
GRAYPASS

Documentation

Build, operate, and integrate with GrayPass

Your guide to evidence collection, action policies, proof verification, and workspace operations.

01 / Build with GrayPass

02 / Operate GrayPass

03 / API reference

October 2026

SDKs Browser core 1.0.1; Node 1.0.0

Canonical v1 API.

app.graypass.org/docs/

Contents

01 / Build with GrayPass

Start here

GrayPass documentation	6
Capabilities	7
Getting started	9
Sandbox quickstart	11
Explore the demonstrations	14

Concepts

Architecture	16
Concepts and lifecycle	18
Interpreting evidence	21
Glossary	23

Integration

Integrate a protected action	25
Protected actions and policies	29
Decisions, challenges, and review	31
Proofs and verification	33
Subject lifecycle	36
Webhooks	38
Optional camera evidence	41
Evaluate GrayPass	43

SDKs

Browser SDK	45
Node SDK	48

02 / Operate GrayPass

Console

Console guide	52
Configure console integrations	55

Operations

Production operation	57
Troubleshooting	60
Versions and compatibility	63

Security and privacy

Security	64
Privacy and data flow	66
Privacy, retention, and individual requests	69
Enrollment and recovery	72
Threat model	74
Endpoint security	77
Supported use and customer responsibilities	78

Account and billing

Use the Billing console	80
Commercial model	83
Usage and billing	86
Invoices and payments	89
Renewal and commercial lifecycle	92

03 / API reference

API reference	95
Authentication and API conventions	98
Errors and response handling	103
Subject API	105
Sessions and evidence API	121
Actions and policy API	128
Authorization and outcome API	137
Challenge API	144
Proof API	149
Evaluation API	152
Activity API	157
Schemas	159
Reason codes	188

PART 01

Build with GrayPass

Connect browser evidence, action policies, and proof verification to your application.

Start here

Concepts

Integration

SDKs

[START HERE](#)

GrayPass documentation

GrayPass connects behavioral evidence, action policies, and signed proofs to the consequential actions in your application. Use the browser SDK to collect evidence, your backend to request a decision, and the Node SDK to verify the exact approved request.

Choose a starting point

Start here	What you will accomplish
Run the quickstart	Enroll a subject, request a decision, and complete a protected sandbox destination change
Integrate an action	Connect evidence, policy, approval, and verification to your application
Manage a workspace	Configure credentials, origins, actions, and operational views

New to GrayPass? [Capabilities](#) introduces the platform, and [Getting started](#) walks through account access and installation.

What an integration does

1. Associate your authenticated application account with a GrayPass subject.
2. Enroll a Brainprint and collect derived interaction evidence with the browser SDK.
3. Submit an exact action request from your backend under an active policy.
4. Handle the returned decision: allow, challenge, hold, or deny.
5. For an enforce-mode allow, verify the signed Proof of Agency and consume it online.
6. Execute the approved request and report the application outcome.

Use **Sandbox** for the complete integration flow and proof-backed mock execution. Use **Live** for shadow evaluations alongside your application's existing execution path. [Action modes](#) explains the configuration for each workflow.

Find your guide

Part	Contents
Build with GrayPass	Setup, concepts, integration guides, SDKs, and evaluation
Operate GrayPass	Console, production operation, security, privacy, and account administration
API reference	Authentication, endpoints, request and response schemas, and reason codes

The [complete PDF](#) contains all three parts. Use [API reference](#) for individual operations and [Troubleshooting](#) for specific response codes and recovery steps.

[START HERE](#)

Capabilities

GrayPass brings evidence collection, policy decisions, action-bound approval, and proof verification into one integration. Your application supplies its authenticated account and proposed operation; GrayPass evaluates the bound request and returns a decision with an Assurance Profile.

Collect evidence and manage subjects

Create tenant-scoped subjects from your application accounts. Enroll a Brainprint through ordinary interaction, then use observation sessions to collect fresh evidence for subsequent actions. The browser SDK derives summaries locally and reports accepted coverage so your application can show enrollment progress.

Manage the complete [subject lifecycle](#): enrollment, adaptation freeze, revocation, recovery, export, and deletion. [Optional camera evidence](#) adds locally processed gaze summaries when your integration includes the compatible camera assets and the user enables collection.

Apply policy to an exact action

Register the action's name, consequence, resource type, request context, and audience. Activate a versioned policy to define evidence, freshness, approval, and review requirements. Each authorization returns one decision, the policy version, reasons, and the Assurance Profile used.

Workflow	Configuration	Result
Validate an integration	Sandbox test or shadow mode	Inspect decisions, request binding, and application outcomes
Exercise protected execution	Sandbox enforce mode	Verify and consume an eligible proof before applying a mock operation
Evaluate with your application	Live shadow mode	Collect policy decisions and outcome measurements alongside your existing business path

The [quickstart](#) implements the sandbox workflow. See [Actions and policies](#) for mode configuration and [Production operation](#) for rollout and monitoring.

Confirm, review, and verify

Use action-bound passkey challenges to obtain explicit approval of a saved request. Route holds to an attributed reviewer. Resolve the resulting child authorization against the original pending request before continuing.

An enforce-mode allow supplies a Proof of Agency when its requirements are satisfied. The proof binds the request to a subject, environment, audience, action, and validity interval. Your backend verifies the binding and consumes the proof online before executing the operation. [Proof verification](#) provides the code and expected claims.

Interpret the Assurance Profile

For sandbox and shadow evaluation, read operator continuity against the enrolled behavioral reference, operator-change indicators within the session, and automation indicators in the accepted observations. Each claim includes categorical status, coverage, freshness, reasons, and applicable limitations. Account authentication remains with your application. Use these categorical indicators without interpreting them as calibrated identity probabilities.

Use [Interpreting evidence](#) to understand the fields and [Evaluate GrayPass](#) to measure policy behavior for your workflow, devices, and users.

Configure and operate the integration

The Console brings credentials, browser origins, actions, policies, subjects, activity, evaluations, webhook delivery, and billing into your workspace. Browser and Node packages include prebuilt JavaScript and TypeScript definitions. Direct HTTP interfaces cover the full [API reference](#).

Start with [Getting started](#). For workspace access and integration assistance, contact founders@graypass.org.

[START HERE](#)

Getting started

Set up a GrayPass workspace, install the SDKs, and connect your first action. The reference integration changes a fictional payout destination in a sandbox, giving you a complete flow to run and inspect.

Before you begin

You need access to a GrayPass tenant and its sandbox environment, an application backend, a browser application served from an origin you control, and the JavaScript SDK archives described below. The runnable example requires Node.js 22.17 or later in the Node 22 line. Contact founders@graypass.org for account access.

Resolve the GrayPass subject from your authenticated application account and store that mapping on your backend. Your existing authentication and permissions establish who can request the operation.

Configure the sandbox

1. Sign in to the Console and confirm the tenant and **Sandbox** environment.
2. Open **Integrations** and register the application's exact origin, including scheme and port where applicable. An origin does not include a path.
3. Create an appropriately scoped sandbox credential. Use a server credential for the runtime path and an admin credential for action and policy setup, or perform setup through the Console.
4. Store the secret credential on your backend. Configure the browser to obtain its short-lived client token through an authenticated backend route.
5. Confirm the credential identity before creating resources.

TypeScript

```
import { GrayPassClient } from '@graypass/node';

const graypass = new GrayPassClient({
  apiKey: process.env.GRAYPASS_SECRET_KEY!,
  endpoint: 'https://app.graypass.org',
  keysUrl: 'https://app.graypass.org',
});
const identity = await graypass.credential();
if (identity.environment_kind !== 'sandbox') {
  throw new Error('This exercise requires a sandbox credential');
}
```

This TypeScript excerpt assumes the Node SDK is installed and GRAYPASS_SECRET_KEY is set on your backend. The expected result is a sandbox identity for your tenant. If the tenant, environment, or scope is wrong, replace the credential before creating resources. Keep the secret value out of diagnostic output.

Install the SDKs

Download these files into one directory:

- [Browser SDK core 1.0.1](#)
- [Node SDK 1.0.0](#)
- [Runnable quickstart 1.0.1](#)
- [SHA-256 checksums](#)

The browser core package supports observation and challenges. It installs as `@graypass/browser`. Camera operation has separate [asset prerequisites](#).

Verify all three archives before installing:

Shell

```
shasum -a 256 -c SHA256SUMS.txt
```

Expect an OK result for each archive. If a checksum fails, discard that copy and download it again before installing. On Linux, use `sha256sum` in place of `shasum -a 256`.

After verification succeeds, install the SDKs:

Shell

```
npm install --ignore-scripts ./graypass-node-1.0.0.tgz ./graypass-browser-core-1.0.1.tgz
```

Keep the checksums and dependency lockfile for repeatable installation. The packages contain prebuilt JavaScript and types; no GrayPass source checkout or build is needed.

Run your first flow

Continue with the [Quickstart](#) to enroll a sandbox subject, request a decision, and inspect the resulting trace. The example first leaves the mock destination unchanged in shadow mode, then offers sandbox enforcement so you can follow challenge, proof verification, consumption, and a local destination update. Its README includes cleanup instructions.

When adapting the flow, use [Integration](#) for request binding and recovery, then [Evaluate](#) to plan a shadow study with your intended users.

START HERE

Sandbox quickstart

Run a complete GrayPass integration: enroll a subject, request a policy decision, confirm an action, verify its proof, and update a fictional payout destination. The standalone example includes a browser interface, Node backend, demonstration account, and SQLite request ledger. All destination changes stay in the local example.

Prerequisites

Use Node.js 22.17 or later in the Node 22 line and a current desktop Chromium browser for the reference walkthrough. The documented automated walkthrough runs in Chromium. You also need the three downloads below and a sandbox admin credential. Request workspace access through [Getting started](#). Admin scope lets this example create its action and policy as well as request runtime decisions.

In Console, select the sandbox and register `http://localhost:3100` as an allowed origin. Download the [browser core archive](#), [Node archive](#), [example archive](#), and [checksum manifest](#). Place all four files in one working directory. The browser core package supplies the collection used in this exercise.

Verify and install

Shell

```
shasum -a 256 -c SHA256SUMS.txt
```

Expect each archive to report OK. On Linux, use `sha256sum` in place of `shasum -a 256`. If verification fails, obtain a fresh copy before continuing.

After verification succeeds, unpack the example, install the SDKs, and run its tests:

Shell

```
unzip graypass-quickstart-1.0.1.zip
cd graypass-quickstart
npm install --ignore-scripts ../graypass-node-1.0.0.tgz \
  ../graypass-browser-core-1.0.1.tgz
npm test
```

The example uses the prebuilt SDKs and `npm test` should pass after installation. Keep the supplied checksums and generated dependency lockfile.

Set `GRAYPASS_SECRET_KEY` in your shell using your approved secret-handling process, then start the example:

Shell

```
GRAYPASS_ENDPOINT='https://app.graypass.org' npm start
```

Open `http://localhost:3100`. The example checks the sandbox credential, prepares its subject and action, and serves the page on the local loopback interface. Keep `GRAYPASS_SECRET_KEY` in the backend environment throughout the exercise.

Follow the exercise

1. Start enrollment and interact normally with the local page. Read accepted coverage rather than counting seconds yourself.
2. Complete enrollment when the server reports readiness. The application moves to observation for the same subject.
3. Submit the mock destination change in shadow mode. Inspect the hypothetical decision. The example must leave its destination unchanged.
4. Use the explicitly labeled sandbox enforce control. Submit a new mock request and inspect any challenge or hold.
5. Complete the labeled simulated confirmation when offered. Use this simulation to test the request chain. The passkey workflow is covered in [Decisions](#).
6. The backend resolves the resulting authorization, verifies the independently bound request, verifies and consumes the proof online, and applies the local mock destination change once.
7. Inspect the recorded authorization and outcome, then use the subject lifecycle and local cleanup instructions in the example README.

For a hold, inspect the reason codes and follow the review path in [Decisions](#). Use [Troubleshooting](#) to resolve credential, collection, or configuration responses.

Check the result

The exercise is complete when you can inspect these results:

Step	Expected result
Verify downloads	All three archives report OK
Run package tests	The test summary reports zero failures
Enroll	Accepted coverage advances and <code>enrollment_ready</code> becomes <code>true</code>
Submit in shadow	An authorization and reasons appear; the mock destination retains its previous value
Complete an eligible enforce request	The backend verifies and consumes the proof, updates the local destination once, and records the outcome

An authorization includes fields such as `decision`, `mode`, `reason_codes`, and `trace_id`. In the Node SDK, `verification.actionHash` identifies the locally bound request. Retain those fields to follow the same operation in Console Activity.

Understand the integration

The browser receives a scoped client token and observation session, while the secret credential stays on the backend. The backend chooses the fixture subject and merchant and freezes the exact proposed destination. Its pending request stores the action binding and original authorization association. Resume requests cannot substitute another destination or child authorization.

The example includes CSRF controls and a local request ledger. When adapting it, connect your authenticated account adapter and transactional store, then implement the [crash-recovery procedure](#) for proof consumption and business execution.

Clean up

1. Select **Stop collection**, close the page, and stop the local server.
2. Reconcile any pending request in the example ledger and retain the outcome records you need.
3. Delete the sandbox subject through Console or `deleteSubject()` using its stored identifier.
4. Retire the example action and revoke its test credential when finished.
5. Remove the local `data/` directory after completing reconciliation and retention.

The archive README describes how to locate the subject identifier and inspect the local ledger.

Adapt the example

Replace the fixed demonstration account with your application's authenticated account and resource-ownership checks. Separate admin setup from runtime credentials, move the request ledger to your transactional store, and implement operational recovery before connecting a real business mutation. [Integration](#) explains those responsibilities.

For an evaluation alongside an existing application, keep its business path in place and use [shadow evaluation](#). Follow [Production operation](#) to plan monitoring, request recovery, and operational ownership.

[START HERE](#)

Explore the demonstrations

GrayPass offers a browser preview for exploring behavioral comparison and a separate protected-action demonstration. Choose the experience that matches your goal. The protected-action walkthrough uses fictional transaction details.

Your goal	Open	What you will see
Explore collection and comparison	Browser preview	Guided activity, a saved reference, and a continuity index
Inspect a protected-action flow	Protected-action demonstration	Enrollment, a fictional destination change, decisions, simulated confirmation, and proof details
Build the flow in your own application	Sandbox quickstart	A downloadable browser and backend example with an inspectable request ledger

Browser preview

Read the collection notice, then choose **Start my Brainprint**. Follow the typing, movement, and scrolling activities at your normal pace. Each activity advances when enough usable measurements have been collected. Camera collection is optional and starts only through the camera step; you can continue without it.

Choose **Create Brainprint** when the page enables it. New usable activity is then compared with the saved reference. The continuity index displays the comparison with the saved reference. It updates when fresh, usable evidence is available; action decisions are shown in the separate protected-action demonstration.

The preview also offers an optional desktop Chrome companion. Follow the page's installation and connection instructions if you choose to try it. You select the tab and start observation; camera collection stays on the enrollment page. The companion preview is separate from the canonical browser SDK integration described in this manual.

Use **Pause collection** to pause. When finished, choose **End & delete**, then **Delete my Brainprint** to remove the demo reference and invalidate its companion connections. Stop any optional camera collection as part of ending the session.

Protected-action demonstration

1. Choose **Start the experience** after reading the scope and collection information.
2. Use the typing, movement, and scrolling activities. **Create my Brainprint** becomes available when the server reports enrollment readiness.
3. Open the fictional payout-destination change and choose **Request the change**. Read the decision and reasons.
4. If a challenge appears, inspect the proposed action and choose **Confirm in this sandbox** only if you intend to continue. This control simulates the approval step for the sandbox request.
5. Inspect the resulting decision and proof details. Follow the resulting decision; any returned proof is scoped to the fictional sandbox operation.
6. Try pausing collection and requesting another change to see how the policy handles the available evidence. Inspect the returned reasons to understand the result.
7. Choose **End demo**, then **Delete my demo data** when finished.

The protected-action demonstration can also offer optional camera collection. Use its explicit enable and stop controls and follow the local calibration guidance. The ordinary action walkthrough uses the core observation flow.

From demonstration to integration

Use the [quickstart](#) for runnable customer integration code. Supply your own authenticated account flow, subject mapping, action schema, registered origin, and durable request ledger before adapting it to your application.

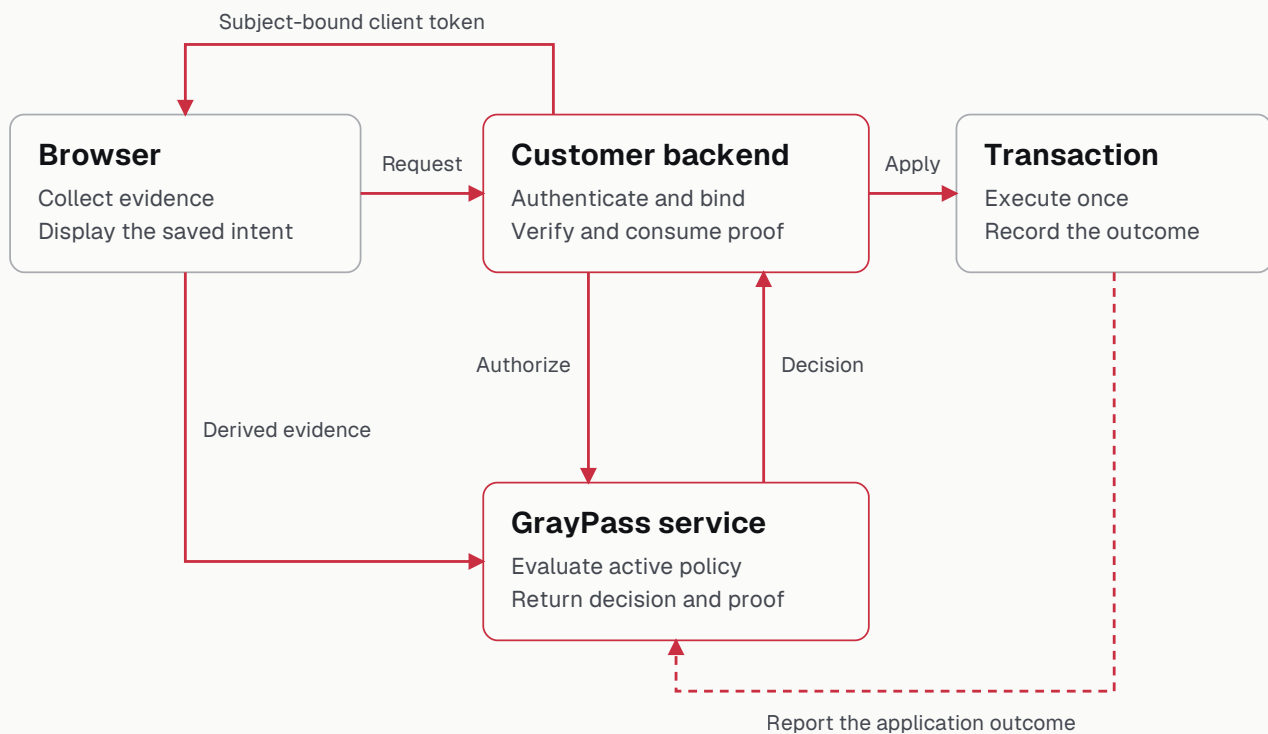
Demonstrations show behavior in a single session. To decide whether a policy suits your users, define a scoped shadow evaluation with outcome labels and known limitations using [Evaluate GrayPass](#).

