** Defines observable, reversible self-modification and governed activation without making model output sovereign.

34.1 Self-modification as a system invariant

TF-S34-R001 Torsionfield **MUST** remain capable of modifying its userscripts, extension modules, adapters, website, Rust services, build profiles, tests, and governed runtime policies through observable proposals and reproducible builds. Self-modification **MUST NOT** mean that arbitrary model output can overwrite the active system.

Every self-change is represented by an immutable ChangeProposal with source, rationale, affected capabilities and rights, patch, Prompt Commits, migrations, tests, security analysis, build profiles, activation policy, and rollback.

34.2 Governable runtime

TF-S34-R002 Subject to constitutional constraints, governance **MAY** replace or modify:

- social and project schemas;
- task priority formulas;
- contribution and credit rules;
- compute scheduling and verification thresholds;
- browser persona and anonymity profiles;
- provider adapters and model routing;
- website presentation and onboarding;
- moderation and labeling services;
- world primitive types and Blender compilation;
- Rust server modules;
- transport providers;
- build profiles;
- release channels;
- governance procedures themselves.

The official Rust server is a reference implementation, not the permanent sovereign of the protocol. Independent implementations can participate when they pass the declared protocol and security conformance suites.

34.3 Governance Build Proposal

A GovernanceBuildProposal contains:

- proposalId and projectId;
- source and parent release revisions;
- target modules and TFProfiles;
- patch and Prompt Commit references;
- rationale and alternatives;
- affected rights, privacy, authority, and compatibility;
- migration and data-export plan;
- tests and acceptance mapping;
- security and supply-chain analysis;
- reproducible-build manifest;
- requested independent build and review quorum;
- approval rule;
- activation epoch and canary population;

- rollback artifact and rollback triggers;
- proposer signature.

The lifecycle is:

proposal → independent builds → tests and analysis → community simulation or fork → deliberation → constitutional validation → approval → canary activation → observed postconditions → general activation or rollback

Donated compute determines whether a proposal builds, reproduces, and passes declared tests. Donated review supplies evidence and criticism. Neither compute volume nor credit balance determines whether the proposal should govern people.

34.4 Website and control-plane governance

TF-S34-R003 The public website and cloud control plane **SHOULD** be built from the governed repository through signed profiles and reproducible pipelines. Content, onboarding, API defaults, and network policies become reviewable changes with previews and rollback rather than silent administrator edits.

TF-S34-R004 Emergency maintainers **MAY** issue short-lived security interventions under a narrowly defined emergency rule. Emergency changes expire unless ratified, remain transparent, and cannot silently alter identity ownership, export, governance rights, or threshold keys.

34.5 First implementation milestone

TF-S34-R005 The initial source-generation launch **MUST** deliver a vertical slice rather than the entire future governance system:

1. preinstall project torsionfield/core;
2. implement ProblemRecord, TaskManifest, WorkOffer, WorkLease, ExecutionReceipt, ReviewAttestation, and BuildReceipt schemas;
3. add TF.work, TF.vault, TF.build, TF.verify, and TF.provenance minimum APIs;
4. implement one Debian 13 isolated source-generation runner;
5. implement encrypted task bundles and local ScriptVault key release;
6. execute tasks through participant-owned provider sessions without central credentials;
7. quarantine and scan every result;
8. require at least one independent verification for merge candidates;
9. compile and test at least the minimal extension and Torsion Node variants;
10. publish signed provenance and a canary artifact;
11. expose the contribution queue, overnight schedule, budgets, and kill switch in the side panel;
12. preserve manual rollback and an immutable last-known-good release.

TF-S34-R006 This vertical slice is the foundation for later anonymous transport, distributed Blender construction, multiple browser variants, and democratic activation. It **MUST** be architected with the final contracts even when the first deployment uses a centralized coordinator and human review.

§35

STAGED TARGET

Torsionfield semantic capability layer

Purpose. Separates stable capability meaning from movable implementations and defines evidence-driven promotion to native layers.