CONCEPTS

Architecture

GrayPass separates evidence collection, policy decisions, and business execution. This separation lets your application bind a decision to the exact action it intends to perform, while keeping credentials and final authorization off the browser.

Components and request path



Your backend connects browser collection to a server decision, then enforces that decision at the business operation.

Component	Responsibilities	Trust boundary
Customer browser	Collect permitted derived evidence; display immutable action details; complete an action-bound challenge	Requires a trusted page and endpoint
Customer backend	Authenticate the user; resolve subject and resource; validate and freeze request material; request a decision	Holds the server credential and business authorization rules
GrayPass API	Enforce tenant and environment scope; assess evidence; apply the active policy; issue and track proofs	Supplies decisions and proof state to the customer transaction
Customer execution service	Verify expected proof bindings; consume online; execute once through its own durable state	Requires backend verification against the saved request

Component	Responsibilities	Trust boundary
Console	Configure environments, integrations, actions and policies; inspect activity and evaluations	Permissions depend on the authenticated role and environment

The browser obtains a short-lived client token from an authenticated route on your backend. That token is bound to one subject and registered origin. The SDK opens an enrollment or observation session and sends signed evidence windows. Your backend submits an action request with subject, session where available, resource, context, audience, time, nonce, and idempotency key.

The decision contains the policy version and the Assurance Profile used. An enforce-mode allow can carry a short-lived signed proof. Your backend compares the proof with request material it already trusts, checks current status and one-time consumption, and only then performs the action. Outcome reporting closes the operational loop.

Tenant and environment isolation

A tenant is the customer's administrative boundary. An environment is a distinct sandbox or live boundary inside that tenant. Credentials, subjects, sessions, actions, policies, traces, and proofs are scoped accordingly. Keep each environment's credentials, subject mapping, and pending requests together so requests and verification use the intended scope.

Sandbox supports integration exercises, including explicitly labeled simulated confirmation and proof-backed execution. Live shadow traffic evaluates real policies alongside your existing application path. Review the applicable production-shadow meter in [Usage and billing](#).

Evidence and authorization are different objects

A Brainprint is a private, tenant-scoped learned behavioral model. An Assurance Profile describes typed claims, their coverage, freshness, limitations, and version information. A decision applies a particular policy to an action request. A proof binds the allowed request for a relying party. Your application supplies the authenticated session and account permissions for this flow.

Derived evidence avoids transmission of typed characters and raw interaction paths, but it remains security-relevant data associated with a pseudonymous subject. Customer-supplied context can contain sensitive information if the customer puts it there. Prefer a digest or opaque identifier over bank details, document text, or free-form user content.

Response handling and recovery

When evidence is missing or stale, the policy selects a challenge, hold, or denial. Handle that decision through its defined workflow. For an API or verifier timeout, preserve the pending operation and reconcile its state before continuing. Complete online verification to check current revocation and consumption state.

GrayPass proof consumption and your application database are separate transactions. Your application needs its own request ledger, idempotency discipline, locking, and recovery process. Use the [integration guide](#) to implement recovery across those two transactions.

Deployment requirements

The integration described here uses the hosted GrayPass service with a browser application and customer backend. Review the [security controls](#), [threat model](#), and [commercial model](#) against your deployment requirements. Confirm any required service commitment or deployment arrangement in your agreement before depending on it.

CONCEPTS

Concepts and lifecycle

An integration connects a person in your application to evidence sessions and decisions about particular actions. Keep its identifiers separate: one subject can have many sessions, and one business request can have an initial authorization followed by a challenge or review result.

Core objects

Object	Meaning	Lifecycle consideration
Tenant	Customer administration and isolation boundary	Select explicitly when operating multiple customers
Environment	Sandbox or live scope	Resources and credentials belong to one environment
Subject	Pseudonymous person record mapped from your authenticated account	Enroll, revoke, recover, export, or delete
Brainprint	Private behavioral model associated with a subject	Supplies continuity evidence; adaptation can be frozen
Session	Enrollment or observation interval	Evidence belongs to the subject and purpose; end on exit
Evidence window	Signed derived measurements over an interval	Acceptance, replay checks, quality, and active time affect coverage
Assurance Profile	Typed evidence claims and limitations	Assessed at a time, with explicit versions and freshness
Protected action	Registered consequential operation	Stable key plus current mode and active policy
Policy version	Immutable requirements and request constraints	Create a version, then activate it; retain version identity
Authorization	Decision for one bound request	Can lead to challenge or review and a child authorization
Proof of Agency	Signed artifact for an enforceable allow	Short-lived; check scope, current status, and consumption
Outcome	What the application actually did	One immutable report per authorization

Subject lifecycle

A newly created subject is ready to begin enrollment. An enrollment session gathers permitted evidence until the server reports readiness. Your backend completes enrollment; an observation session can then supply fresh evidence for action requests. Use the server's `enrollment_ready` field to drive completion.

Revocation ends the subject's ability to continue on existing evidence and credentials. Recovery follows the subject's declared policy and requires your application to establish the appropriate anchor. Deletion removes the subject's supported operational data; billing records, security audit records, and backups have separate handling described in [Privacy](#).

Action lifecycle

Register a descriptive action key such as `payout.destination.change`. Define what resource it changes and every security-relevant request field. Configure an immutable policy and activate it. New actions begin in shadow mode. Test and shadow requests evaluate the policy without generating an enforceable proof. Enforce mode requires a proof for the exact request before execution. Exercise that path in sandbox around a mock operation before connecting a business mutation.

The customer request ledger should assign a stable logical request ID. Retrying the same request reuses the same idempotency key and request material. Changing the destination, subject, resource, or another bound value creates a different action request. A challenge preserves the original request binding through approval.

Four decisions



`allow` means the evaluated policy permitted the request. The application still checks mode, proof, account permission, and business invariants. `challenge` requires an action-bound approval step. `hold` pauses the operation for review. `deny` blocks it.

`insufficient_evidence` is a claim status and reason. Policy maps it to challenge, hold, or deny. `coverage` describes the amount and variety of accepted evidence; `freshness_ms` describes its age. Interpret both alongside the categorical claim status.

Trace and outcome lifecycle

Keep the original authorization ID with your pending business request. A completed challenge or an approved review can produce a child authorization. Resolve that child from your backend and verify its relation to the original request. Treat the browser-returned ID as a lookup reference and verify the retrieved record.

After executing or refusing the business operation, report what actually occurred. A repeated identical outcome is idempotent; a contradictory report is rejected. Define who can assign each outcome and retain the evidence for that assignment in your application. [Evaluate GrayPass](#) explains how those labels contribute to a report.

Continue with [Actions and policies](#), [Decisions and challenges](#), and [Proofs and verification](#).

CONCEPTS

Interpreting evidence

An Assurance Profile describes the evidence used for an action decision. Read claim status together with coverage, freshness, reason codes, and limitations. The active policy determines how those observations become an allow, challenge, hold, or denial.

For sandbox and shadow evaluation, operator continuity compares current behavior with the subject's behavioral reference without assigning an identity probability. Operator-change and automation signals describe the current observations. Read collection coverage, comparison results, and camera calibration quality according to their definitions, alongside your application's account authentication and business controls.

Match the conclusion to the evidence

Observation	What it tells you	What to check next
The sandbox flow completes	Enrollment, decisions, challenges, and proof handling are connected in your integration	Exercise your intended devices, users, failure conditions, and business process
A proof passes local verification	Its signature and expected request binding are valid	Check current proof state and consumption, then application permissions and transaction state
A passkey challenge completes	The authenticator ceremony confirmed the bound request	Resolve the child authorization; apply account recovery and credential ownership controls
A shadow evaluation passes its criteria	The selected traces meet the criteria recorded for that evaluation	Review sample representativeness, outcome labels, and failure cases
Camera calibration reports ready	Local screen estimation passed its measurement checks	Continue monitoring local screen-estimation quality; use account authentication for identity assurance

The canonical Assurance Profile uses categorical claims with their limitations. The browser SDK reports collection status and coverage so you can guide evidence collection; the backend requests the action decision.

Design a useful evaluation

State the decision you need to make, such as whether accepted evidence is available often enough for a proposed challenge policy. Define the workflow, population, consent basis, devices and browsers, observation period, expected missing-evidence response, and people responsible for review.

Record policy and SDK versions, sample size, exclusions, and how outcome labels were assigned. Keep calibration input separate from held-out measurements. Examine cohorts and failure cases alongside aggregate results. If the sample excludes a device type or accessibility workflow, preserve that boundary when applying the result.

Use synthetic input for repeatable integration, request-binding, and refusal tests. Use representative participants and independently reviewed outcomes when measuring how a policy behaves for your users. Customer labels such as `confirmed_attack` and `false_challenge` require a documented basis: the service records the label, while your review process establishes its meaning.

Present results clearly

Describe the measured workflow, population, observation period, criteria, and result together. For example: “For the participating desktop users, the payout-change shadow evaluation met the recorded outcome-coverage criterion.” Include challenge and hold rates, missing-evidence cases, and exclusions needed to interpret that result.

Report each criterion against the sample and property it measures. Use [Evaluate GrayPass](#) for the evaluation workflow and report fields, and the [threat model](#) for the trust assumptions to include in your assessment.

CONCEPTS

Glossary

Use these terms across the guides, Console, SDKs, and API reference. API identifiers appear in code style; product concepts use the names shown here.

Use these definitions when reading an authorization trace, configuring a policy, or discussing an integration. Billing terms link to their full contractual usage definitions below.

Term	Meaning
Action hash	Canonical digest binding the intended environment, action, subject and actor, resource, context, audience, and nonce
Active Brainprint	Commercial usage unit based on a subject's qualifying assurance within its contractual billing window; see the precise billing definition
Actor	The entity performing the request; the supported canonical actor type is human
Anchor	Customer-declared basis for associating a subject with an account; the customer verifies that association
Assurance Profile	Typed claims with status, coverage, freshness, evidence references, reason codes, limitations, and versions
Assured Action	Commercial usage unit for qualifying authorization activity, subject to exclusions and deduplication
Audience	The relying-party application or service for which an action and proof are intended
Authorization	A recorded decision for a particular bound action request
Brainprint	Private, tenant-scoped learned behavioral reference contributing continuity evidence
Challenge	Action-bound passkey confirmation or a clearly labeled sandbox simulation that reevaluates the original request
Client token	Short-lived browser credential scoped to a subject, environment, and registered origin
Consumption	Online one-time state transition that prevents the same proof from being accepted again through the consuming API
Context	Small, schema-validated request material needed to bind the consequential operation
Coverage	Quantity and modality coverage of accepted evidence; not identity confidence
Decision	Exactly one of allow, challenge, hold, or deny

Term	Meaning
Enrollment	Collection and completion process that establishes the subject's initial model under server readiness checks
Environment	Sandbox or live resource and credential boundary within a tenant
Evaluation	Time-bounded shadow study with recorded success criteria and outcomes
Evidence	Derived interaction features and integrity metadata submitted by the SDK
Freshness	Age of evidence relevant to a claim or decision
Idempotency key	Stable logical request identifier reused only for retries of the same request material
Insufficient evidence	Assurance status or reason translated by policy into challenge, hold, or denial
JWKS	Published JSON Web Key Set used to verify proof signatures
Nonce	Request-specific value bound into the action hash and protected against improper reuse
Outcome	Customer report of what actually happened after an authorization
Policy	Immutable, versioned requirements and constraints applied to a protected action
Proof of Agency	Short-lived signed artifact bound to the allowed action and relying-party audience
Protected action	Registered consequential operation with a stable key, resource type, context schema, risk class, and policy
Relying party	Customer service that verifies the proof and enforces the business operation
Research consent	Separate recorded permission for supported research use, distinct from ordinary service operation
Session	Enrollment or observation interval for a subject
Shadow mode	Policy evaluation that records hypothetical decisions without enforceable proof issuance
Subject	Tenant- and environment-scoped pseudonymous person record owning a Brainprint
Tenant	Customer administrative and data isolation boundary

Billing terms are defined in [Usage and billing](#). Security boundaries are described in [Architecture](#) and [Threat model](#).

INTEGRATION

Integrate a protected action

Build the gate around one exact payout-destination change. Start with a sandbox and a mock destination store. Run your existing authentication, account permissions, CSRF controls, and business validation before requesting GrayPass authorization.

The TypeScript snippets below illustrate the integration in stages. `pending` is your durable application record, `graypass` is a configured runtime Node client, and `controlPlane` is a separate client with an admin credential for setup. Run the [quickstart](#) for the complete application, then use these excerpts to adapt each stage.

Freeze the business request first

Resolve the merchant and GrayPass subject from your authenticated session. Parse and validate the proposed destination on the backend. Store an immutable pending request with a logical ID, authenticated owner, subject, merchant, exact destination, expected audience, and an expiry. Accept the proposed destination from the browser and resolve the subject, merchant ownership, and action key on the backend.

Bind a digest of the exact normalized destination into the GrayPass action context. The normalization used to create the digest must be the same normalization used by the mutation. Prefer an opaque identifier or keyed digest when an ordinary hash would expose a low-entropy value to guessing.

TypeScript

```
import { createHash } from 'node:crypto';

// 'pending.destination' is already validated and frozen by your backend.
const destinationFingerprint = createHash('sha256')
  .update(JSON.stringify({
    version: 1,
    destination: pending.destination,
  }))
  .digest('hex');

const resource = { type: 'merchant_account', id: pending.merchantId };
const context = { destination_fingerprint: destinationFingerprint };
```

The snippet's `pending` record represents your application's durable request state. Do not rebuild it from a later browser submission when resuming an approved action. Keep bank details out of GrayPass context and logs.

Register the action

Use an admin-scoped credential or the Console for setup. Use a server-scoped credential for runtime authorization.

TypeScript

```
const action = await controlPlane.registerAction({
  key: 'payout.destination.change',
  name: 'Change payout destination',
  consequence: 'Future payouts go to a different bank account.',
  riskClass: 'high',
  resourceType: 'merchant_account',
  contextSchema: {
    type: 'object',
    required: ['destination_fingerprint'],
    properties: {
      destination_fingerprint: {
        type: 'string', minLength: 64, maxLength: 64,
      },
    },
  },
  additionalProperties: false,
},
  allowedAudiences: ['payments-app'],
});
```

The action starts in shadow mode. Review its active policy before testing authorization. [Actions and policies](#) explains policy creation, activation, and mode configuration. Use sandbox enforce mode for the mock operation, then follow [Production operation](#) for evaluation, monitoring, and recovery.

Establish the subject and evidence session

Create or retrieve the subject on the backend with `createSubject`. Mint a short-lived client token for that subject and a registered origin using `createClientToken`. Run enrollment with `purpose: 'enroll'`; complete it from the backend only after the server reports `enrollment_ready`. Start an observation session with `purpose: 'observe'` for subsequent actions.

Your backend must associate the supplied observation session with the signed-in subject. The browser SDK exposes `observer.session`; it never exposes a secret server credential or requests a decision. Flush pending evidence before an action, then inspect accepted coverage and the policy decision.

Request the decision

TypeScript

```
const decision = await graypass.authorize({
  subject: pending.subjectId,
  session: pending.observationSessionId,
  action: 'payout.destination.change',
  resource,
  context,
  audience: 'payments-app',
  idempotencyKey: pending.id,
  requestedAt: pending.requestedAt,
});
```

Persist the authorization ID, `decision.verification.nonce`, and `decision.verification.actionHash` with the pending request. The SDK computes that hash from the submitted request and checks the server response against it. For explicit independent recomputation, use `actionHash` with the expected environment ID from your trusted configuration. Use the saved request as the authoritative source for verification.

Test and shadow decisions record the policy result for evaluation. In the sandbox example, the destination stays unchanged; in a live shadow study, your existing application path executes and supplies its outcome labels.

Challenge and resume

For challenge, send the browser the challenge ID, your opaque pending-request ID, and the immutable intent saved by your backend for that request, including the destination and merchant needed for meaningful confirmation. Use `observer.completeChallenge()` with a confirmation callback that displays that saved intent alongside `details.display`. The GrayPass display includes resource/context digests. Pair them with the exact destination saved by your backend for this request. For a passkey challenge, this callback runs before the ceremony. For a sandbox simulation, fetch `challengeDetails()` and show the saved intent yourself before calling `completeChallenge(id, { method: 'sandbox_confirm' })`; this method uses the confirmation UI you present before calling it.

On resume, authenticate the user again and look up the pending request by its server-side ownership. Resolve the returned authorization ID through `graypass.getAuthorization()`. Require it to be the original authorization or a valid child for that pending authorization, with the same subject, action, audience, request binding, environment, and expected policy context. A browser-provided child ID alone is insufficient.

The `authorize()` response carries proof as a compact string. The trace from `getAuthorization()` carries it as `trace.proof?.compact`. Preserve this distinction when handling direct allows and resumed allows.

Verify, consume, and execute

For an enforce-mode allow, use the independently stored request hash:

TypeScript

```
const expected = {
  audience: pending.audience,
  expectedAction: pending.actionKey,
  expectedActionHash: pending.actionHash,
};
const offline = await graypass.verify(proof, expected);
if (!offline.valid) throw new Error(`Proof refused: ${offline.reason}`);

const online = await graypass.verifyOnline(proof, {
  ...expected,
  consume: true,
});
if (!online.valid) throw new Error(`Proof inactive: ${online.reason}`);
```

Only after both checks succeed may your application execute the frozen destination change. Use a transaction or a downstream idempotency key to ensure your business operation runs at most once for the logical request. Serialize concurrent resumes for that pending request. Do not turn hold, deny, verifier exceptions, transport timeouts, or an ambiguous consumption response into permission.

The crash boundary

Proof consumption and your business mutation are separate transactions. A crash after consumption but before mutation can leave a consumed proof and an unexecuted operation. Your own ledger must distinguish pending, executing, completed, and uncertain attempts. If a downstream mutation may have succeeded, reconcile it using your business idempotency record. For an unexecuted operation whose proof has completed its single use, obtain fresh authorization. Associate successful consumption with the specific attempt in your ledger before execution.

The downloadable quickstart uses a durable SQLite ledger for its local mutation. For a remote business service, integrate its idempotency and reconciliation controls to coordinate proof consumption with the separately committed operation.

Report the outcome

After confirming the business result, call `reportOutcome(authorizationId, 'executed')` or the appropriate non-execution or review outcome. Save the terminal application result and its outcome report together in a durable transaction, then send reports through a retryable outbox. This also applies to `not_executed` reports from test and shadow decisions: a transport failure must not lose the evaluation outcome or restart authorization. An outcome-reporting retry must never repeat the payout change. Record customer overrides with the corresponding attributed outcome, separately from GrayPass policy allows.

Test changed destinations, wrong users, foreign environments, stale requests, repeated nonces, duplicate resumes, expired and revoked proofs, provider timeouts, and process crashes. The [Quickstart](#) supplies the runnable sandbox companion; [Production operation](#) covers rollout and recovery.

INTEGRATION

Protected actions and policies

A protected action names one consequential operation in your application. Register a stable dotted key, a human-readable name and consequence, a risk class, a resource type, and the context fields that distinguish one request from another. Each authorization uses a specific immutable policy version.

Choose the action boundary

For a destination change, bind the merchant resource and the exact destination fingerprint. For a permission grant, bind the target account, role, and relevant scope. For a sensitive export, bind the dataset, selection, recipient, and export purpose when those values determine what is approved. Validate all of these values on your backend before requesting a decision.

Give each action enough context to identify its exact destination, dataset, or other consequential values. Use integers for amounts in minor units and strings for identifiers; floating-point values are not accepted in canonical action context.

Action modes

Mode	Policy evaluated	Enforceable proof	Application behavior
test	Yes	No	Validate wiring and schema without executing the protected operation
shadow	Yes	No	Record a hypothetical decision; separately label the existing business path
enforce	Yes	On an eligible allow	Verify and consume the proof before performing the exact approved operation

New actions start in shadow mode. Configure the mode independently of the policy version through the action mode API or Console. Use sandbox enforce mode for proof verification and consumption around a mock operation, and live shadow mode for evaluation. For `error.enforcement_not_approved`, retain the current mode and contact support with the environment and trace ID.

Immutable policy versions

A policy version snapshots context schema, audiences, allowed callers, proof lifetime, assurance requirements, missing-evidence behavior, challenge methods and compensation, and hold and deny conditions. Each version has an integrity digest. A digest mismatch causes authorization to fail closed.

Create a new version for a security-relevant change, review it, then activate it. Activation retires the previous active version in the same transaction. To roll back, activate a reviewed prior version rather than editing an immutable record. Keep the action ID and policy version with deployment records, incident traces, and evaluation results.

The following illustrative policy request body requires explicit approval. Submit it to the policy-version creation operation for your action, then activate the returned version with an admin credential. The [action reference](#) lists the exact endpoints and fields:

JSON

```
{
  "requirements": {
    "claims": { "operator_continuity": "uncertain" },
    "coverage_min": "partial",
    "freshness_max_ms": 60000,
    "insufficient_evidence": "challenge",
    "challenge_methods": ["passkey"],
    "explicit_approval_required": true,
    "challenge_compensates": ["operator_continuity", "coverage", "freshness"],
    "hold_on": ["operator_change.detected", "automation.detected"],
    "deny_on": []
  },
  "allowed_audiences": ["payments-app"],
  "proof_ttl_seconds": 300,
  "context_schema": {
    "type": "object",
    "required": ["destination_fingerprint"],
    "properties": {
      "destination_fingerprint": {
        "type": "string", "minLength": 64, "maxLength": 64
      }
    }
  },
  "additionalProperties": false
}
```

Choose freshness, coverage, compensation, and proof lifetime for the workflow you are evaluating. The example values illustrate the request contract; evaluate their effect on challenges and holds before adopting them. After activation, confirm the action reports the intended active policy version. If activation conflicts with a concurrent update, re-read the action before retrying.

Context schema support

The supported schema subset includes type, properties, required, additionalProperties, enum, minLength, maxLength, minimum, maximum, items, title, and description. Types include object, string, integer, boolean, array, and null. Unsupported keywords and malformed schema values are rejected at registration. Required properties must be declared, and excessive nesting is rejected.

Use `additionalProperties: false` where the approved request shape is fixed. Schema validation checks shape and type. Your backend separately verifies merchant ownership and permission to change the destination.

Decision precedence

GrayPass validates the registered action, active policy integrity, resource type, schema, caller, audience, request freshness, idempotency, and nonce before applying policy. Deny and hold conditions take precedence over compensable missing evidence. Operator-change and automation hold conditions remain part of the policy evaluation after challenge completion.

Insufficient evidence maps to challenge, hold, or denial according to policy. When the result differs from your expectation, inspect accepted evidence, policy intent, and the resulting reasons using [Troubleshooting](#).

INTEGRATION

Decisions, challenges, and review

Every successful authorization response contains exactly one of four decisions. An HTTP success confirms that the decision was computed. Select the application path using the decision, mode, proof, and account permissions.

Decision	Meaning	Safe application response
allow	The active policy permits this bound request	In enforce mode, verify and consume its proof before execution
challenge	An action-bound approval step is required	Keep the request pending and present the approved challenge
hold	The operation needs review	Pause execution and route to an attributed reviewer
deny	The request is refused	Do not execute; explain the permissible recovery path

The Assurance Profile and reason codes explain the decision. Use the authorization decision to select the next application step.

Passkey registration and confirmation

Before using a passkey challenge, your backend may create a one-time registration grant for its authenticated subject with `createPasskeyRegistrationGrant(subjectId)`. Send that grant only through your authenticated application flow and call `observer.registerPasskey(grant.token)` in the browser. Authorize grant issuance through your account-recovery and reauthentication rules for the signed-in subject.

For a returned challenge, obtain the saved pending intent from your authenticated backend along with the challenge ID. Present that immutable intent alongside the browser SDK's challenge details in an explicit confirmation callback:

TypeScript

```
const receipt = await observer.completeChallenge(challenge.id, {
  confirm: async (details) => {
    return showAndConfirmExactAction({
      intent: pendingIntent, // Saved by the backend for this pending request.
      assurance: details.display,
    });
  },
});
```

`showAndConfirmExactAction` is your application's accessible confirmation UI. GrayPass's `details.display` contains action and consequence information, audience, and resource/context digests. Display it alongside the exact destination and intent fields saved by your backend for that pending request. Use the immutable saved values throughout confirmation. Clearly name the operation and provide explicit confirm and cancel controls.

The browser receives a completion receipt identifying the resulting authorization. Your backend must fetch that authorization, associate it with the original pending request, and perform proof verification. Use the verified authorization and proof to determine execution. See [Integration](#).

Sandbox confirmation

`sandbox_confirm` is a labeled integration-test method available only in sandbox when permitted by policy. Use it to simulate the confirmation step in an integration test. Use a passkey ceremony for authenticator-backed approval. To show a confirmation screen for this method, first fetch `observer.challengeDetails(id)` and display the backend's saved intent. Only after the user confirms should you call `observer.completeChallenge(id, { method: 'sandbox_confirm' })`. This method uses your preceding confirmation UI; the `confirm` callback belongs to the passkey path.

Use [Capabilities](#) to check the supported challenge methods for your integration.

Expiry, cancellation, and changed requests

Challenges expire; the response includes `expires_at`. Treat cancellation, expiry, unavailable credentials, and failed ceremonies as non-execution. Request fresh authorization when necessary. If the user changes the business request, create a new logical request and authorization. Do not resume the old challenge with new parameters.

The challenge reruns the original authorization with the relevant detector and policy conditions preserved. Inspect the resulting child decision, including any continuing hold.

Manual review

A hold can be reviewed in Console Activity by a security reviewer, administrator, or owner. For API-based review, use an admin credential. The review operation records an attributed approve or reject resolution and may produce a child authorization. Fetch that child and apply the same independently stored action binding and proof requirements before execution. Preserve the original subject, account, resource, and destination throughout review.

Your organization must define who can review, what evidence the reviewer sees, when another channel of confirmation is required, and how conflicts of interest are handled. GrayPass records the attributed review decision; your review process establishes the supporting evidence and decision criteria.

Outcomes and user messages

Tell the person what happened in operational language: approved, verify again, paused for review, or blocked. Avoid displaying numeric identity certainty. Offer a route to support or your account recovery process when applicable.

Report what actually happened using the outcome API. If your business system did not execute despite an allow, report `not_executed`. If a reviewer or customer used a separate authorized override process, use the relevant attributed outcome. If the final result is uncertain, reconcile the operation before assigning a terminal outcome.

INTEGRATION

Proofs and verification

A Proof of Agency is a short-lived ES256-signed JWS for one allowed action request and relying-party audience. It carries no Brainprint. It lets the execution service check the authorization's request binding, then consult current server state before using it.

Verification has two steps: validate the signature and request binding locally, then check current proof state and consume it online. Your application executes the approved operation through its own transaction controls.

Values the verifier must already know

Expected value	Trusted source
Issuer	Reviewed service configuration; hosted issuer is https://app.graypass.org
Tenant and environment ID	Your credential identity or independently configured values
Environment kind	The intended sandbox or live integration
Audience	The relying-party service configured in the active policy
Action key	The operation implemented by this route
Action hash	Locally computed from the frozen pending business request
Time	A synchronized server clock

Take expected values from trusted configuration and the saved request. For a resumed challenge, compare the child authorization with the original server-held request binding.

Compute the binding

The Node SDK's `actionHash` implements canonical request hashing. This illustrative TypeScript excerpt assumes the variables come from your configured environment and durable pending request:

TypeScript

```
import { actionHash } from '@graypass/node/verify';

const expectedActionHash = actionHash({
  environment: environmentId,
  action: 'payout.destination.change',
  resource: { type: 'merchant_account', id: merchantId },
  context: { destination_fingerprint: destinationFingerprint },
  subject: subjectId,
  actor: subjectId,
  actorType: 'human',
  audience: 'payments-app',
  nonce: pendingNonce,
});
```

All values in this example come from trusted configuration or your persisted pending request. `authorize()` also returns `verification.actionHash`, calculated locally by the SDK from the submitted request, and `verification.nonce`. Persist them before any asynchronous challenge or review.

Verify the signature and claims

TypeScript

```
const expected = {
  audience: 'payments-app',
  expectedAction: 'payout.destination.change',
  expectedActionHash,
};
const result = await graypass.verify(proof, expected);
if (!result.valid) {
  throw new Error('Execution refused: ${result.reason}');
}
```

The client checks signature, algorithm, key ID, issuer, time bounds and maximum lifetime, audience, tenant, environment, enforceability, actor constraints, action, and request hash. It resolves tenant and environment from `credential()` when they were not explicitly configured. `jwtks()` can fetch verification keys as part of this convenience method.

For a cryptographic check that makes no network request, call `verifyProof()` with an already obtained JWKS and explicit expectations. `MemoryReplayStore` tracks reuse within one process; use a shared implementation if local replay checking must span several processes. In either case, execute only after the online consumption step below.

Check current state and consume

TypeScript

```
const state = await graypass.verifyOnline(proof, {
  ...expected,
  consume: true,
});
if (!state.valid) {
  throw new Error('Execution refused: ${state.reason}');
}
```

`consume: true` accepts the proof once through a scoped conditional state change. `consume: false` returns a state snapshot. For concurrent workers, execute only in the worker whose consumption succeeds and serialize the business mutation through your application ledger.

Refuse execution if the online request fails or has an ambiguous result. A timeout may occur after consumption; inspect your own operation ledger and reconcile rather than assuming it never happened. See [Integration](#) for crash recovery.

Keys and rotation

The hosted public key set is available at `/.well-known/graypass-proof-keys.json`. The Node client caches JWKS and refreshes once on an unknown key ID. Fetch only from the configured service origin. Do not follow an untrusted proof header to an arbitrary key URL.

Cache public keys according to your integration's key-refresh policy, keep clocks synchronized, and exercise key rotation in testing. If a key cannot be resolved or a signature cannot be verified, the action remains unexecuted. Complete both signature verification and the current-state consumption check before execution.

Reading failures

Common verifier reasons include `malformed`, `unsupported_alg`, `unknown_kid`, `invalid_signature`, `issuer_mismatch`, `audience_mismatch`, `not_yet_valid`, `expired`, `ttn_too_long`, `not_enforceable`, `environment_mismatch`, `tenant_mismatch`, `actor_mismatch`, `action_mismatch`, `action_hash_mismatch`, and `replayed`. Log a reason and trace ID rather than the raw proof.

Preserve verifier expectations and correct the request, configuration, or pending-request association that caused the mismatch. The [proof API reference](#) describes state queries and online verification.

INTEGRATION

Subject lifecycle

A subject is a tenant- and environment-scoped pseudonymous record associated with a person in your application. Your application owns the mapping between its authenticated account and the GrayPass subject. Keep that mapping on the backend and verify account ownership on every lifecycle request.

Create and identify a subject

TypeScript

```
const subject = await graypass.createSubject({
  externalRef: authenticatedUser.id,
  anchor: 'customer_session',
  recoveryPolicy: 're_enroll_with_anchor',
});
```

Use a stable application identifier rather than an email address or display name when possible. GrayPass derives a scoped representation of the external reference. Treat linked metadata as pseudonymous personal data. Keep optional metadata limited to the fields needed by the integration, with credentials and payment information in their dedicated systems.

An anchor records how your application associated the subject with an account: `none`, `customer_session`, `idp`, `passkey`, or `verified_identity`. Choose the value supported by your actual account flow. Your application verifies that association and retains the evidence referenced by the declared anchor.

Enroll using server readiness

Mint a browser client token for the subject and registered origin. Start `GrayPass.observe({ token, subject: subject.id, purpose: 'enroll' })`. Let the person interact normally and inspect `observer.status()` or evidence acknowledgements. Call `completeEnrollment(subject.id, observer.session)` from your backend only when the server reports readiness.

Use the returned `enrollment_ready` flag, accepted window count, active seconds, and modalities to drive progress. A pending state means collection should continue or its rejection reasons need attention. Accepted coverage advances with valid interaction windows. Use the server readiness flag as the completion signal.

After successful backend completion, end the enrollment observer and start an observation session for later activity. If completion is refused, keep the enrollment state pending and inspect the returned reason.

Adaptation and revocation

The subject record exposes `adaptation_frozen`. The adaptation-freeze operation stops further model adaptation for that subject; use it as part of a reviewed investigation or recovery procedure. Existing evidence retains its original assessment.

Revocation invalidates continued reliance on the subject's current enrollment and associated access state. Your application should stop its observers, clear local GrayPass state, refuse pending operations that depend on the revoked subject, and require its own recovery process before re-enrollment. A signed proof can remain cryptographically well formed after lifecycle changes, which is why execution also checks online state.

TypeScript

```
await graypass.revokeSubject(subject.id);
```

Recovery and passkeys

Recovery begins through the subject recovery API, supplying an anchor and, when required, evidence reference consistent with the subject's recovery policy. The supported policies are `re_enroll_with_anchor`, `re_enroll_any`, and `manual_review`. Treat the policy as part of your account recovery design, not a substitute for it.

The ordinary recovery API does not supply an attributed manual approval. A subject configured for `manual_review` requires the authorized Console recovery workflow; a direct API request without that review is refused. Recovery starts from a revoked subject, and an anchor-based policy refuses a weaker replacement anchor.

Do not let possession of a browser token or subject ID alone authorize account recovery. Reauthenticate through your own trusted process. A passkey registration grant should likewise be issued only after an authenticated server decision. Revoke a lost or unwanted subject passkey through the supported passkey revocation operation.

Export and deletion

TypeScript

```
const exported = await graypass.exportSubject(subject.id);
await graypass.deleteSubject(subject.id);
```

The export contains subject information and bounded enrollment, authorization, session, passkey, and consent records. It indicates whether a Brainprint is stored but does not export the template or passkey credential material. Authorization, session, passkey, and consent collections each contain at most 500 records; enrollment history contains at most 100. The response has no pagination for retrieving the remainder.

For history beyond those collection bounds, contact founders@graypass.org to arrange a complete export for the individual request or review. Store the downloaded export under your own access and retention controls.

Deletion removes the subject's operational records and model data. Financial records, audit records, and backups follow their respective retention procedures. See [Privacy](#) and [Data flow](#).

After a lifecycle exercise, test that old tokens, sessions, and proofs cannot continue the protected flow. Also clean up your own subject mapping, pending business requests, local browser state, and any downloaded exports according to your policy.

INTEGRATION

Webhooks

Webhooks notify your backend about events such as authorization decisions, completed challenges, issued or revoked proofs, enrollment completion, and subject revocation. Use them to update operational views and trigger reconciliation. Protected execution uses the synchronous authorization and proof-verification flow.

Use delivery retries, dead letters, and replay controls to manage event processing. Design your receiver for duplicate and out-of-order events, and periodically reconcile against authenticated API records to maintain a complete operational view.

Register a destination

Use the Console **Integrations** view to configure a public HTTPS endpoint in the intended environment, choose relevant event subscriptions, and obtain the signing secret. Keep it in your backend's secret store. Private, loopback, link-local, and unsafe destinations are rejected, including when a delivery resolves its destination again.

Save the secret when issued and prepare a rotation procedure. The endpoint should durably accept a valid event before returning success, then process it asynchronously. Keep the request handler fast and bound its body size.

Verify the exact raw body

For new integrations require all three V2 headers: `X-GrayPass-Signature-V2`, `X-GrayPass-Timestamp`, and `X-GrayPass-Delivery-Id`. V2 signs the timestamp, delivery ID, and exact raw body bytes. It checks a default freshness tolerance of 300 seconds. Maintain a synchronized server clock.

Add this Express handler to an existing app with a configured signing secret, expected tenant and environment IDs, and your own durable inbox.

TypeScript

```
import express from 'express';
import { parseWebhook } from '@graypass/node';

app.post('/webhooks/graypass',
  express.raw({ type: 'application/json', limit: '256kb' }),
  async (req, res) => {
    const signature = req.get('X-GrayPass-Signature-V2');
    const timestamp = req.get('X-GrayPass-Timestamp');
    const deliveryId = req.get('X-GrayPass-Delivery-Id');
    if (!signature || !timestamp || !deliveryId) {
      return res.sendStatus(400);
    }
    const event = parseWebhook(
      req.body,
      req.get('X-GrayPass-Signature'),
      process.env.GRAYPASS_WEBHOOK_SECRET!,
      { signature, timestamp, deliveryId },
    );
    const eventId = event?.metadata?.event_id;
    if (!event || typeof eventId !== 'string' || !eventId) {
      return res.sendStatus(400);
    }
    if (event.tenant_id !== tenantId ||
      event.metadata?.environment !== environmentId) {
      return res.sendStatus(400);
    }
    try {
      await durableInbox.insertIfAbsent(eventId, event);
      return res.sendStatus(200);
    } catch {
      return res.sendStatus(503);
    }
  });
```

Mount this route before JSON-parsing middleware. `tenantId` and `environmentId` are the intended values from your trusted configuration or credential identity. `durableInbox` represents your own durable, unique-key event inbox. Its insert and acceptance must survive a process crash. If storage fails, return a failure response so delivery can be retried; acknowledge only after the event is durable. Validate the expected event shape and apply your business permissions before downstream processing.

The three-argument `parseWebhook` form retains body-only signature compatibility. Explicitly rejecting missing V2 headers prevents accidental downgrade to that form. Passing a whole headers object as the second argument is incorrect.

Deduplication and processing

For canonical events, use `metadata.event_id` as the business processing deduplication key; retain delivery IDs separately for diagnostics. Repeated delivery or replay must not repeat a mutation. Mark processing completion durably, and design retry behavior for failures after an external side effect.

Do not infer ordering from arrival time. Fetch the current authorization or subject state through the authenticated API when acting on lifecycle-sensitive information. Reconciliation must not depend solely on having received every event.

Deleting a subject does not purge events already queued for delivery or held as dead letters. Retain an appropriate deletion marker in your receiver and ignore stale lifecycle events for a deleted subject; never recreate a subject solely because an older webhook arrives. Apply your retention policy to the inbox and deletion marker. See [Privacy](#) for deletion boundaries.

Rotation and recovery

During a rotation overlap, a signature header may contain multiple comma-separated signatures. The SDK accepts a matching signature for the configured secret. Update your secret through the approved rotation process, test the endpoint, and avoid logging either secret or full signature material.

Delivery failures that exhaust retry handling can appear as dead letters in Console. Inspect the failure, correct the endpoint or authentication issue, and use the replay or discard control intentionally. A failed destination resolution before an initial delivery may require sending another test event after the destination is fixed.

Monitor the age of your last received event and your own reconciliation lag. An absence of events may mean no activity, a transport failure, or an enqueue gap. For a consequential action, use the synchronous decision and proof flow described in [Integration](#).