TF-S35-R001 35.1 Scope and non-disruption This section defines the architecture of meaning above the concrete runtime. It clarifies how Torsionfield capabilities may acquire multiple implementations and may later move downward into browser-native or operating-system-native layers. It does not replace, postpone, or broaden the direct implementation milestone. Sections 3 through 23 and section 34.5 remain the authoritative first implementation path: fork and extend the current ScriptCat baseline; implement the Router Core, Mother Script, mutation and installation transactions, provider adapters, Agent Tab Manager, Torsion Node, Debian 13 runner, signed task bundles, work donation, scanning, verification, and release rollback. No Chromium fork, Minefield-native provider, Android runtime, Arti-WASM client, standards process, or automatic capability-promotion system is required for that vertical slice. The semantic layer **MUST** be implemented initially as lightweight contracts, schemas, provider descriptors, assurance declarations, and conformance tests around the direct implementation. It **MUST NOT** create a second runtime, scheduler, permission model, operation model, or repository fork. 35.2 Capability semantics rather than implementation identity A Torsionfield capability describes an observable intention and its guarantees independently of the component currently satisfying it. Each CapabilityContract **SHOULD** define: - capability identifier and semantic version; - intent and non-goals; - input, output, streaming, and event schemas; - preconditions and postconditions; - required permissions and delegable scopes; - state transitions, cancellation, timeout, and unknown-outcome behavior; - failure families and recovery obligations; - evidence and provenance requirements; - privacy, security, locality, and resource assumptions; - assurance levels and permitted degradation; - conformance fixtures, property tests, and acceptance tests; - compatible provider bindings and version ranges. TF.* is the canonical developer binding used by the present product. It is not the ontology itself. CAT.* is the first compatibility and reference binding. A future Minefield, Android, server, peer, WASM, or alternative-browser binding can satisfy the same CapabilityContract without reproducing the same internal call path. 35.3 Provider bindings and assurance A capability provider advertises the contracts it implements, implementation revision, dependencies, resource limits, isolation boundary, evidence type, and assurance level. Initial assurance vocabulary **SHOULD** include: - emulated; - best-effort; - observed; - restorable; - isolated; - native-enforced; - attested. Different bindings are not silently equivalent. An extension may satisfy tab protection at restorable assurance by recreating a closed tab; Minefield may later satisfy the same semantic contract at native-enforced assurance by rejecting ordinary closure. A request that requires a stronger assurance than any available provider supplies **MUST** fail explicitly or obtain an explicit policy-approved degradation. The first release may expose only the ScriptCat-derived extension and Torsion Node providers. Provider multiplicity is a compatibility property of the contract model, not a launch prerequisite. 35.4 Bidirectional semantic evolution The long-term architecture permits two governed directions of evolution. Upward promotion observes repeated userscript, workflow, adapter, and agent patterns and may propose a new or refined CapabilityContract when the behavior has become stable and broadly useful. Downward promotion may replace an extension or daemon implementation with a browser-native, operating-system-native, remote, or hardware-assisted provider while preserving the observable contract and declaring any stronger guarantees. Semantic migration may also adapt an implementation when Chromium, ScriptCat, Android, Rust dependencies, provider frontends, or other upstream systems change. A semantic migration record describes the capability intention, invariants, affected subsystems, old and new insertion points, compatibility impact, and required tests rather than relying only on line-oriented patches. This evolution is conceptual and governed. It does not authorize automatic source promotion or autonomous activation. Every generated migration or native-promotion proposal remains subject to the existing Prompt Commit, quarantine, license, contract-test, independent-review, reproducible-build, canary, provenance, and rollback requirements. 35.5 Implementation preservation rule The direct implementation **MUST** continue to optimize for one working ScriptCat-first system rather than prematurely constructing a standards organization or a complete matrix of runtimes. The immediate repository additions required by this section are limited to: shared/capabilities/contracts/ shared/capabilities/schemas/ shared/capabilities/provider-manifests/ shared/capabilities/conformance/ shared/capabilities/assurance/ Existing CAT and TF implementation methods **MAY** remain where they are while these artifacts describe their semantics and map them to current providers. Refactoring a working implementation solely to resemble a hypothetical future native API is prohibited unless it removes real duplication, fixes an identified boundary defect, or is required by a conformance test. Minefield remains an optional optimized provider and experimental native target. It is not on the critical path to the first useful Torsionfield release. 35.6 Promotion criteria A behavior becomes a candidate stable capability or native primitive only when evidence demonstrates: 1. repeated use across independent scripts, workflows, agents, or projects; 2. a stable intent that is not merely one implementation convenience; 3. explicit permissions, privacy consequences, and abuse analysis; 4. testable postconditions and failure semantics; 5. a conformance suite independent of one provider; 6. measurable deficiencies in the current userscript, extension, or daemon implementation; 7. a realistic second provider or a justified native-assurance requirement; 8. compatibility, migration, and rollback paths; 9. no unnecessary dependence on Chromium-specific concepts where a portable resource model is possible. Promotion is therefore evidence-driven

optimization, not architectural speculation. The ScriptCat ecosystem remains the innovation and observation layer; native runtimes become selective implementation targets only after capabilities prove themselves. 35.7 Version 2.4 interpretation Version 2.4 preserves the committed delivery sequence while adding two conceptual layers above it. Torsionfield is understood as a semantic capability layer with movable implementations and as a participant-trust architecture for delegated authority, evidence, verification, and acceptance. The product being built first remains the integrated ScriptCat-derived extension, Torsion Node, source-generation runner, provider-adapter system, work-donation queue, and verification pipeline already specified.