INTEGRATION

Optional camera evidence

Add coarse gaze summaries to browser observation with the optional first-party camera controller. Processing runs locally after the user enables the camera and completes calibration. Ordinary observation remains the default integration path.

Camera access is off by default and starts through an explicit user gesture. Offer a clear enable control, explain the purpose before permission is requested, and keep a visible **Stop camera** control. Keep ordinary observation available when a person declines camera access or measurements are unavailable. Choosing the non-camera path carries no camera-related penalty.

Availability and prerequisites

This guide applies to integrations with an approved compatible camera asset set. Before starting, confirm that set with founders@graypass.org, then host its worker, models, WASM runtime, and notices as described below. The public browser core download supplies ordinary observation and challenges; camera assets are obtained separately. Use the [browser SDK observation flow](#) as the base integration, then add this controller after installing the complete asset set.

Start from a user gesture

TypeScript

```
import { GrayPass } from '@graypass/browser';

const tracker = GrayPass.createEyeTracker({
  assetBase: '/gaze/',
  video: cameraPreview,
  onUpdate: (state) => renderCameraState(state),
});

const observer = await GrayPass.observe({
  token: clientToken,
  subject: subjectId,
  purpose: 'observe',
  endpoint: 'https://app.graypass.org',
  eyeTracking: tracker,
});

enableCameraButton.addEventListener('click', async () => {
  await tracker.start();
  const result = await tracker.calibrate({
    onTarget: (target) => showCalibrationTarget(target),
  });
  if (result.status !== 'ready') showCalibrationRetry();
});

stopCameraButton.addEventListener('click', () => tracker.stop());
```

This is an illustrative TypeScript excerpt. The UI functions and DOM elements belong to your application, and the token and subject come from its authenticated backend. Handle start and calibration exceptions, hide the target when calibration ends, and preserve a usable non-camera path. Camera access requires HTTPS or a browser-supported localhost secure context.

Calibration and lifecycle

Render targets using normalized viewport coordinates supplied by the tracker. Calibration fits estimates on one target set and checks them on separate targets. A completed calibration call can return `needs_calibration`; inspect `result.status`, local quality information, and guidance before displaying a ready state.

Calibration quality describes screen-estimation quality for this visit, rather than identity confidence. Viewport changes, sustained head-position changes, calibration age, or a restart can require fresh calibration. Hiding the page stops the camera; returning requires explicit enablement again.

Use a tracker during enrollment if the evaluation requires a compatible enrolled camera reference. Only one observer should consume a tracker at a time. A shared tracker has a lifecycle separate from `observer.stop()`. On pause, exit, subject deletion, or application teardown, call `tracker.stop()` or `tracker.dispose()` as appropriate and end the observer.

TypeScript

```
tracker.stop();
await observer.stop();
```

Host the complete asset directory

For an integration with the required compatible asset set, host its complete `dist/gaze/` directory. Copy the entire directory, including worker, models, WASM files, licenses, and notices, to your application's `/gaze/` route. Preserve its structure. Serve JavaScript with its normal JavaScript MIME type and WASM as `application/wasm`.

The camera directory contains the client-side processing runtime, including SIMD and non-SIMD variants selected by the worker. Keep the supplied licenses and notices with these files. Host this directory when your integration offers camera evidence.

`assetBase` must use the page's own origin. It is independent of the observer's API endpoint. A page at `https://merchant.example` must load its camera assets from that origin even when evidence is sent to `https://app.graypass.org`. Cross-origin asset directories are rejected.

The page's Content Security Policy must allow first-party scripts, `worker-src 'self'`, and API connections to the configured GrayPass endpoint. Its Permissions Policy must permit `camera=(self)`; microphone access is unnecessary. The worker response needs its own WASM-capable policy:

HTTP

```
Content-Security-Policy: default-src 'none';
  script-src 'self' 'wasm-unsafe-eval'; connect-src 'self';
  worker-src 'none'; object-src 'none';
  frame-ancestors 'none'; base-uri 'none';
```

Send the policy as one HTTP header value; the line breaks above are for readability. Keep the HTML page's script policy restricted to its required first-party scripts. Configure headers on the server serving the worker, not on the remote API host.

What leaves the device

Raw video frames, eye crops, face geometry, individual gaze paths, and calibration diagnostics remain local in this SDK flow. The observer can submit coarse dwell, transition, dispersion, and region summaries with sampling and calibration quality. The observer submits usable measurements according to collection and calibration quality. The summaries remain subject-linked derived data and require appropriate handling.

Review [Data flow](#), [Browser SDK](#), and [Troubleshooting](#) before offering camera collection to users.

INTEGRATION

Evaluate GrayPass

Evaluate your integration using accepted evidence, policy decisions, and application outcomes. Define the workflow and success criteria, collect shadow decisions, and review the report. For an agreed paid evaluation, manage the service period and deliverables through [Commercial model](#).

Define the question before collecting results

Choose one action, population, environment, observation window, and operating procedure. Record the current policy and SDK versions. Identify what remains on your existing business path and who reviews holds. Establish consent, data minimization, deletion, and incident handling before involving participants.

Before the run, choose integration criteria such as minimum shadow-decision count, maximum abstention rate, and minimum outcome coverage. Define how you will measure challenge and hold rates, investigate missing evidence, and label outcomes. Use synthetic input for repeatable integration checks and representative users for workflow evaluation.

Create the evaluation in Console **Evaluations** or through the [evaluation API](#). This illustrative request body shows the supported criteria:

JSON

```
{
  "name": "Payout protection integration check",
  "action_keys": ["payout.destination.change"],
  "success_criteria": {
    "min_shadow_decisions": 50,
    "max_abstention_rate": 0.5,
    "min_outcome_coverage": 0.5
  }
}
```

These values illustrate the API shape; choose justified criteria for your protocol. The evaluation record freezes its criteria. Optional start and end timestamps must include a timezone, and the end must follow the start.

Operate in shadow mode

Shadow evaluates the active policy and records a hypothetical decision, Assurance Profile, reasons, and customer-reported outcome. Your existing application path continues to own execution and supplies the actual outcome for each shadow decision.

Collect missing-evidence and refusal cases as well as smooth-path examples. Include slower interactions, accessibility tools, device and browser changes, session interruption, camera refusal where applicable, and recovery flows. Record known sampling limitations and changes to the application during the study.

Report what happened

TypeScript

```
await graypass.reportOutcome(authorizationId, 'executed');
```

Use the correct outcome for each authorization. Repeating an identical report is idempotent. A contradictory second outcome is rejected, and note text is reduced to a digest for this operation. Keep a separate reviewed incident record when a narrative is needed.

A label such as `confirmed_attack` or `false_challenge` is supplied by the customer. Define who may assign it and what evidence they require. Your review process establishes the evidence supporting each label. If an outcome is not yet known, resolve the application record before choosing a terminal label.

Interpret the report

The evaluation report includes selected shadow decision counts, the four-way decision distribution, insufficient-evidence rate, coverage distribution, challenges, outcome coverage and distribution, operator-change counts, latency summaries, and individual criterion results.

The result is passed whenever all recorded criteria are met. Otherwise it is `in_progress` while the study is open or `failed` after closure. An open study can therefore pass before you close it and fix the cutoff. Interpret the result using the selected traces and their outcome coverage. Review each criterion alongside the underlying reasons and cohort results; an aggregate pass can conceal a workflow with frequent missing evidence or difficult recovery.

Close and retain the result

Close the evaluation when the planned observation is complete. The cutoff is the earlier of the planned end and closing time; repeated closure preserves it. Decisions and outcomes after that cutoff are excluded. Save a reviewed report before underlying trace retention removes data you need for analysis.

Document the sample, exclusions, versions, reviewer process, incidents, and unresolved limitations alongside the result. Stop collection and delete test subjects when appropriate. For a commercial decision, review the paid evaluation deliverables and credit eligibility separately in [Commercial model](#).

Use the result to decide whether to continue, adjust the policy or integration, or stop the evaluation. Record the decision and the measurements supporting it. [Production operation](#) describes monitoring and recovery procedures for the next stage.

SDKS

Browser SDK

@graypass/browser collects permitted derived evidence and handles supported browser challenge ceremonies. Your backend requests authorization and receives the Assurance Profile; the browser receives collection status and coverage. The documented core package version is 1.0.1.

Install and start

Download the [browser core archive](#) and verify it against the [checksum manifest](#) before installing. See the [installation instructions](#).

Shell

```
npm install --ignore-scripts ./graypass-browser-core-1.0.1.tgz
```

For a bundled application, import the package. For a script-tag integration, serve the package's `dist/graypass.min.js` from your own application origin. Keep third-party notices when redistributing packaged assets. The core archive contains the observation and challenge runtime with its TypeScript definitions. Install the separate compatible assets in [Optional camera](#) when using the camera controller.

TypeScript

```
import { GrayPass } from '@graypass/browser';

const observer = await GrayPass.observe({
  token: clientToken,
  subject: subjectId,
  endpoint: 'https://app.graypass.org',
  purpose: 'observe',
  onEvidence: (ack) => renderCoverage(ack.coverage),
  onError: (error) => showEvidenceTransportError(error),
});
```

`clientToken` and `subjectId` come from your authenticated backend. The backend must create the token for the signed-in account's subject and the registered page origin. Your application supplies the rendering and error UI functions.

Observe options

Option	Behavior
token	Required short-lived subject-bound browser credential
subject	Required GrayPass subject ID associated with that token
endpoint	API base; default <code>https://api.graypass.org</code> ; configure explicitly for your environment
purpose	enroll or observe; defaults to observe
windowMs	Evidence window interval; default 5000 milliseconds

Option	Behavior
<code>timeoutMs</code>	Per-network-attempt timeout including body reading; default 10000 milliseconds
<code>refreshToken</code>	Async callback returning a replacement token after HTTP 401; the request is retried once
<code>onEvidence</code>	Receives accepted, neutral, or rejected acknowledgement and coverage
<code>onStatus</code>	Receives updated collection status
<code>onError</code>	Receives transport or shutdown errors
<code>researchConsent</code>	Requests consent-bound research behavior; server-side consent requirements still apply
<code>debug</code>	Local console diagnostics, off by default
<code>eyeTracking</code>	Optional camera controller, started through an explicit user gesture

Enable the research flag after completing the separate consent workflow in [Privacy](#).

Observer methods

Member	Purpose and important behavior
<code>session</code>	Session ID to associate with the backend's authenticated action request
<code>flush()</code>	Submit the current window; returns an acknowledgement or null when no useful frame is available
<code>status()</code>	Retrieve coverage, accepted windows, active seconds, modalities, freshness, limitations, and enrollment readiness
<code>challengeDetails(id)</code>	Retrieve public challenge display details, including action information and resource/context digests
<code>completeChallenge(id, options)</code>	Perform explicit passkey confirmation or a permitted labeled sandbox simulation
<code>cancelChallenge(id)</code>	Cancel an outstanding challenge
<code>registerPasskey(grant)</code>	Register using the one-time grant string issued by the authenticated backend
<code>enableMotion()</code>	Request permitted device-motion collection from an explicit application interaction
<code>stop()</code>	Stop collectors, finish supported session shutdown, and release observer resources

An evidence acknowledgement has `status`, `reason_codes`, `coverage`, `evidence_windows`, `active_seconds`, `modalities_present`, and `enrollment_ready`. Coverage values are `none`, `sparse`, `partial`, and `sufficient`. Use these fields to display collection progress; request action authorization through your backend.

Challenge flow

For passkeys, `completeChallenge` requires a `confirm(details)` callback that returns a boolean or promise. Display `details.display` alongside the immutable pending intent supplied by your authenticated backend, and obtain explicit consent before the ceremony. The GrayPass display contains resource/context digests. Present them with the exact destination and meaningful intent fields saved by the backend for that pending request. See the [confirmation example](#). The returned browser receipt identifies the resulting authorization. Send it to your backend, which fetches and verifies the result against its original pending request.

For sandbox-only simulation, use `{ method: 'sandbox_confirm' }` when the returned challenge offers it and the policy permits it. This method uses your application's confirmation UI before completion; the `confirm` callback belongs to the passkey path. Fetch `challengeDetails(id)`, display the backend's saved intent with a simulation label, and obtain confirmation before completing it. Keep cancellation as a non-executing path.

Collection and privacy boundaries

The SDK reads transient local key timing, pointer/touch movement, scroll and focus behavior, and supported motion samples to derive summaries. It does not transmit typed characters, clipboard contents, form values, screen contents, raw pointer paths, or raw motion streams through this evidence flow. Optional camera processing keeps images and geometry local and transmits only supported coarse summaries.

Evidence signatures enforce session integrity and replay checks. Protect the evidence-producing page, extensions, and device using the controls in [Endpoint security](#).

Teardown and errors

Stop observation when the authenticated account changes, the session ends, the subject is revoked or deleted, or the application component unmounts. If a camera tracker is shared, explicitly stop or dispose it as well; the shared controller has its own lifecycle.

Display collection status and actionable retry guidance. Complete enrollment when the server reports readiness; refresh credentials or start a new session as the response requires. The [Quickstart](#) provides a runnable installation and lifecycle example.

SDKS

Node SDK

@graypass/node requests canonical decisions, manages supported subject and action operations, verifies proofs, and supplies Express and webhook helpers. It runs on your backend with a secret credential. The documented version is 1.0.0. Download the [Node SDK archive](#), verify it with the [installation instructions](#), and install the prebuilt package:

Shell

```
npm install --ignore-scripts ./graypass-node-1.0.0.tgz
```

The runnable example targets Node.js 22.17 or later in the Node 22 line.

Client configuration

TypeScript

```
import { GrayPassClient } from '@graypass/node';

const graypass = new GrayPassClient({
  apiKey: process.env.GRAYPASS_SECRET_KEY!,
  endpoint: 'https://app.graypass.org',
  keysUrl: 'https://app.graypass.org',
  issuer: 'https://app.graypass.org',
  timeoutMs: 5000,
});
```

Set the required `apiKey` to your secret server or admin credential. The default API endpoint is `https://api.graypass.org`; the hosted app origin also serves the canonical API. `keysUrl` is the configured trusted base origin used to retrieve published proof keys. `issuer` defaults to the hosted issuer shown above.

You may independently configure `tenantId`, `environmentId`, and a `replayStore`. When either identity binding is absent, verification obtains it through `credential()`. That response is cached for the client lifetime. Public key retrieval is cached separately, with a refresh on an unknown key ID. Keep endpoint, key origin, and issuer consistent with the service configuration supplied for your environment.

Supported methods

Method	Purpose
<code>credential()</code>	Inspect tenant, environment, scope, key mode, and reported proof issuer
<code>createSubject(input)</code>	Create or retrieve a scoped subject from a stable external reference
<code>createClientToken(subject, { origin, ttlSeconds })</code>	Mint a short-lived browser token for a registered origin
<code>completeEnrollment(subject, session)</code>	Complete enrollment after the server reports readiness
<code>createPasskeyRegistrationGrant(subject)</code>	Issue a one-time grant after your authenticated account checks

Method	Purpose
<code>revokePasskey(subject, passkeyId)</code>	Remove a registered subject passkey
<code>revokeSubject(subject)</code>	Revoke the subject's existing operational reliance
<code>deleteSubject(subject)</code>	Delete supported subject operational records
<code>exportSubject(subject)</code>	Return the bounded export described in the subject guide
<code>registerAction(input)</code>	Register an action using admin scope; starts in shadow mode
<code>updateAction(id, input)</code>	Update display fields or request a mode change; handle a refused transition without changing the execution path
<code>authorize(input)</code>	Request a decision and verify the returned request hash against local input
<code>getAuthorization(id)</code>	Retrieve an authorization trace and nested proof information
<code>reportOutcome(id, outcome, note?)</code>	Record one immutable customer-reported outcome
<code>jwtks()</code>	Obtain the published proof key set
<code>verify(proof, expectations)</code>	Check signature and expected claims locally using available or fetched keys
<code>verifyOnline(proof, expectations)</code>	Check server state and optionally consume; consumption defaults to true
<code>actionHash(input)</code>	Recompute the canonical action binding

Use direct HTTP for operations without a convenience method, including policy activation, assessment retrieval, review, recovery, and evaluation management. See [API reference](#).

Configure modes using [Actions and policies](#). If the service refuses a mode transition, the action retains its previous mode. After an ambiguous timeout, retrieve the action's current state before retrying.

Authorization inputs and results

`authorize()` accepts `subject`, `action`, `resource`, `audience`, optional `context`, `session`, `human actor`, `nonce`, `idempotencyKey`, and `requestedAt`. The backend supplies authenticated identity and validated resource ownership. Context must obey the registered schema and canonical integer-safe request rules.

Supply a stable idempotency key for one logical request. If you supply that key but omit a nonce, the SDK derives a stable nonce for retries. Without a key, it generates a new request identity. Persist the original business material, nonce, and idempotency key. A transport retry may refresh `requestedAt` within the API freshness window while retaining that saved binding; it remains the same logical operation. See [request conventions](#).

The result contains `decision`, `mode`, `assurance`, `reason codes`, `challenge`, `compact proof` when applicable, `trace information`, and `verification.nonce` and `verification.actionHash`. The latter is computed locally from the submitted request. `getAuthorization()` returns a different trace shape: access its compact proof as `trace.proof?.compact`.

Express integration

`protect()` and `clientTokenRoute()` are exported by the package and the Express entry point. `clientTokenRoute()` must be mounted behind your application's authentication and use a server-resolved subject and registered origin. Your application authenticates the customer before calling the handler.

`protect()` takes the client, action key, audience, and callbacks for subject, resource, optional context, and optional idempotency key. For an enforce-mode allow, it verifies the locally bound proof and consumes it online before calling the next handler. Test and shadow modes return a non-executing response. Challenge, hold, denial, missing proof, failed verification, and unavailable dependencies do not call the protected handler.

Default responses are challenge 401, hold 202, deny 403, and unavailability 503. `onUnavailable: 'hold'` returns 202 while preserving non-execution. Use custom response callbacks to present the result; execute the protected mutation only through the verified handler. For asynchronous resumption, connect the middleware to your durable pending-request and challenge-resume ledger using the [integration guide](#).

Verification and webhooks

`@graypass/node/verify` exports `verifyProof`, `fetchJwks`, `actionHash`, and `MemoryReplayStore`, plus the relevant types. Supply explicit expected issuer, tenant, environment, audience, action, and action hash when calling the standalone verifier. Use a shared replay strategy across a fleet and complete online consumption before execution.

`parseWebhook(rawBody, signatureHeader, secret, v2Headers)` verifies the exact raw bytes and parses the event. Require the V2 signature, timestamp, and delivery ID before calling it. See the complete [webhook handler](#), including canonical event deduplication and rotation.

Errors and retries

HTTP failures raise `GrayPassApiError` with `status` and `code`; malformed or non-JSON error responses are also surfaced as errors. The client applies its configured timeout. Your application owns retry and business transaction policy. Catch failures, preserve non-execution, and retry only according to the operation's idempotency semantics.

Report an executed outcome after your application confirms execution. Retry a failed outcome report through your own outbox rather than repeating the consequential operation.

For direct `actionHash` calls, explicitly supply `actorType: 'human'`. The direct helper requires this value at runtime. `GrayPassClient.authorize()` supplies it automatically for human actions.

PART 02

Operate GrayPass

Configure your workspace, manage integrations, and run your customer operations.

Console

Operations

Security and privacy

Account and billing

CONSOLE

Console guide

Use the [GrayPass Console](#) to manage protected actions, inspect decisions, and configure your workspace.

Sign in and select scope

Before signing in, the console shows Overview and Sign in. Sign in with your invited email address or Google to reveal the workspace views and Environment selector. Your tenant membership determines which controls you can use.

Choose Sandbox or Live before changing actions, inspecting subjects, or creating credentials. Each environment keeps its data separate. Allowed browser origins and commercial agreements are tenant-wide; an action's mode is a separate setting.

If a workspace is missing, check your signed-in email and contact its owner. New personal workspaces start with a sandbox.

Find the right view

View	Task	What to check
Overview	See recent activity and work needing attention	Pending holds and setup state
Actions	Create an action or change its policy	Active policy version and action mode are separate settings
Subjects	Inspect enrollment, revoke, recover, export, or delete	Subject ID and selected environment
Activity	Inspect a decision or review a hold	Bound action, reasons, authorization chain, and reported outcome
Evaluations	Create and close a technical evaluation	Criteria, selected actions, observation period, and outcome coverage
Integrations	Configure credentials, origins, and webhooks	Credential environment and any tenant-wide setting
Settings	Configure retention and research participation	Environment scope and consent requirements
Billing	Review agreements, usage, installments, and invoices	Contractual billing window and your account permissions

Roles

Assign roles by task: developers configure integrations, security reviewers resolve holds, and administrators manage workspace controls.

Role	Intended access
Viewer	Read permitted tenant views and billing information.

Role	Intended access
Developer	Viewer access plus supported development operations such as creating actions, policy versions, subjects, and evaluations.
Security reviewer	Viewer access plus permitted outcome/review operations.
Administrator	Developer and reviewer capabilities plus administrative activation and lifecycle operations.
Owner	Administrator capabilities plus owner-only environment administration.

An owner or administrator can manage agreement drafts, the payment portal, and non-renewal. Developer, security-reviewer, and viewer memberships can inspect Billing. Contact GrayPass for invoice release and financial corrections.

For membership changes, contact your workspace owner or founders@graypass.org. If a control is unavailable, check your role before asking an authorized colleague to perform the operation under their own account.

A routine integration review

1. Select the sandbox environment and verify the expected credential scope and exact application origin.
2. Create or inspect the protected action and its policy version. Confirm the action mode separately.
3. Follow a test subject from enrollment through authorization. Inspect coverage, reasons, limitations, and the challenge/hold path.
4. Confirm that your application verifies the resulting proof and reports its outcome against the correct logical operation.
5. Inspect webhook test delivery and a deliberate failure/replay scenario.
6. Review retention and research settings before introducing real user data.
7. For a live shadow evaluation, confirm the intended users, policy, data handling, commercial terms, and [evaluation criteria](#).

The [quickstart](#) exercises the API sequence; [actions and policies](#) explains the policy model.

Review a hold

Use Activity to locate the authorization by its identifier, subject, action, and time. Review the exact pending operation in your own system as well as the GrayPass reason codes. Confirm the reviewer has the required role and an independent basis for the decision.

Submit the approve or reject resolution with the review context. Your backend must retrieve the resulting authorization and verify it against the saved request before any execution. Record the final business outcome. If the amount, destination, or account context has changed, start a new authorization instead of approving the old hold.

Retention and research settings

Settings shows each environment's retention and research participation. Retention governs decision traces, events, and derived evidence. Use subject lifecycle operations to remove customer subjects and templates. Research retention also requires deployment enablement, current per-subject consent, and an explicit opt-in on each evidence frame. Collect consent from each subject separately from environment configuration. See [research choices](#).

The usage table in Settings is diagnostic. Use Billing for the agreement's UTC anniversary windows, allowances, estimates, and invoice details. See [usage and billing](#).

When the console looks wrong

First check tenant, environment, account, and role. A 401 generally requires a valid sign-in; a 403 may reflect insufficient permission or a rejected cross-origin mutation. Confirm the live environment and commercial stage, then use that environment's credentials.

For support, record the view, UTC time, request identifier if shown, environment kind, and a minimal synthetic reproduction. Remove secrets and subject data from screenshots. See [troubleshooting](#) and [security](#).

CONSOLE

Configure console integrations

Open Console **Integrations** to create credentials, register browser origins, or configure signed webhook delivery. Confirm the workspace identity and environment in the sidebar first. An administrator or owner can manage these settings. Store issued secrets on your backend.

Create a credential

The credentials list is filtered by the selected environment: sandbox corresponds to test-mode credentials. Give each credential a name that identifies its application and use. Select the minimum supported scope.

Scope	Purpose
client	Use a backend server credential for browser-token minting and protected-action API calls.
server	Backend subject/session operations, authorization, and proofs.
admin	Control-plane work such as action registration, policy activation, and permitted reviews.

The browser SDK receives a short-lived token minted by your backend with its `server` or `admin` credential. Authenticate the user at your token endpoint, resolve the subject from server-side account data, and bind the token to the intended origin. Keep the Console-issued secret off the browser.

Plaintext credential material is shown once. Copy it directly into an appropriate secret-management workflow, verify the integration, and close the display. Do not put it in a URL, client bundle, documentation example, or screenshot. See [authentication reference](#) for accepted credential types.

Revoke and rotate deliberately

Revoking a credential retires that credential's access. Check dependent services before revocation and validate that the intended replacement is working. Emergency exposure may require immediate revocation followed by incident review.

The console's **Rotate signing secret** control concerns HMAC authorization-token signing. Rotation invalidates outstanding HMAC authorization tokens for the tenant, including tokens minted through other credentials. Manage API credentials and browser client tokens through their respective lifecycle controls. Use proof or subject revocation to invalidate ES256 Proofs of Agency.

The different credentials and signing mechanisms protect different boundaries. Follow the credential replacement and [proof lifecycle](#) procedures appropriate to the incident.

Register exact browser origins

An origin includes scheme, host, and non-default port; it does not include a path. Register the exact application origin that will collect evidence. For example, `https://merchant.example` and `https://admin.merchant.example` are different origins, and an explicit local development port is distinct from another port.

Origins are shared across the tenant. Removing an origin may affect more than the environment currently selected in the console. Inventory applications before changing the list. Production collection requires the allowed secure application origin; follow the [quickstart](#) for the supported localhost development arrangement.

Authenticate the application session on your backend before issuing a subject-bound token for the registered origin. Keep the server credential in the backend secret store.

Add a webhook destination

Configure a public HTTPS receiver you control, choose the event subscriptions it needs, and select whether it receives sandbox, live, or both environments. The page lists tenant destinations with their event environment. Use separate receivers or explicit environment checks when that helps isolate development from production processing.

Copy the one-time webhook signing secret to the receiver's secret store. Implement raw-body V2 signature verification, all required signature headers, timestamp tolerance, and durable event deduplication before acting on events. The [webhook guide](#) includes the exact verification contract.

Choose **Send test event**, then check that your receiver accepted the signature and stored the event. If it fails, inspect delivery errors before retrying. Treat test events as diagnostics in the receiver so they cannot trigger business effects.

Inspect delivery and recover failures

Deliveries in flight shows queued work, attempt count, next attempt, and the latest error. Successfully delivered events leave that list. A delivery that exhausts its retries is parked in **Dead letters**, where permitted operators can replay or discard it.

Before replaying, fix the receiving endpoint and check whether the event already produced an effect. A timeout may occur after your receiver committed its work. Replays therefore require the same durable idempotency handling as ordinary duplicate deliveries.

Replays are audited and signed with the current webhook secret. After a secret rotation, deploy the expected verification configuration before replaying old events. Account for the supported overlap/rotation behavior without accepting unsigned or stale requests.

Discarding a dead letter records the decision to stop that delivery attempt; it does not undo customer-side work. Preserve the event reference and reason in your operational process when the event matters to an investigation.

Reconcile important activity

A recorded event can lack a queued delivery, and deliveries can arrive repeatedly or out of order. Reconcile important asynchronous workflows against authenticated activity records. The [webhook guide](#) covers deduplication, freshness, and receiver recovery. Continue using synchronous authorization and proof verification for protected execution.

OPERATIONS

Production operation

Configure, monitor, and operate your hosted GrayPass integration. Assign owners for evidence collection, policy administration, request handling, and outcome reporting. Use live shadow mode alongside your existing execution path; exercise proof-backed execution in your sandbox.

Keep configuration explicit

Record the tenant, environment, API origin, proof issuer, audience, SDK archive versions, action key, active policy version, registered browser origins, and webhook endpoint. Separate sandbox and live secrets and application state. Resolve subject and merchant ownership from the authenticated application session.

Give runtime services only the credential scope they need. Keep action and policy administration in a controlled setup path. Rotate credentials through a reviewed process and verify the new credential identity before withdrawing the old one. Never log secret keys, browser tokens, raw proofs, full request bodies containing sensitive material, or webhook secrets.

Roll out in stages

1. Complete sandbox schema, challenge, proof, replay, lifecycle, and failure testing around a mock business operation.
2. Review the data flow, user messaging, account recovery, optional evidence consent, and operating responsibilities.
3. Run an agreed shadow evaluation alongside the existing application path. Label actual outcomes accurately and budget for applicable production usage.
4. Review evidence coverage, decision distribution, support needs, and the evaluated sample.
5. Record the evaluated configuration and your security and business owners' rollout decision. Confirm the action's effective mode after each configuration change; if the API rejects a change, keep the previous mode and investigate the returned code.

Changing policy requirements is a versioned configuration change. Review and activate a new version, observe its results, and preserve a rollback target. Investigate the evidence and risk behind challenge rates before approving a policy change.

Monitor the application boundary

Signal	Why it matters	Operational response
API errors and timeouts	Decisions may be unavailable	Refuse or hold; investigate dependency health
Rejected or stale evidence	Coverage may not satisfy policy	Check origin, token, transport, timing, and browser lifecycle
Challenge and hold rates	User friction or policy mismatch	Inspect reasons and evaluated populations
Proof verification failures	Binding, time, key, or scope problem	Stop execution and preserve a safe diagnostic trace

Signal	Why it matters	Operational response
Online consumption ambiguity	Business operation may need reconciliation	Use your durable request ledger
Outcome-report lag	Evaluation data may be incomplete	Retry reports without replaying business mutations
Webhook and reconciliation lag	Notifications may be delayed or missing	Inspect delivery state and fetch authoritative records
Usage and invoice differences	Commercial accounting may need review	Compare the agreed billing window and meter definitions

Set alert thresholds from your application requirements and observed behavior. Confirm any contractual service commitment with GrayPass and record the incident contact and escalation procedure with your operators.

Failure and retry policy

Retry transient transport failures with bounded exponential backoff and jitter. Preserve the same logical request, idempotency key, nonce, and security-relevant payload. A changed request requires a new key. Resolve policy, schema, credential, or ownership errors before retrying.

If an idempotent request is still in progress, wait before retrying. If its stored result became stale because the subject, action, or policy changed, create a newly reviewed request. A consumed proof has completed its single use. Timeouts after consumption need business-state reconciliation.

Maintain synchronized clocks for request freshness, proof validity, and webhook freshness. Keep a supported TLS configuration and restrict egress to reviewed service origins. Generate the authorization timestamp from the backend's synchronized clock.

Incident response and recovery

For a suspected compromised customer account, suspend affected business operations through your application, revoke the subject as appropriate, stop active observers, and invalidate your own sessions. Review pending requests and proofs. Use your established account recovery process before re-enrollment or passkey registration.

For a leaked server credential, revoke and replace it, inspect its activity and configuration changes, and rotate related application secrets according to your incident scope. Coordinate GrayPass subject revocation with your application's own login-session revocation.

Preserve trace IDs, timestamps, environment, stable error codes, versions, and sanitized descriptions of expected versus observed behavior. Contact founders@graypass.org for integration assistance; use the security reporting process in [Security](#) for vulnerabilities.

Data and commercial operation

Assign an owner and schedule for customer-side deletion and export requests, review retention settings, and distinguish subject lifecycle from billing records. Match each export request to the endpoint's response bounds and arrange additional history through the agreed privacy process. Keep optional research consent separate from ordinary service use. Retain a suitable deletion marker in asynchronous receivers so a delayed event cannot recreate a deleted subject. Queued webhook deliveries and dead letters follow a separate delivery lifecycle; follow [Webhooks](#) when handling them.

Use the contractual billing window in Console Billing when reviewing usage. Assessment retrieval can record qualifying assurance activity, so avoid using session-assessment requests as health probes. Use service health or a bounded operational check instead. Disputes, corrections, invoices, payment issues, renewal, and termination follow [Usage and billing](#), [Invoices and payments](#), and [Renewal and lifecycle](#).

OPERATIONS

Troubleshooting

Start with the environment, operation, stable error code, and authorization trace ID. A decision such as `hold` selects the review path. Keep its request pending while you inspect the reason and complete the next step.

Canonical API errors use problem documents with a machine-readable code. The Node client exposes `GrayPassApiError.status` and `.code`; match these rather than human-readable detail text. Full request and response contracts are in the [API reference](#).

Credentials and isolation

Unauthorized request

Likely cause: Missing, expired, revoked, or inappropriate credential

Next step: Check the active credential and its scope without logging the secret

`error.not_found` **for a known identifier**

Likely cause: Wrong tenant or resource boundary

Next step: Confirm the authenticated tenant and identifier source

`error.environment_mismatch`

Likely cause: Sandbox and live values mixed

Next step: Match the credential, subject, action, and requested environment

`error.browser_credential_not_allowed`

Likely cause: Browser token used on a server operation

Next step: Move authorization or subject administration to the backend

`error.caller_not_allowed`

Likely cause: Credential not permitted by the active policy or request chain

Next step: Review allowed callers and use the expected authenticated backend

Origin rejected

Likely cause: Scheme, host, or port differs from registration

Next step: Register the intended exact origin and mint its token

Cross-tenant resources can appear absent. Retrieve the identifier from your own authenticated application mapping and verify the credential identity before retrying.

Enrollment and evidence

If readiness never becomes true, inspect accepted window count, active seconds, modalities, and rejection reasons. Elapsed wall-clock time is not coverage. `evidence.rejected_unsigned` indicates missing or invalid signing; `evidence.rejected_replay` indicates repeated or overlapping counted input; `evidence.rejected_quality` indicates integrity or quality rejection.

When the SDK reports paused transport, inspect connectivity, registered origin, token expiry, page lifecycle, and whether the session ended. Use the supported refresh-token callback if your application mints replacement tokens. Do not keep submitting evidence after revocation or teardown.

`error.subject_not_enrolled` requires completed server-side enrollment, not a browser flag.

`error.subject_not_found` may indicate an incorrect environment. Re-enrollment and recovery should follow the declared subject policy.

Actions, policies, and request binding

Code	Meaning and response
<code>error.action_not_registered</code> or <code>error.action_archived</code>	Check the exact action key and environment; register or select an active action
<code>error.policy_inactive</code>	Create and activate an intact policy version
<code>error.policy_conflict</code>	A concurrent change raced with the request; re-read before retrying
<code>error.policy_integrity</code>	Stored policy no longer matches its digest; keep the gate closed and report the trace
<code>error.context_invalid</code>	Validate the supported schema subset and request field values
<code>error.audience_not_allowed</code>	Use a relying-party audience allowed by the active policy
<code>error.enforcement_not_approved</code>	The requested enforcement configuration was rejected; preserve the existing mode and contact support with the environment and trace ID
<code>error.request_stale</code>	Correct server clock or create a fresh request
<code>error.nonce_reused</code>	A nonce was reused outside its allowed idempotent request
<code>error.idempotency_conflict</code>	The same key was used for different request material
<code>error.request_in_progress</code>	Wait and retry the same request
<code>error.idempotency_stale</code>	Subject, action, or policy state changed; request fresh authorization

Do not fix request-binding errors by omitting the destination or dropping verifier checks. Preserve the full approved operation.

Repeated challenges or holds

Inspect coverage, freshness, session association, and typed claim requirements. `decision.insufficient_evidence` can correctly map to challenge or hold. Confirm an observation session for the same subject is active and use the actual accepted evidence status.

An `operator_change.detected` or `automation.detected` hold survives a challenge when the policy requires review. Scripted input may trigger automation signals. Keep the operation pending for an attributed review, or stop it. If the result is unexpected, retain the authorization ID and the accepted-evidence status for investigation.

An expired challenge requires a new authorization. `error.challenge_mismatch` can indicate a wrong request, duplicate completion, or disallowed method. `sandbox_confirm` is permitted only in sandbox and is not a passkey ceremony.

Proof failures

For `issuer_mismatch`, check the explicitly configured expected issuer. For `unknown_kid`, refresh JWKS from the configured GrayPass origin. For audience, tenant, environment, or action-hash mismatch, compare against your trusted pending request and credential identity. Never fix a mismatch by copying the proof's own claim into the expected value.

For `expired`, obtain fresh authorization. For `not_enforceable`, inspect mode and actor. For replay, revocation, or consumption failure, refuse execution and reconcile your request ledger. A successful offline signature check cannot override online refusal.

`error.proof_unavailable` means an enforceable proof could not be issued. Keep the operation unexecuted. A dependency timeout may be ambiguous; do not assume a business mutation or proof consumption failed without checking durable state.

Optional camera

If permission is refused, preserve the non-camera path. If models fail to load, inspect first-party `assetBase`, complete worker/model/WASM files, MIME types, page worker permission, and the worker response's WASM policy. If calibration does not become ready, use local guidance and let the person retry or stop. A hidden page stops the camera and a restart requires fresh calibration. See [Optional camera](#).

Webhooks and billing

For signature failures, retain exact raw bytes, require all V2 headers, confirm the correct signing secret, and check clock skew. Use the logical canonical `metadata.event_id` for deduplication. Inspect dead letters and reconcile authoritative state; missing notifications do not imply missing activity.

For billing differences, compare meter definitions, qualifying production activity, contractual window boundaries, allowances, corrections, and credits. Commercial access errors are not authorization allows. See [Invoices and payments](#).

Send a useful support report

Send the environment, timestamp and timezone, code, safe trace identifier, SDK and policy versions, reproduction steps, and expected versus observed behavior to founders@graypass.org. Redact secrets, raw proofs, payment details, and personal data. Your application's request ledger can establish whether the business operation ran; do not infer execution only from a GrayPass decision.

OPERATIONS

Versions and compatibility

This manual documents the canonical `/api/v1` API, contract version `2026-10-01`, and browser core SDK version `1.0.1` and Node SDK version `1.0.0`. The API version identifies the service contract. The [website](#) provides the current documentation.

Inspect the service you use

`GET /api/v1/meta` returns the service API version and capability information without a customer credential. Your backend can call `GET /api/v1/credential`, or the Node client's `credential()`, to check its tenant, environment, scope, and proof issuer. Compare those values with your integration configuration before creating resources or diagnosing a mismatch.

A sandbox key, subject, action, and proof belong to the same environment. Create the corresponding resources and credentials separately in each environment. Keep separate configuration and application state for sandbox and live use.

Install compatible packages

Follow [Getting started](#) to obtain the versioned JavaScript SDK archives and their checksums. They contain prebuilt JavaScript and TypeScript definitions. Keep the supplied checksums and your dependency lockfile so the same package bytes can be installed again.

Use the documented methods in the delivered package. For operations such as policy activation and evaluation management, use the direct HTTP interfaces described in the [API reference](#). The [SDK guides](#) describe available client methods and verifier behavior.