§36

FIRST RELEASE

Participant trust, delegated authority, and acceptance

> **Purpose.** Makes participant trust, delegated authority, assurance, verification, and acceptance explicit and independently inspectable.

36.1 Scope and non-disruption

This section adopts the participant-trust-centred architectural decision as the governing security and acceptance model above the concrete runtime. It changes how Torsionfield interprets authority, evidence, verification, risk, and shared acceptance. It does not replace the committed ScriptCat-first implementation sequence.

TF-S36-R001 Sections 3 through 23 and section 34.5 remain the authoritative direct implementation path. The Router Core, Mother Script, provider adapters, Agent Tab Manager, Torsion Node, Debian 13 runner, signed task bundles, work-donation queue, scanning, verification, packaging, and rollback remain unchanged. The participant-trust layer **MUST** initially be implemented as additional records, policies, explanations, fixtures, and user-interface states around those components. It **MUST NOT** introduce another scheduler, operation model, permission system, or execution runtime.

36.2 Governing doctrine

Torsionfield preserves practical capability reach. A warning, default-off state, missing automatic delegation, low trust classification, or refusal by a shared release policy does not imply that the underlying capability has ceased to exist.

Warnings, isolation, permissions, profiles, and sandboxes remain supporting controls for informed participation, attribution, evidence collection, failure containment, and dispute reconstruction. They are not represented as proof that a determined participant cannot perform the same work elsewhere or through an unrestricted local profile.

The controlling security question is:

Which participant should receive which authority for which task, on the basis of what evidence, under what execution conditions, with what independent verification, and with what consequences when the work is wrong, fabricated, compromised, or malicious?

The governing practical rule is:

Nothing is silently prohibited. Nothing is silently trusted. Nothing is silently verified. Nothing is silently accepted.

36.3 Permission, trust, assurance, and acceptance are separate

Permission answers whether an operation is allowed by the current authority policy.

Participant trust estimates how reliable a participant is for a particular task class under particular conditions.

Assurance describes properties of the execution path, including enforcement, observation, isolation, recovery, identity continuity, and attestation.

Acceptance records which participant, project, audience, or release policy is willing to rely on a result despite remaining uncertainty.

TF-S36-R002 These dimensions **MUST** remain independent. A highly trusted participant may operate from a compromised device. A low-history participant may produce a strongly reproducible result. A locally permitted operation may remain unacceptable to a project release. A signed statement authenticates its signer but does not establish its truth.

36.4 Multidimensional and scoped trust

TF-S36-R003 Torsionfield **MUST NOT** use one scalar reputation score to control consequential authority.

Trust assessments are scoped by:

- participant identity;
- domain and task class;

- consequence level;
- confidentiality class;
- producer, verifier, reviewer, or governance role;
- device and execution environment;
- provider and toolchain;
- project and authority domain;
- evidence quality and calibration history;
- time window, decay, disputes, and recovery state.

A participant may be highly reliable in Rust compilation and unreliable in security review. Trust in one device does not automatically transfer to another device. Identity continuity establishes only that the same key or an approved successor returned; it does not establish competence, honesty, or current environment integrity.

TF-S36-R004 Compute contribution, wealth, popularity, task volume, or social influence **MUST NOT** automatically purchase consequential authority or governance power.

36.5 Participant-trust records

The participant-trust model augments the existing TaskManifest, AuthorityGrant, OperationRecord, EvidenceBundle, ExecutionReceipt, ReviewAttestation, and AssuranceVector with the following records.

ParticipantRecord

Describes identity continuity, declared affiliations, known devices, competence domains, role history, trust assessments, conflicts, disputes, revocations, and rehabilitation history.

DeviceEnvironmentRecord

Separately describes device identity, operating environment, runtime revisions, isolation, key custody, provider sessions, compromise events, repairs, and available attestation.

TrustAssessment

A versioned, evidence-linked assessment over one participant, domain, task class, consequence level, role, device or environment, confidence, decay policy, calibration history, and unresolved disputes. It is not an intrinsic property of the participant.

VerificationRecord

Describes the verifier, device, environment, claims examined, evidence inspected, methods, result, uncertainty, conflicts, relationship to the producer, prior exposure to the candidate, dissent, and signature.