Upgrade procedure

1. Review the new version's interface changes against the operations your application uses.
2. Obtain the archives and matching checksums, verify each archive, and update the dependency lockfile.
3. Run the [sandbox quickstart](#) and your application's request-binding, challenge, replay, expiry, and lifecycle checks.
4. If you already use camera evidence, confirm a compatible approved camera asset set separately before upgrading. Use the camera package's `dist/gaze/` directory and preserve its complete runtime and notices; verify worker and WASM loading from your own origin.
5. Record the API contract, package versions, active policy version, and application version together. Retain a compatible rollback configuration.
6. Roll out through your normal process and watch API errors, refusal reasons, and outcome-report lag.

For a changed response after an upgrade, check the returned code and expected binding before retrying. Keep the operation pending during investigation; report the contract and package versions with a sanitized reproduction to founders@graypass.org.

Read a saved edition

The PDF and website contain the same chapters and reference material. Match a saved edition to `/api/v1/meta` and your installed SDK versions. Obtain customer-specific agreements and security review material through your GrayPass contact.

SECURITY AND PRIVACY

Security

GrayPass adds action-specific assurance to an application that already authenticates and authorizes its users. The customer backend decides which business operation is being attempted, requests a GrayPass decision, verifies any resulting proof, and controls execution. Execution follows the verified authorization for the saved request.

Read this chapter with the [threat model](#) and [proof verification guide](#). Keep your existing authentication, authorization, and transaction controls in place: GrayPass assurance adds evidence at the protected action boundary.

Credentials and scope

Credential or identity	Intended holder	Security responsibility
Server API credential	Customer backend	Keep in a secret store; use only for its tenant and environment.
Administrative credential	Restricted control plane	Use for action, policy, and lifecycle administration; separate from ordinary application execution.
Browser client token	One authenticated subject in an allowed origin	Obtain from your backend after checking the application session; keep short-lived and avoid persistent browser storage.
Console membership	Named administrator or reviewer	Grant the role needed for the person's work and review access when responsibilities change.
Webhook signing secret	Customer webhook receiver	Verify the exact request bytes, freshness, and duplicate event handling before processing.

Do not embed a server or administrative credential in a page, mobile package, screenshot, support ticket, or source repository. Browser tokens are constrained credentials; they are still sensitive. A browser token can submit evidence, obtain bounded status, and perform permitted challenge ceremonies. It cannot request an authorization or obtain a canonical Proof of Agency.

Tenant and environment checks apply to data access. Sandbox and live environments have separate identifiers and proof-signing keys. An application must verify both the environment kind and the expected tenant/environment identifiers; the word `live` by itself does not identify your environment.

Action and proof integrity

A Proof of Agency is an ES256 signed JWS. It binds a decision to the intended audience, subject, action, resource, context, actor where applicable, nonce, and validity interval. Build the action description from server-authoritative records. Include every value that could change the consequence of the operation, such as the exact amount and destination fingerprint.

Before executing an operation:

1. Require a final `allow` decision in the intended enforcement mode.
2. Verify the signature using the expected issuer's keys and a supported algorithm.

3. Check issuer, audience, tenant, environment, subject, action, nonce, and time claims.
4. Compare the proof's action hash with the hash of your stored, independent pending request.
5. Verify current proof state online and consume the proof before execution.
6. Commit the business operation through your own durable idempotency and transaction controls.

Online verification checks current revocation and consumption state. Use your application ledger to recover across proof consumption and the business commit, following [Integration](#).

Evidence and endpoint trust

The browser SDK summarizes interaction locally. Evidence frames are authenticated to their session and checked for replay and malformed values. Session authentication establishes possession of the session key. Pair it with the endpoint controls described in [Endpoint security](#).

Continuity compares session behavior with the enrolled Brainprint; change indicators describe changes within the observed session. Interpret these through the typed Assurance Profile, including evidence quality, freshness, and limitations. Your account controls establish enrollment identity, and your device controls establish endpoint trust. Passkeys add a verified authenticator ceremony for the bound challenge. Review [Endpoint security](#) when selecting the transaction-confirmation path.

Encryption and operational boundaries

The hosted service uses HTTPS. Stored templates and certain recoverable proof/request material are encrypted at the application layer. Proof signing uses managed asymmetric keys, with separate sandbox and live keys.

Apply access control, context minimization, and proof verification alongside encryption. GrayPass processes the evidence and request fields needed to produce a decision. Your backend and any system that receives a proof also need appropriate retention and access policies.

Confirm availability, support, backup handling, and recovery commitments during procurement; the applicable agreement defines those commitments. Request supporting material for the specific controls your security review requires.

Safe operation

Use separate credentials and application configuration for development and production. Rotate exposed credentials immediately, then examine action and administrative activity. A credential rotation does not replace subject revocation, proof-status checks, or webhook-secret rotation: each protects a different boundary.

Keep action policy changes under change control. Test insufficient evidence, timeouts, expired proofs, subject revocation, and changed transaction payloads. An unavailable assurance service must lead to the explicit customer-defined safe path, such as retry, hold, or another approved verification process; it must not silently execute the protected operation.

Vulnerability reporting and review material

Report suspected vulnerabilities privately to founders@graypass.org. Include the affected surface, approximate time, a synthetic reproduction, and the impact you observed. Do not send credentials, private keys, raw behavioral data, or customer records.

For a security questionnaire or a request for supporting evidence, contact founders@graypass.org with the controls you need to assess and the intended reviewers. The scope and permitted recipients of controlled security material are agreed separately.

SECURITY AND PRIVACY

Privacy and data flow

This chapter follows the data needed for an action from the customer's application through GrayPass and back to the customer's backend. It distinguishes browser evidence from the fields the customer independently submits. Use the [privacy chapter](#) for retention and individual requests.

The action journey

Step	Sender and recipient	Information crossing the boundary
1. Authenticate	User and customer application	Your existing login, session, and account controls establish the account used by the integration.
2. Associate a subject	Customer backend to GrayPass	A customer-selected external reference and optional minimized metadata; GrayPass returns an opaque subject identifier.
3. Issue a browser token	Customer backend to authenticated browser	A short-lived token bound to the intended subject and allowed origin.
4. Observe interaction	Browser SDK to GrayPass	Bounded derived feature windows, session and integrity metadata, and optional coarse camera-derived summaries when enabled.
5. Request a decision	Customer backend to GrayPass	Subject, action, resource, context, audience, nonce, and idempotency identity.
6. Return a result	GrayPass to customer backend	Decision, typed Assurance Profile, reasons and limitations, and a proof or challenge where applicable.
7. Verify and execute	Customer backend and GrayPass	Proof status/consumption request, then the customer's own business transaction and outcome report.
8. Notify asynchronously	GrayPass to configured customer receiver	Selected event data through signed webhooks, alongside the synchronous authorization result used for execution.

The customer's backend maintains the account-to-subject mapping and the authoritative transaction record. Reconcile browser-supplied identifiers and transaction details with that authenticated session and saved record before proceeding.

Webhook destinations become additional recipients of event data. Apply access and retention controls, deduplicate deliveries, and reconcile lifecycle state through the API. Use the export process for an audit request. Recipient copies and previously queued deliveries follow their own retention and delivery lifecycles after subject deletion. See [webhooks](#) and [individual requests](#).

Browser-derived evidence

The SDK computes summary statistics about interaction timing and movement. It does not transmit typed characters, clipboard contents, page text, form values, screen contents, audio, raw input event streams, or raw pointer paths as behavioral evidence. Keyboard handling may inspect physical key codes locally to derive timing and hand-class summaries; the codes and typed characters are not transmitted as evidence.

Evidence summaries are pseudonymous, subject-linked data. Protect their derived features, session identifiers, timestamps, and network metadata with the applicable access and retention controls.

Camera processing is optional and off by default. After an explicit choice and browser permission, the SDK processes images and face geometry on the device. Coarse gaze summaries and measured quality may enter the evidence window. Images, eye crops, face geometry, and raw gaze paths stay local. Calibration lasts for the current camera session. See [optional camera evidence](#) for consent, stopping, asset hosting, and quality handling.

Customer-supplied information

Choose the fields sent through subject metadata, resource identifiers, and action context deliberately. Apply the same data minimization to your application logs.

Use opaque identifiers and the transaction facts needed by the policy. A destination fingerprint binds an exact payment destination while keeping the full bank record in your system. Treat a fingerprint linked to a person as pseudonymous personal data. Keep free text, credentials, full documents, payment-card data, and unrelated business records in their dedicated systems.

Anchor evidence references record the customer's enrollment provenance. Your integration verifies the identity-provider assertion or identity-verification result and retains the evidence associated with that reference.

Processing and storage

GrayPass associates evidence and decisions with tenant- and environment-scoped subjects. An environment-scoped keyed hash transforms the external reference, preserving separation between customer environments.

Processing produces an enrollment template when enrollment succeeds and typed assurance when requested. The browser credential receives coverage and status. Authorization results and proofs belong to the customer backend, which decides what to show to the user.

Retention differs by record type. Decision traces, evidence-derived artifacts, sessions, events, and proof records follow environment retention rules. Subjects and templates persist until subject lifecycle operations remove them. Billing, administrative audit, and backup records have separate purposes and lifecycles. Use [privacy](#) to coordinate retention and erasure procedures.

Other service data

Console authentication processes administrator identity and session information. Administrative activity supports access control and incident investigation. Logs use redaction and identifier fingerprinting in relevant paths, while audit records may intentionally retain identifiers. Handle audit identifiers under the applicable access and retention controls.

Commercial operation adds contracting contacts, invoice details, payment status, and usage totals. The configured payment service handles payment collection separately from browser evidence. Usage totals follow the contractual meter definitions in [usage and billing](#).

Research is a separate path

Research participation is off by default and is separate from the evidence processing needed to provide the service. Research retention requires all of these conditions:

1. Research retention is enabled for the GrayPass deployment.
2. Your environment permits research participation.
3. The subject has a current granted consent receipt for the research disclosure.
4. The submitted evidence frame explicitly opts in. The browser SDK's `researchConsent: true` option sends `research_optin: true` in each frame body.

The session response's `research_enabled` field reports environment and subject eligibility when the session starts. Storage is evaluated separately for each write against all four conditions above. Per-frame opt-in uses the JSON body field; request headers are independent of that choice.

Withdrawal records a revocation and stops future research writes. Historical donations remain until deletion; ordinary product processing continues for the service your application requests. Use the [research and deletion procedures](#) for those choices. Collect camera permission separately.

Research records are pseudonymous personal data. Apply the relevant disclosure, access, and retention controls, and collect each person's consent independently of environment settings.

Deployment and procurement questions

The hosted service runs in a United States cloud region. Review the applicable agreement and data-processing materials for the deployment, parties, subprocessors, transfer arrangements, and retention commitments that apply to your use.

Send review requests to founders@graypass.org with the scope and deployment details. Agree a secure transfer process before sharing confidential records.

SECURITY AND PRIVACY

Privacy, retention, and individual requests

GrayPass processes derived interaction evidence for action-specific assurance, without inferring health, disability, emotion, demographics, productivity, or unrelated traits. This chapter describes data handling and integration responsibilities; your executed agreements and applicable privacy notices govern the legal terms of processing.

Before collecting evidence, explain its purpose to users and provide a recovery and support route. Integrate GrayPass alongside your authentication with transparent, purpose-specific collection. See [supported use and customer responsibilities](#).

Minimize collection at the integration boundary

The browser SDK summarizes how interaction occurred, without uploading the typed content or raw event streams as behavioral evidence. Optional camera processing is local; only coarse derived summaries are sent when enabled. These properties apply to the SDK evidence path. Your backend can separately transmit sensitive data in subject metadata or action context, so review those fields independently.

Use an opaque customer reference where practical. Store your account-to-subject mapping in your own authorized system and scope identifiers to their customer, environment, site, and purpose.

Retention is record-specific

Record category	Product behavior	Integration consequence
Evidence records, decision traces, events, outcomes, ended session records, expired proof/challenge records	Environment retention applies; sandbox defaults to one day and live to 90 days. Different categories age from their creation, decision, end, or expiry timestamp.	Preserve the operational evidence you legitimately need before it expires, under your own retention policy.
Customer subject and Brainprint template	Persist beyond trace retention until subject lifecycle operations remove them.	Use subject deletion to erase the subject and template.
Public-demo subjects	Ephemeral, with a one-day environment lifecycle.	Configure customer-environment retention separately.
Research donations	Separate consent-scoped records; subject deletion removes associated donations.	Keep product processing and research participation distinct.
Webhook delivery records and recipient copies	Separate from the subject's canonical event history.	Account for queued deliveries, retry records, and your receiver's retained copies.
Administrative audit, billing, and backups	Separate records and lifecycles.	Apply their agreed retention and erasure procedures separately from subject deletion.

Retention is applied by a periodic sweep, rather than at the exact instant each record reaches its age limit. The sweep also stops abandoned sessions. Use environment settings for operational retention and the applicable agreement for contractual retention commitments.

Revocation

Use `POST /api/v1/subjects/{id}/revoke` when assurance for a subject must stop, including suspected compromise or the beginning of recovery. Revocation marks the subject revoked, revokes its outstanding proofs and registered passkeys, cancels pending challenges, stops associated sessions, retires browser-token access, and discards the stored Brainprint template.

Revocation preserves relevant lifecycle and decision records according to their retention rules. Use deletion for erasure. Recovery creates a fresh enrollment and template under the subject's recovery policy. Check proof status online to observe revocation of a previously issued proof.

Deletion

Use `DELETE /api/v1/subjects/{id}` through an authorized server or administrative workflow. Deletion stops the subject's sessions, removes the stored template, deletes subject-scoped records including evidence, enrollments, authorizations, proofs, challenges, passkeys and research consent records, and removes linked research donations. It records a deletion event without retaining the deleted subject identifier in that event payload.

Keep a customer-side deletion request record and confirm successful completion. For an error response, inspect its code and retry according to the lifecycle procedure. Use the subject deletion operation for erasure and revocation for stopping reliance on the subject.

Backups, billing evidence, independent administrative audit records, customer systems, and webhook recipient copies follow separate retention and erasure procedures. Queued webhook deliveries and delivery-retry records continue through their delivery lifecycle, so an older event can arrive after subject deletion. Keep a minimal deletion marker at your receiver to discard stale events for that subject. Coordinate remaining service-side records through the agreed privacy process, including backup and legal-retention handling.

Export scope

`POST /api/v1/subjects/{id}/export` returns the subject representation, template-presence metadata, enrollment history, and selected authorization, session, passkey, and research-consent records. Template vectors and passkey credential material remain protected within the service and are excluded from the export.

Authorization, session, passkey, and research-consent lists include at most 500 records each; enrollment history includes at most 100 records. The endpoint returns this bounded view without pagination. Check these bounds against the scope of an individual request, especially for a long-lived subject.

For an individual request requiring additional records, contact founders@graypass.org with the requested scope. Agree a secure retrieval and transfer process for personal records before sharing an export.

Research choices

Research participation is optional and off by default. Set up the applicable research disclosure with GrayPass before offering it. Research retention requires deployment and environment enablement, a current granted receipt from the subject, and explicit opt-in on each donated evidence frame. Collect each person's consent separately from tenant configuration.

Collect and honor the person's choice as follows:

1. Present the applicable research purpose and disclosure separately from your core application terms and any camera permission.

2. After the authenticated person makes a choice, have your backend call `POST /api/v1/subjects/{id}/research-consent` with `{"consented": true}` or `{"consented": false}`. Use the subject resolved from your authenticated account mapping.
3. Keep your own record of the choice and disclosure shown. The endpoint returns the subject record; GrayPass stores a separate receipt with the fixed purpose `behavioral_model_research` and disclosure version `research-consent/1`. Show the disclosure matching that version. Receipt metadata is included in the bounded subject export.
4. Enable `researchConsent: true` on the browser observer only for a granted choice. It sends the required per-frame JSON field `research_optin: true`; opt-in is evaluated from this body field, independently of request headers. The ordinary observer default is `false`.
5. On withdrawal, record `{"consented": false}` and stop the opted-in observer. If ordinary observation continues, restart it with research consent disabled. Continuing to send opted-in frames without current environment permission and subject consent returns a validation error.

Receipts are append-only: a new grant or withdrawal changes the current choice while preserving its history. The session's `research_enabled` field summarizes environment and subject eligibility at session creation. Storage is evaluated separately at research-write time and requires deployment enablement, the current granted receipt, and per-frame opt-in. A withdrawal takes effect for subsequent writes, including those from existing sessions.

Withdrawal stops future research writes and preserves historical records. Subject deletion removes the linked consent receipts and donations through the procedure above. Research records are pseudonymous personal data; apply the appropriate disclosure, access controls, and retention handling.

Logs, support, and disclosures

Service logs use identifier fingerprinting and secret redaction where applicable; audit records can retain identifiers needed for investigation. Review customer-provided metadata separately before logging it. When escalating an issue, share request identifiers, timestamps, environment kind, reason codes, and a synthetic reproduction first.

Your user notice should explain the consequential actions covered, the kinds of derived evidence, the authentication used for enrollment, the recovery route, relevant retention, and how to make an individual request. Explain and collect optional camera and research choices separately. Use an approved privacy notice and data-processing agreement for your deployment and users.

SECURITY AND PRIVACY

Enrollment and recovery

Enroll a Brainprint after your application authenticates the account and associates it with a GrayPass subject. That account binding establishes the reference for subsequent behavioral comparison.

Define the enrollment point, required authentication, and recovery roles before collecting evidence. See [Subject lifecycle](#) for the API sequence.

Establish the customer-side identity first

Authenticate the application user using your existing controls. Resolve the GrayPass subject from the authenticated server-side account, rather than trusting an arbitrary subject identifier in a browser request. Issue the browser token for that subject and the exact application origin.

Choose an enrollment point with meaningful customer-side assurance: for example, after a recent authenticated account-recovery review or an appropriately verified account setup. Apply your account-assurance requirements to that enrollment event.

Enrollment anchors

An anchor records the source of assurance the customer says supported enrollment.

Anchor	What the customer declares	Integration responsibility
none	No identity assurance was supplied.	Enrollment is unanchored.
customer_session	An authenticated customer application session; the default.	Authenticate the account owner before enrollment.
idp	An identity-provider assertion, with an evidence reference.	Verify the assertion in your identity-provider integration.
passkey	A customer-declared passkey enrollment anchor, with an evidence reference.	This declaration is distinct from a GrayPass-verified action challenge ceremony.
verified_identity	A customer-run identity-verification step, with an evidence reference.	Verify the provider result and retain its supporting evidence.

The anchor is recorded on the subject and enrollment. Use its reference to correlate the event with identity evidence verified and retained in your system. `identity_anchor_validity` has status `unsupported` because this field records customer-declared provenance.

Collect enough evidence

Start an enrollment-purpose observer and let the person interact naturally with the supported page. Use `enrollment_ready` from evidence acknowledgements or `observer.status()` to determine when your backend can request completion. The server owns the readiness requirements and rechecks them when completing enrollment. Use its reported state as the completion signal.

Readiness reflects accepted windows, active interaction time, supported modalities, and evidence quality. Keep the page focused and show the returned counts, modalities, and rejection reasons as progress information. While `enrollment_ready` is false, continue natural interaction or address the reported collection issue. Complete enrollment when the server reports readiness.

Your backend completes enrollment after checking the authenticated subject and the intended enrollment session. Use browser status for progress display and backend authorization for completion.

Recovery policy

Policy	Recovery requirement	Appropriate consideration
re_enroll_with_anchor	Replacement anchor must meet at least the recorded original anchor strength.	Default; retain your own evidence for the replacement anchor.
re_enroll_any	Allows a weaker declared replacement anchor.	Deliberately weaker recovery; assess the account-takeover exposure.
manual_review	A permitted human review must approve recovery.	Provide a staffed process and review evidence outside the potentially compromised session.

Anchor strength orders the declared categories for recovery-policy checks. Select a category that matches the evidence verified by your application.

Recovery begins after revocation. Revocation removes the old template and invalidates outstanding subject proof/passkey paths. Beginning recovery returns the subject to a fresh enrollment state under the enforced recovery policy. Complete the new enrollment, then request authorization for subsequent actions.

A customer recovery procedure

1. Pause the sensitive action and authenticate the person through your approved recovery channel.
2. Review takeover indicators and any changed account details before trusting the existing browser session.
3. Revoke the subject when compromise or template replacement warrants it.
4. Obtain the required anchor evidence or human approval and begin recovery.
5. Collect and complete a new enrollment; register new passkeys where needed.
6. Request a new action authorization and proof with fresh binding and time limits.
7. Record the result in your own support and security workflow.

Support device and input-method changes with an accessible recovery route and accountable review. Use enrollment and recovery results for the application-security purpose described in [Supported use](#).

What to monitor

Track enrollment attempts, recovery frequency, anchor changes, and privileged reviewer activity. Compare GrayPass events and Console activity with your account-authentication signals. Apply your account-recovery policy and reconcile current state through the API when investigating an event.

SECURITY AND PRIVACY

Threat model

GrayPass evaluates evidence about the operator of an authenticated application session and binds its decision to a specific action. This threat model explains what each participant must protect, which protocol controls address an attack, and where independent customer controls are needed.

The scope includes the customer API, browser and Node SDKs, proofs, and your application's integration boundary. Use it when choosing protected actions, policies, verification rules, and recovery procedures.

Assets and adversaries

Protect the subject-to-account binding, stored Brainprint, session evidence, policy configuration, proof-signing authority, pending business operation, and resulting financial or administrative action. Decision traces, enrollment records, and administrative activity also require confidentiality and integrity.

Relevant adversaries include an account attacker, a person taking over an unattended session, a script controlling the page, a thief replaying a proof, an operator using the wrong environment, and an administrator weakening a policy. Select controls for each attacker capability, including the page and device assumptions in [Endpoint security](#).

Trust boundaries

Boundary	Responsibility	Integration control
Customer identity system	Authenticate the account and authorize enrollment.	Retain the account-to-subject mapping and enrollment evidence.
Browser and SDK	Collect derived observations under a scoped session.	Apply page and endpoint controls to the evidence producer.
Customer backend	Define the intended operation and enforce its decision.	Validate ownership and save the exact request before authorization.
GrayPass service	Evaluate policy, manage subject/proof state, and sign the result.	Interpret typed claims with coverage, freshness, and assessment context.
Relying-party verifier	Check proof binding and current state before execution.	Verify independently expected values and consume online.
Webhook channel	Deliver authenticated asynchronous events.	Deduplicate, handle reordering, and reconcile API records.

Threats addressed by protocol and implementation controls

Threat	Control	Integration requirement
Action substitution	Canonical action hash includes consequential request fields; verifier compares an independently stored request.	Include every consequential field and compare with the saved request.
Proof replay	Expiry, nonce binding, online revocation state, and single-use consumption.	Check current revocation and consumption through online verification.

Threat	Control	Integration requirement
Forged proof	ES256 signature validation against the expected issuer and key set.	Protect signing authority and configure independent verifier expectations.
Sandbox proof in live use	Separate signing keys and explicit tenant/environment checks.	Validate the expected tenant and environment for every proof.
Cross-tenant object access	Tenant and environment predicates; foreign-object requests receive non-disclosing failures.	Your own account-to-subject mapping must also be correct.
Evidence replay	Session-bound authentication, frame identifiers, timestamp/order checks, and overlapping-window rejection.	Protect the page that produces new authenticated evidence windows.
Malformed numerical evidence	Schema checks and validation of finite, in-range input.	Assess evidence provenance alongside numerical validity.
Browser policy tampering	Policies and authorization decisions live server-side under server credentials.	Restrict policy changes to approved administrators.
Repeated challenge loops	Bound challenge count and escalation to hold.	Your application must preserve the authorization chain and stop execution while unresolved.
Enrollment poisoning	Explicit anchor provenance, recovery policy, and customer-authenticated enrollment.	Establish the rightful account owner through your enrollment process.

Behavioral threats and evidence interpretation

For stolen-session use and operator handoff, examine continuity against the enrolled Brainprint and change indicators within the session. Review the claim status, evidence coverage, freshness, and quality together. Missing or insufficient evidence should follow the policy's explicit safe path rather than being treated as a match.

Automation indicators describe regularity or replay patterns in the observed evidence. Combine them with your endpoint controls and account authentication. Interpret freshness and coverage as evidence properties. Evaluate outcomes with representative users, input methods, handoff scenarios, and accessibility needs for the workflow you intend to protect.

Integration assumptions

- Maintain a trusted browser, endpoint, first-party script environment, and extension policy.
- Establish the rightful account owner before enrollment and recovery.
- Apply independent controls for physical coercion and malicious intent by an authenticated user.
- Execute protected operations only after the required decision and proof checks succeed.
- Protect privileged access and signing-key authority with operational access controls and audit.
- Include every consequential business field in the action binding.

Customer acceptance exercises

Test a changed destination or amount after authorization, a proof for another subject or audience, an expired or revoked proof, concurrent duplicate execution, stale evidence, a cross-environment credential, and an unavailable verification service. Each should reach the intended safe outcome without executing the protected mutation.

Test recovery with a weaker anchor, an unauthorized reviewer, and a previously revoked passkey. Test the crash between proof consumption and your business commit, including reconciliation across those separate transactions.

Use synthetic tests to exercise protocol handling, and consented participant sessions to measure challenge rates, holds, insufficient evidence, and false positives for your workflow. Set acceptance criteria before the evaluation and investigate differences across input methods and accessibility needs. See [evaluation](#).

Review and change

Revisit this model when changing protected actions, enrollment/recovery policy, browser dependencies, proof verification, hosting, or role assignment. Request controlled supporting material through founders@graypass.org; its availability and permitted recipients are agreed separately.

SECURITY AND PRIVACY

Endpoint security

Combine GrayPass action binding and proof verification with your application's browser, device, and transaction controls. This guide explains where each control applies and how to handle a suspected endpoint compromise.

What an attacker can control

The browser is the collection environment. Code with page privileges can access session signing material, construct feature windows, and change the visible interface. Device management, extension policy, application dependency review, and browser security controls address this layer.

GrayPass authenticates evidence to its session and checks replay, ordering, and numerical validity. The `device_integrity` claim has status `unsupported; limitation.compromised_endpoint_not_detectable` identifies the endpoint trust assumption in the Assurance Profile. Use the profile together with your device controls.

What action binding provides

Your backend saves an exact pending operation before requesting authorization. A proof binds the subject, action, resource, context, audience, actor where applicable, and nonce. At execution, compare its hash with the saved request and execute the stored values.

Include every consequential field, such as the destination fingerprint and amount. This lets the verifier detect substitution after the request is bound. To establish what the person reviewed before authorization, supply the saved intent through the transaction-confirmation path appropriate to your application.

What passkeys provide

An action-bound passkey challenge verifies the authenticator response and required user verification for the ceremony. Display the saved transaction details in your application before invoking the passkey prompt. For operations requiring a trusted transaction display, add an independent review or confirmation channel.

The sandbox's labeled confirmation control exercises the integration chain. Use a registered passkey for the authenticator ceremony described in [Decisions and challenges](#).

Design for independent controls

Keep account authentication, application permissions, device controls, and transaction-risk checks in your integration. Add a second reviewer, execution delay, limits, or an independent confirmation channel where the operation requires them.

Route `hold` decisions to an authorized reviewer with the saved request and an independent basis for approval. Set evidence freshness and proof lifetimes for the operation's risk and expected duration.

Incident response

1. Pause the pending operation and invalidate the affected application session.
2. Revoke the relevant GrayPass subject or credential.
3. Preserve the trace, timestamp, and required investigation records under your retention policy.
4. Complete account recovery through your trusted channel before re-enrollment.
5. Request a fresh authorization for the saved business intent when the account is ready.

Follow [Security](#) for credential handling and [Enrollment and recovery](#) for the recovery sequence.

SECURITY AND PRIVACY

Supported use and customer responsibilities

Use GrayPass for transparent, proportionate assurance at consequential application actions. Keep collection, policy, and review focused on the session and the bound request. The applicable agreement defines use restrictions and contractual remedies.

Surveillance and unrelated profiling

Behavioral evidence is collected for application security. Employee productivity scoring, attention monitoring, engagement rankings, disciplinary surveillance, sensitive-trait inference, and unrelated profiling are prohibited uses.

Keep identifiers within the authorized application context. Cross-site or cross-customer tracking, population identity matching, and re-identification outside that context are prohibited.

Covert collection

Explain the security purpose, evidence categories, and actions covered before collection. Provide an explicit choice and browser permission for camera collection. Offer research participation through its separate consent flow.

Covert collection and pressuring users into unrelated data collection are prohibited. Preserve the ordinary non-camera workflow when users decline camera access.

Inappropriate decision authority

Maintain your application's authentication, identity verification where applicable, and account permissions. Use GrayPass session and action evidence within those controls. Sole-factor authentication and identity-proofing uses are outside the supported integration model.

Employment, credit, insurance, housing, education, immigration, law-enforcement, and other rights or livelihood decisions require their own appropriate evidence and human review. GrayPass session assurance supplies no evidence of eligibility for those decisions. Automated adverse decisions based on it without appropriate human review are prohibited.

Report coverage, signed decisions, and evaluation results according to the property they measure. Attribute results to the tested population, workflow, and conditions.

Research and data handling

Obtain consent before behavioral research. Research retention requires deployment and environment enablement, a current subject consent receipt, and per-frame opt-in. Follow the [research choices procedure](#) for grant, withdrawal, and deletion.

Use opaque identifiers and narrowly selected action context. Keep passwords, payment-card data, identity documents, and unrelated sensitive records out of metadata and context. Cross-context template reconstruction and matching are prohibited.

Product safeguards

GrayPass scopes subjects to tenants and environments, returns template-presence metadata in exports, separates research consent from ordinary processing, and scopes browser tokens to evidence and ceremony operations. Your access controls, disclosures, and review procedures complete the integration's governance.

Reporting concerns

Contact founders@graypass.org for guidance on a proposed use, misuse reports, or a private vulnerability report. Start with a description and synthetic example; use the agreed secure process for any required personal records.

ACCOUNT AND BILLING

Use the Billing console

Open **Console**, select the correct tenant, and choose **Billing** to review your commercial stage, usage, agreements, installment schedules, invoices, enterprise additions, and payment access. You can use the page to reconcile agreed terms and see what needs attention before paying or renewing.

The commercial stage describes your tenant's agreement and access. It is separate from the API environment selected elsewhere in the console. A tenant can still be commercially in sandbox when using the hosted console; confirm the intended agreement and environment before sending live traffic.

Understand permissions

All signed-in tenant roles can view Billing. Owners and administrators can save supported agreement drafts, open the payment portal, and record non-renewal. Developer, security reviewer, and viewer roles have read-only billing access.

Task	Your team	GrayPass billing team
Agree commercial terms	Confirm payer, scope, dates, usage needs, and signed terms.	Prepare and approve the commercial arrangement.
Save a supported draft	An owner or administrator can record the agreed details when using this workflow with GrayPass.	Review the draft against the signed terms.
Activate service and release invoices	Satisfy the agreed prerequisites and review the schedule.	Approve the agreement and release eligible invoices.
Pay an invoice	Use its payment link or agreed remittance instructions.	Reconcile receipts and any payment exceptions.
Correct tax, charges, or balances	Supply the reference and requested correction.	Review and apply the appropriate financial adjustment.
Record non-renewal	An owner or administrator records the decision for the correct term.	Handle any contractual notice or settlement questions.

GrayPass's billing team handles financial approval, tax setup, invoice release, refunds, credit notes, accounting exports, and commercial exceptions. Send the relevant agreement or invoice reference when requesting one of these changes.

Read current usage

The page shows the selected meter's used quantity, monthly included amount, estimated additional charge, and contractual UTC window when an active production window exists. Use the active window for current estimates and the invoice list for historical amounts.

Compare the displayed meter and window with the agreement. Assured Actions and Active Brainprint months count different units; the Settings diagnostic counters are not substitutes. See [usage and billing](#).

Review an agreement

An agreement section identifies its kind, annual amount, collection cadence, service dates, meter, allowance, and additional-usage rate where applicable. It also shows relevant evaluation-credit deadlines, first-term concessions, renewal amount, and renewal status.

The **Platform installment schedule** separates gross amount, evaluation credit, net amount, invoice date, and status. The **Invoices** table separates invoice total, recorded cash and adjustments, financial state, due date, and payment link. Enterprise additions have their own service dates and finite schedules.

Held identifies an invoice awaiting financial release. **Pay** opens the hosted payment page. Completed settlement appears in the invoice's payment state.

Prepare an agreement draft

Agree the commercial arrangement with GrayPass before filling in a draft. Use the form to record the agreed information and submit it for review. If GrayPass is preparing the agreement for you, review the resulting record instead of creating a second draft.

When an owner or administrator uses the draft workflow:

1. Confirm the tenant, legal payer, billing contact, application scope, and service dates.
2. Use the signed-document reference and integrity information supplied for the agreement. Enter the supplied values and the records belonging to this agreement.
3. For an evaluation, record the agreed fee, scope, and success criteria. For production, verify the annual commitment, collection cadence, selected meter, monthly allowance, additional-usage rate, and signature date.
4. Identify any eligible source evaluation or first-term concession agreed with GrayPass. Check how the credit changes the installment schedule.
5. Save the draft and review its recorded details with GrayPass. Agreement approval, invoice release, and commercial activation are separate steps.

Changes to tax setup, payment-term exceptions such as Net 30, free production-shadow terms, and other commercial exceptions require GrayPass's billing team. Coordinate the agreed arrangement through founders@graypass.org before submitting it.

Manage payments

Use **Open payment portal** to manage payment methods and invoices for your tenant. Handle plan changes, cancellation, and contract modifications through the signed-agreement process.

If the portal is unavailable, confirm with GrayPass that payment access has been set up for this tenant. Follow the methods shown on the hosted invoice. Preserve remittance references and inspect updated state after pending bank payments, refunds, disputes, or credit notes. See [invoices and payments](#).

Non-renewal and service end

For an eligible production agreement, an owner or administrator can choose **Record non-renewal**. Confirm the term and keep the recorded result. The recorded decision applies to the future renewal; the current term and its obligations continue under the agreement.

If service has ended, the page can distinguish remaining scheduled fees retained under settlement, waived under settlement, or awaiting a settlement decision. Existing invoices require separate resolution. Review [renewal and lifecycle](#) before interpreting these states.

Resolve a discrepancy

Check the correct tenant, agreement, UTC usage window, and role first. Compare the schedule and invoice lines with the signed terms. Record the invoice or agreement reference and discrepancy, then request reconciliation against the existing agreement and payment records.

Contact founders@graypass.org with the minimal references needed. Do not include API secrets, payment-card details, full subject exports, or raw behavioral data in a billing request.

ACCOUNT AND BILLING

Commercial model

GrayPass is one platform for action-specific assurance. Commercial terms cover an agreed application scope and service period. Contact founders@graypass.org for evaluation and production terms.

The signed agreement determines fees, allowances, service dates, payment terms, renewal, and any service additions. This chapter explains the platform commitment, usage allowances, evaluation credit, and collection schedule so your team can plan and reconcile the agreement.

From sandbox to production

Stage	Purpose	Commercial boundary
Sandbox	Integrate the API, exercise failure paths, and understand the console.	Sandbox activity does not generate production usage overages.
Paid evaluation	Answer an agreed application-specific question against defined success criteria.	A signed agreement defines the fixed evaluation fee and service period separately from technical evaluation runs.
Production agreement	Cover a defined service term, included usage, and additional usage.	An annual platform commitment, selected meter, and collection schedule govern the term.
Enterprise addition	Cover an additional service or capability in the agreement.	Each addition has an agreed amount, scope, and service period.

Define the intended workflow, protected actions, evaluation criteria, and integration scope before selecting terms. The [capabilities](#) and [evaluation](#) guides help your technical and procurement teams agree that scope.

Annual platform fee and collection

A production agreement has an annual platform fee. The supported collection schedules are annual upfront, four quarterly installments, or twelve monthly installments. Quarterly and monthly collection distribute the annual commitment across the agreed installment schedule.

When an amount cannot divide evenly into cents, the final installment carries the remainder. The schedule shows gross platform fees, any eligible evaluation credit, net installments, and invoice dates. Taxes, usage, and additions are represented separately where applicable.

Prepaid collection is the standard access prerequisite. An approved Net 30 arrangement uses the prerequisite and due dates recorded in the agreement.

Included and additional usage

The agreement selects one meter: **Active Brainprint months** or **Assured Actions**. It specifies the units included in each contractual monthly window and the exact additional-usage rate. The included allowance applies throughout the annual term, one window at a time. Unused units expire at the monthly window's end.

The billing window follows the original UTC service-start anniversary. Covered applications share one allowance pool for that window, regardless of application labels.

Commercial feature	Treatment
Annual platform commitment	Collect upfront, quarterly, or monthly under the agreed annual obligation.
Included usage	A monthly allowance for the selected meter, shared by the covered application scope.
Additional usage	Rate eligible units above that window's allowance in USD.
Approved allowance, meter, or rate changes	Take effect at the next contractual window. The current window keeps its agreed terms.
Enterprise additions	Explicit service dates and a finite agreed amount, including an addition covering part of a term.

Standard agreements rate usage in USD against one monthly pool for the covered application scope. Annual collection uses the annual payment schedule while preserving monthly allowance windows. A partial-term addition uses its explicitly agreed amount and service period. Confirm these terms when planning additional applications or services.

[Usage and billing](#) defines both meters, deduplication, exclusions, and rounding. Production shadow-mode assurance counts according to the agreement's usage and shadow-period terms.

Evaluation conversion credit

An eligible paid evaluation can produce a credit toward the first production term when the production agreement is signed within the agreed conversion window. The supported standard deadline is the end of the UTC date 30 days after the evaluation end date, inclusive. The actual signature timestamp is used to assess eligibility.

Eligible evaluation fees collected and available after refunds or adjustments provide credit, excluding taxes. Apply that credit to the earliest first-term platform installments until exhausted. Additional usage, taxes, enterprise additions, and renewal fees retain their own charges. Credit offsets eligible invoices; cash refunds follow their separate agreed terms.

Review the credit amount, source evaluation, signature date, and net installment schedule with GrayPass before activation. Request any evaluation extension or correction through the agreed contracting process so the recorded schedule matches the signed terms.

Enterprise additions

An addition has signed terms defining the deliverable and a service period within the production agreement. It can be a one-time charge or a term total collected on the agreement's installment cadence. Recurring additions align to complete installment periods within that term. Enterprise additions retain their agreed charge, independently of evaluation credit.

Additions apply to their signed term. Include any continuing addition explicitly in the renewal agreement. Confirm the deliverable, service dates, dependencies, and renewal treatment in the agreement.