AcceptanceDecision

Records required and achieved trust, required and achieved assurance, missing evidence, accepted uncertainty, accepting participant or policy, affected audience, exact subject digest, and acceptance state.

DisputeRecord

Preserves conflicting claims, minority reports, evidence references, appeals, later resolutions, affected trust assessments, and unresolved uncertainty.

No trust value may exist without scope, evidence references, issuer, assessment time, confidence, and decay policy.

36.6 Qualification, routing, and verification

TF-S36-R005 New participants begin with bounded authority rather than zero participation. They **MAY** contribute through low-consequence work, calibration tasks, reproducible execution, review, or explicitly local self-directed operation.

TF-S36-R006 Qualification **SHOULD** include:

- known-answer tasks;
- seeded faulty tasks;
- deliberately impossible tasks;

- prompt-injection fixtures;
- fabricated-provider-evidence fixtures;
- dependency and archive attacks;
- evidence-quality tests;
- confidence and uncertainty calibration.

TF-S36-R007 Correctly reporting impossibility, insufficient evidence, conflict of interest, or uncertainty is positive evidence. The system **MUST NOT** reward unsupported certainty merely because it resembles successful completion.

TF-S36-R008 Producer selection **SHOULD** consider domain competence, recent calibration, evidence quality, task history, confidentiality eligibility, device and provider state, unresolved disputes, availability, and consequence level.

TF-S36-R009 Verifier selection **SHOULD** additionally consider verification accuracy, independence from the producer and other verifiers, shared control domains, organization, infrastructure, model family, toolchain, prior exposure, and declared conflicts. Different keys or network addresses do not by themselves establish independence.

A verifier is a participant whose own verification history is updated by later evidence. Correct dissent and useful minority reports remain attributable even when the majority accepts another result.

36.7 Trust correction, revocation, and rehabilitation

Positive trust evidence may include correct work, accurate uncertainty, finding seeded defects, transparent correction, conflict disclosure, independent reproduction, high-quality evidence, and successful unknown-outcome reconciliation.

Negative trust evidence may include fabricated completion, copied evidence, concealed uncertainty, undisclosed shared control, repeated unsupported approval, confidentiality failure, misuse of delegated authority, or operation from a reported compromised device.

TF-S36-R010 Trust **MUST** decay or be requalified where context changes. A recent device compromise may suspend device-specific authority without erasing unrelated competence history. A failure in one domain does not automatically erase trust in another domain.

TF-S36-R011 Shared authority and active leases **MUST** be revocable. Rehabilitation **MUST** remain possible through probation, new evidence, corrected performance, and domain-limited requalification. Rehabilitation does not erase the original failure.

36.8 Consequence and acceptance lanes

TF-S36-R012 Every consequential result **MUST** distinguish at least the following states:

PRIVATE_LOCAL

The participant performs and evaluates the work privately. No project or external audience is implied.

ATTRIBUTABLE_LOCAL

The operation is bound to a participant, device, authority grant, and evidence bundle, but independent verification is not required.

DELEGATED_WORK

Another participant or project granted bounded authority under a TaskManifest and eligibility policy.

LOCALLY_ACCEPTED

A named participant accepts the result and remaining uncertainty for local use.

PROJECT_ACCEPTED

A project policy accepts the exact result digest for a declared purpose.

RELEASE_ACCEPTED

The client-enforced release policy accepts the exact artifact through the required producer, verifier, threshold, update, and rollback rules.

DISPUTED

The result remains inspectable and may be used by accepting participants, but disagreement and minority reports remain attached.

REJECTED_FOR_AUDIENCE

A specific participant, project, or release policy refuses to rely on the result. This does not imply that local execution is technically impossible.

TF-S36-R013 The interface **MUST NOT** collapse local acceptance, independent verification, project acceptance, and release acceptance into one success indicator.

36.9 Capability reach and explicit responsibility

No capability is removed from the catalog merely because it is dangerous, experimental, high-consequence, anonymous, self-modifying, provider-sensitive, or insufficiently proven.

A capability may instead be represented as:

- configured;
- provider absent;
- available;
- participant qualified;
- participant unqualified;
- weak evidence;
- experimental provider;
- elevated authority;
- unrestricted local owner;
- pseudonymous;
- anonymous low-history;
- disputed;
- locally accepted;
- project accepted;
- release accepted;
- trust degraded;
- shared authority revoked.

TF-S36-R014 An unrestricted local owner profile remains available. Its use **MUST** identify the accepting participant, scope, duration, affected providers, evidence limitations, and irreversible consequences. Local owner authority does not automatically transfer to peers, projects, verifiers, or release systems.

36.10 Direct implementation integration

TF-S36-R015 The first implementation **MUST** preserve the current repository and execution architecture.

The minimum participant-trust additions are:

```
shared/trust/schemas/
shared/trust/policies/
shared/trust/fixtures/
shared/trust/conformance/
daemon/crates/torsion-trust/
extension/src/ui/participant-console/
extension/src/ui/verification/
extension/src/ui/acceptance/
tests/trust-calibration/
tests/verifier-independence/
tests/sybil/
tests/collusion/
tests/disputes/
```

The existing Router Core remains the operation authority. The existing permission system remains the authority gate. The existing evidence and provenance systems remain the source of execution claims. The trust layer consumes those records to explain eligibility, verifier selection, acceptance, correction, and revocation; it does not execute effects itself.

The first vertical slice adds only:

1. one persistent participant identity and one separately recorded device;
2. scoped ParticipantRecord and DeviceEnvironmentRecord schemas;
3. a small calibration-fixture set;
4. one task-specific eligibility decision with an explanation;
5. one independent verifier-selection decision;
6. one VerificationRecord with uncertainty and conflict disclosure;
7. explicit local, project, and release acceptance states;
8. one dispute and minority-report path;
9. trust updates for both producer and verifier;
10. export, revocation, and rehabilitation records.

These additions extend the work-donation and verification milestone already specified. They do not delay implementation of the extension, adapters, Torsion Node, Debian runner, or source-generation pool.

36.11 Falsification and pivot conditions

The participant-trust architecture is retained only if evidence demonstrates that it improves accepted-result reliability.

TF-S36-R016 The project **MUST** radically revise the trust model when:

- task-specific trust predicts later performance no better than random assignment, static administrator selection, or simple completed-task counts;
- calibration can be cheaply copied, anticipated, or farmed;
- multiple identities under one control domain satisfy diversity rules;
- verifier-history weighting fails to improve detection of seeded defects;
- popularity, wealth, compute contribution, or task volume dominates direct competence evidence;
- participants conceal uncertainty to preserve trust;
- producer-selected verification becomes circular;
- evidence volume substitutes for evidence quality;
- local, project, and release acceptance cannot be distinguished;
- a coordinator can silently rewrite trust history or suppress disputes;
- rehabilitation is impossible and identity abandonment becomes rational;
- revocation fails to remove active delegated authority.

TF-S36-R017 A failed trust model **MUST** be replaced or simplified. The response **MUST NOT** be to hide capability reach or inflate the number of trust dimensions.

PART VI

References

§37 REFERENCE

References

Purpose. Lists the primary documentation and source projects used as implementation and interoperability references.

ScriptCat Agent overview
https://docs.scriptcat.org/en/docs/dev/agent/

ScriptCat DOM API
https://docs.scriptcat.org/docs/dev/agent/agent-dom/

ScriptCat conversation API
https://docs.scriptcat.org/docs/dev/agent/agent-conversation/

ScriptCat OPFS API
https://docs.scriptcat.org/en/docs/dev/agent/agent-opfs/

ScriptCat MCP API
https://docs.scriptcat.org/docs/dev/agent/agent-mcp/

ScriptCat background scripts
https://docs.scriptcat.org/en/docs/dev/background/

MCP-SuperAssistant https://github.com/srbhptl39/MCP-SuperAssistant

libp2p https://libp2p.io/docs/

libp2p hole punching https://libp2p.io/docs/hole-punching/

Cloudflare Durable Object WebSockets
https://developers.cloudflare.com/durable-objects/best-practices/websockets/

Cloudflare Tunnel
https://developers.cloudflare.com/tunnel/

Violentmonkey
https://github.com/violentmonkey/violentmonkey

Nanobrowser
https://github.com/nanobrowser/nanobrowser

AIPex https://github.com/AIPexStudio/open-claude-for-chrome

BrowserOS https://github.com/browseros-ai/BrowserOS

MCPMonkey https://github.com/kstrikis/MCPMonkey

Algonius Browser https://github.com/algonius/algonius-browser

Automa https://github.com/AutomaApp/automa

UI.Vision RPA https://github.com/A9T9/RPA

SingleFile https://github.com/gildas-lormeau/SingleFile

Vimium https://github.com/philc/vimium

Renovate https://docs.renovatebot.com/

Tree-sitter https://tree-sitter.github.io/

ast-grep https://ast-grep.github.io/

END OF SPECIFICATION

Torsionfield Runtime Specification v3.0

37 complete source sections · 94 indexed normative clauses · source SHA-256 `c49684e0b01487206118be9eb4f790f54f91b378e62cc59664ffb8a2f5fb739f`

Source map · Requirements manifest · Editorial loop · Torsionfield product site