Changes and renewal

Agreement changes require the recorded approval process. Use the payment portal for payment methods and invoices, and the signed-agreement process for plan changes and cancellation. Renewal amount and any first-term concession must be explicit. Non-renewal follows the signed agreement and is recorded through the console or agreed contact process.

Service access, annual obligations, invoices, and payment adjustments each follow their recorded lifecycle. See [renewal and lifecycle](#) for the separation between access, remaining commitments, and receivables.

Procurement and activation

Before activation, align the intended use, applicable legal/data-processing documents, support commitments, tax treatment, payment method, and protected actions. GrayPass confirms activation after the agreement and its prerequisites are satisfied. Customer-specific pricing, commercial annexes, controlled security evidence, and executed documents are supplied through an agreed private review process.

Your team reviews the agreement, maintains accurate payer and contact details, pays released invoices, and records non-renewal when appropriate. GrayPass handles approval of commercial terms, invoice release, tax setup, and financial corrections. The [billing console guide](#) explains which actions your account can perform.

ACCOUNT AND BILLING

Usage and billing

GrayPass bills according to the meter and allowance in the production agreement. Each meter counts qualifying logical units. Use those units and their contractual window to reconcile an invoice.

Your agreement selects one meter for its shared usage pool. Usage follows contractual monthly windows, exact decimal rates, and deduplication of the same logical unit. Read [commercial model](#) for the annual platform fee and [invoices](#) for collection.

Assured Actions

An Assured Action is a persisted, qualifying production decision in a live environment. The meter covers enforce decisions when permitted for the environment. Qualifying live shadow decisions count under a production agreement unless an explicit free production-shadow term applies. See [action modes](#) for the distinction between observation and protected execution.

The final decision can be allow, challenge, hold, or deny. A decision resulting from insufficient evidence, identified by `decision.insufficient_evidence`, is an abstention and does not create a billable unit.

The unit follows the logical action. Idempotent retries, a challenge and its resulting child authorization, and proof retrieval or verification share that logical action's single unit.

A new logical operation with a different identity can be a separate unit. Preserve stable idempotency for one business intent and reject payload changes under the same identity in your application.

Active Brainprint months

An Active Brainprint month is one subject for which GrayPass produced a qualifying assurance profile during the contractual monthly window. The profile must have sufficient evidence. The first qualifying assurance records or references the durable evidence of that qualification; the same subject contributes at most one unit to the selected pool in that window.

The same subject can contribute one unit in the next window after producing a new qualifying assurance there. Creating a subject, completing enrollment, or submitting raw or accepted evidence frames does not alone create this unit. An insufficient-evidence assessment also does not count.

A session status or assessment read can produce and record qualifying assurance. Both `GET /api/v1/sessions/{id}/status` and `GET /api/v1/sessions/{id}/assessment` can therefore create Active Brainprint usage under the agreement's shadow-use terms. The status read preserves enrollment state and reports assurance; request action authorization separately. Repeated qualifying reads for the same subject in one contractual window share one subject/window unit. Use health endpoints for monitoring and request status or assessments for their intended integration purpose.

Inclusion and exclusion table

Usage eligibility follows the environment, effective action mode, and agreement. Use live shadow mode for the production evaluation workflow; sandbox enforcement exercises the complete proof flow. The table also defines enforce-meter semantics for environments permitted to use that mode.

Activity	Assured Actions meter	Active Brainprint months meter
Subject creation or enrollment	No unit.	No unit from enrollment alone.
Browser evidence submission	No unit from the frame itself.	No unit from the frame itself.

Activity	Assured Actions meter	Active Brainprint months meter
Sandbox development activity and evaluation agreement activity	Excluded from production overages.	Excluded from production overages.
Qualifying live shadow decision	Counts unless covered by an explicit free production-shadow term.	A qualifying assurance can activate the subject unless excluded by that recorded shadow term.
Qualifying enforce decision, when permitted for the environment	Counts as one logical action.	A qualifying assurance can activate the subject for the window.
Insufficient-evidence result	Excluded.	Excluded.
Retry of an existing logical action	No duplicate unit.	No duplicate subject/window unit.
Challenge continuation and resulting proof	Part of the existing logical action.	No duplicate subject/window unit.

Record a free production-shadow period explicitly in the agreement, including its ending boundary.

Live credentials produce live-environment traffic, including requests from a development machine. Select the intended environment and agreement scope before running tests against a service.

Contractual monthly windows

Windows are UTC, start-inclusive and end-exclusive. They are calculated from the original service-start anchor. For a January 31 start, the next boundary is the final day of February and the following boundary returns to March 31. The original anchor remains in effect after February's shorter month.

For example, an agreement beginning at 12:00 UTC on January 31 has a first window ending at 12:00 UTC on the final day of February. An eligible event one instant before the ending boundary belongs to the first window; an event exactly at the boundary belongs to the next. A timezone-offset date is normalized to the same UTC instant before assigning its window.

Agree a production service start on a whole UTC minute. That exact start determines the contractual usage boundaries. Signature timestamps retain their actual precision for conversion-credit checks.

Use Billing for the contractual usage ledger and Settings for calendar-month diagnostics. Their different boundaries can produce different totals. Replays retain their original logical event time, agreement binding, and usage identity.

Allowances and arithmetic

For a window, additional units are the eligible units above the included allowance, with a minimum of zero. The charge is additional units multiplied by the contracted decimal unit rate, rounded once at the invoice line to cents using half-up rounding. Rates support up to six decimal places. Event amounts retain precision until that invoice-line rounding step.

Use this symbolic check against an invoice:

```
additional_units = max(eligible_units - included_units, 0)
usage_line = round_half_up(additional_units × contracted_rate, 2 decimals)
```

Use the rate and allowance recorded for that agreement and window. Each window retains its applicable terms when a later change is approved. The included monthly allowance resets at the next window, with unused units expiring at the current window's end.

Estimates, close, and corrections

The Billing view shows current usage, the included allowance, the contractual window, and an estimated additional charge. Current estimates can change as eligible work is recorded and reconciled. Final invoicing uses the closed window and its agreed economics.

GrayPass reconciles the agreement, usage identities, window, and totals before releasing a usage invoice. A discrepancy can leave the invoice held for review. Replays retain the same logical usage unit. Corrections appear as adjustments so you can reconcile the original charge and its resolution.

If a total looks wrong, record the tenant, environment, agreement reference, UTC window, affected authorization or subject references, and your expected result. Contact founders@graypass.org. Send references and a minimal description first, rather than personal data or complete request payloads.

ACCOUNT AND BILLING

Invoices and payments

GrayPass records the agreed commercial obligations and usage. Stripe supports invoice delivery and payment collection. The invoice and payment state shown in Billing should be read together with the agreement and any credits or adjustments.

Use the invoice and settlement breakdown to reconcile cash, credit notes, customer-balance allocations, and bank payments. The Console presents these amounts separately.

What appears on an invoice

Depending on the agreement and billing event, an invoice can cover an evaluation fee, a platform installment, additional usage, or an approved enterprise addition. The schedule identifies gross platform fees, evaluation credit, and net installment amounts. Applicable taxes and financial adjustments are separate from the definition of usage units.

Review these fields before paying:

- The legal payer and billing contact.
- Agreement and service-period references.
- Invoice kind, currency, issue date, and due date.
- Gross fee, applied evaluation credit or other adjustments, and net amount due.
- For usage, the window, eligible/included/additional quantities, and contracted rate.
- Applicable tax treatment and the supported payment instructions.

For an invoice correction, contact GrayPass with the existing invoice reference and the line to review. Retain that reference throughout reconciliation.

Held, released, and paid are separate states

State or indication	Meaning	Customer action
Draft or held	An obligation or a later collection issue is awaiting review.	Review details if requested; confirm payment instructions with GrayPass if the invoice is held.
Released/open	The invoice is available under the approved collection process.	Use its hosted payment link or agreed instructions.
Paid/collected	The system has recorded a settlement outcome.	Review the payment and adjustment breakdown if reconciling cash.
Credited, void, refunded, or disputed	A financial adjustment or exception affects the invoice.	Reconcile the adjustment document, current invoice state, and service access separately.
Reconciliation or collections review	Evidence or a payment event needs review.	Contact the billing team with the invoice reference.

GrayPass releases an invoice after the agreement and applicable billing prerequisites have been checked. The console may show **Held** instead of a payment link. A later financial review can also affect an invoice that previously had a payment link, so use its current status and the billing team's instructions. If a hold affects your planned start date, contact GrayPass using the existing agreement reference.

Payment methods

Use the invoice's hosted page to see the payment methods available for your account and invoice. Supported payment flows include cards, bank transfer, and ACH debit. Bank methods require the relevant account capability, customer eligibility, and commercial setup. Confirm an essential payment method with GrayPass before your due date; only use the instructions presented for your invoice.

Bank transfer and ACH debit have different settlement and reversal semantics. Track a transfer or debit through final settlement. Retain the reference for a pending bank event and request reconciliation before initiating another payment.

Wallet options and authentication prompts depend on your device, payment method, and account configuration. Follow the hosted page's instructions if authentication is requested. Check the tax lines on the actual invoice and raise an incorrect payer address or tax treatment with GrayPass before payment.

Hosted payment portal

Owners and administrators can open the payment portal from [Console → Billing](#) when a supported customer payment account is available. The portal provides the supported payment-method and invoice functions. Use the signed-agreement process for plan changes, renewal terms, cancellation, and enterprise additions.

Use the link created by the console for your current tenant. Do not share a portal session link in a ticket or reuse another person's link. If a portal action fails, record the time and invoice reference and contact founders@graypass.org.

Receipts, balances, and external payments

The invoice view distinguishes collected cash, refunds, credit notes, and, where applicable, verified externally received cash. Reconcile credits as invoice-ledger adjustments and cash as received funds.

For an agreed payment made outside the hosted invoice flow, provide GrayPass with the invoice and remittance references through the agreed billing channel. GrayPass verifies funds received and reconciles the invoice. A receipt records the funds already received; initiate transfers through the agreed payment method. Partial or excess external amounts require review rather than assuming a full settlement; retain your payment record until the invoice reflects the agreed resolution.

Keep remittance details and invoice references together in your own accounting system. Reconcile the invoice's current state after a refund, dispute, reversal, or credit note rather than relying on an old screenshot or notification.

Payment failure and access

A failed or reversed payment may change commercial access according to the agreement and review process. It must not change an assurance decision from deny, hold, or challenge into allow. If commercial access is unavailable, an integration receives an explicit commercial-access error and must follow its safe operational path.

After commercial access is restored, request fresh authorization for the pending operation and follow its decision and proof requirements.

Notifications and disputes

Use the console to inspect the current agreement and invoice state. Keep your billing contact current and confirm the agreed invoice-delivery method during setup. If an expected email is missing, check Billing and contact GrayPass; the signed due date and invoice remain in effect independently of reminders.

For a billing question, share the invoice reference, the discrepancy, UTC dates, and relevant remittance reference through the agreed support route. Never send full payment-card details, API keys, or unrelated customer data. Contractual dispute and payment deadlines remain those of the signed agreement.

ACCOUNT AND BILLING

Renewal and commercial lifecycle

Commercial access has a lifecycle separate from enrollment, API environment, and financial balances. A sandbox can be available while no production agreement is active. An ended service term can still have outstanding invoices or remaining obligations.

Use the signed agreement and [Console → Billing](#) together to confirm service dates, renewal decisions, and settlement amounts.

Agreement progression

Event	Result	Next step
Save a draft	Commercial details are ready for review.	Confirm the record against the signed terms.
GrayPass approves signed terms	The agreement and schedule are recorded.	Complete payment, legal, tax, and technical prerequisites.
Meet access prerequisites	The agreed commercial stage begins.	Run application operations through their authorization flow.
Reach a usage boundary	The contractual window closes.	Reconcile usage and review the released invoice.
Record non-renewal	The future renewal decision is recorded.	Complete the current term and its obligations.
End service	Access ends under the recorded lifecycle action.	Resolve invoices, settlement, and offboarding procedures.

Prepaid terms ordinarily require the applicable payment evidence. An expressly approved Net 30 arrangement uses its recorded access prerequisite and due dates. Neither a failed payment nor a commercial-state transition turns a non-allow assurance result into permission to execute.

Evaluation end and conversion

A paid evaluation has an agreed period and success criteria. Track the commercial term and technical success criteria separately; renewals and extensions require an agreed term.

When converting, review the annual amount, selected meter, monthly allowance, rate, collection cadence, service start, signature timestamp, and eligible credit. The standard conversion-credit deadline is inclusive through the end of the UTC date 30 days after the evaluation end date. Record an agreed extension to change the deadline.

Credit reduces eligible first-term platform installments only. Renewal, additional usage, tax, and enterprise additions retain their separate charges. See [commercial model](#).

Renewal

Auto-renewal applies only when present in the signed agreement. Standard terms use a 30-day non-renewal notice period before term end. The agreement records the renewal platform fee separately when a first-term concession changes the first term's economics.

Renewal review should cover:

1. The successor service dates and collection cadence.
2. The renewal annual platform fee and any expiring first-term concession.
3. The meter, included units, additional-usage rate, and shadow-use treatment.
4. Any tax, payer, contact, or payment changes.
5. Additions that expire and require new signed approval.
6. The delivery and support commitments for the next term.

Review the renewal date and notice deadline when the agreement starts, and keep your billing contact current. Follow the signed notice requirements and confirm the future term through the approval process. Contact GrayPass before the deadline to clarify the renewal amount, service scope, or notice status.

Record non-renewal

An owner or administrator can use **Record non-renewal** for an eligible production agreement in the Billing view. Confirm the correct tenant and term before using it, and retain the recorded outcome. For contractual notices or an issue using the console, contact founders@graypass.org through the agreed process before the applicable deadline.

Non-renewal stops the future renewal process for that term. The current service period, annual commitment, and open invoices continue under the agreement. Submit the notice through the stated process.

Additions expire with their approved term

An enterprise addition has its own signed service period and finite schedule. The addition applies to that signed term. A renewed capability requires a newly approved addition with confirmed service dates and deliverables.

Check operational dependencies before allowing an addition to expire. Confirm continued delivery and service dates through the agreement for that addition.

Ending access and settling obligations

At an early end of access, resolve remaining annual platform obligations and scheduled addition fees through the agreed settlement process. Record the treatment of unissued fees.

If retained, remaining obligations follow the applicable schedule while service access stays ended. If waived, the waiver concerns the specified unissued fees. Already issued invoices, receivables, refunds, and credit notes require their own documented resolution; they retain their own financial lifecycle after access ends.

Residual invoice-ledger balances also require explicit handling. A credit or deferred debit on the ledger is distinct from cash and from a bank balance. Reconcile missing balance evidence before determining the balance.

Operational handoff at term end

Plan for the application behavior when commercial access ends. Stop new protected operations or route them through an independently approved safe process. Keep execution gated by the approved process. Preserve the evidence and exports you legitimately need before their retention windows expire.

Handle subject revocation or deletion through the [subject lifecycle](#) and [privacy](#) procedures. Complete any data-erasure request through the subject lifecycle and privacy procedures. Review stored credentials, webhook destinations, and access to customer-owned copies of records as part of offboarding.

PART 03

API reference

Find endpoints, authentication, request and response fields, and reason codes.

API reference

API REFERENCE

API reference

The canonical JSON API uses HTTPS on <https://app.graypass.org>. Requests and responses use JSON unless an operation returns no content. The API version is an explicit contract identifier, not a deployment date.

Begin with [authentication](#) and [errors and retries](#). Each operation below links to its credential scope, parameters, request body and response types. The [schema catalog](#) defines fields and constraints.

Operations

Method	Path	Operation
GET	/.well-known/graypass-proof-keys.json	Read public proof verification keys
GET	/api/v1/actions	List protected actions
POST	/api/v1/actions	Register a protected action
DELETE	/api/v1/actions/{action_id}	Archive a protected action
GET	/api/v1/actions/{action_id}	Read a protected action
PATCH	/api/v1/actions/{action_id}	Update action metadata or mode
GET	/api/v1/actions/{action_id}/policy-versions	List action policy versions
POST	/api/v1/actions/{action_id}/policy-versions	Create a draft policy version
POST	/api/v1/actions/{action_id}/policy-versions/{version}/activate	Activate a reviewed policy version
GET	/api/v1/activity	List authorization activity
GET	/api/v1/authorizations/{authorization_id}	Read an authorization trace
POST	/api/v1/authorizations/{authorization_id}/outcome	Record the business outcome
POST	/api/v1/authorizations/{authorization_id}/review	Resolve a held authorization
POST	/api/v1/authorize	Evaluate a bound business request
GET	/api/v1/challenges/{challenge_id}	Read a scoped challenge
POST	/api/v1/challenges/{challenge_id}/cancel	Cancel a pending challenge
POST	/api/v1/challenges/{challenge_id}/complete	Complete explicit action confirmation
POST	/api/v1/challenges/{challenge_id}/passkey/options	Get action-bound passkey options
POST	/api/v1/client-tokens	Mint a scoped browser token
GET	/api/v1/credential	Read calling credential identity

Method	Path	Operation
GET	/api/v1/evaluations	List shadow evaluations
POST	/api/v1/evaluations	Create a shadow evaluation
POST	/api/v1/evaluations/{evaluation_id}/close	Close a shadow evaluation
GET	/api/v1/evaluations/{evaluation_id}/report	Read evaluation metrics
GET	/api/v1/meta	Read API capabilities and reason vocabulary
GET	/api/v1/proofs/{jti}/status	Read current proof state
POST	/api/v1/sessions	Open an evidence session
GET	/api/v1/sessions/{session_id}/assessment	Read a server Assurance Profile
POST	/api/v1/sessions/{session_id}/end	End an evidence session
POST	/api/v1/sessions/{session_id}/evidence	Submit an SDK evidence window
GET	/api/v1/sessions/{session_id}/status	Read browser-safe session status
GET	/api/v1/subjects	List subjects
POST	/api/v1/subjects	Create a pseudonymous subject
DELETE	/api/v1/subjects/{subject_id}	Delete operational subject data
GET	/api/v1/subjects/{subject_id}	Read a subject
POST	/api/v1/subjects/{subject_id}/adaptation/freeze	Freeze template adaptation
POST	/api/v1/subjects/{subject_id}/enrollments/complete	Complete ready enrollment
POST	/api/v1/subjects/{subject_id}/export	Export bounded subject records
POST	/api/v1/subjects/{subject_id}/passkeys/register/complete	Complete passkey registration
POST	/api/v1/subjects/{subject_id}/passkeys/register/options	Begin passkey registration
POST	/api/v1/subjects/{subject_id}/passkeys/registration-grants	Issue a passkey registration grant
DELETE	/api/v1/subjects/{subject_id}/passkeys/{passkey_id}	Revoke a registered passkey
POST	/api/v1/subjects/{subject_id}/recovery	Begin subject recovery
POST	/api/v1/subjects/{subject_id}/research-consent	Record or withdraw research consent
POST	/api/v1/subjects/{subject_id}/revoke	Revoke subject assurance

Method	Path	Operation
POST	/api/v1/verify	Verify and optionally consume a proof

Machine-readable contract

[Download OpenAPI JSON](#) for client generation and request and response schemas, including field names, required values, defaults and constraints. Compare its API version with `GET /api/v1/meta` when integrating. Configure generated clients to handle the [validation response convention](#).

Shared response and example conventions

Authentication, scope, rate-limit and dependency failures apply across operations; see [errors and safe retry rules](#). Operation-specific failure notes supplement that common contract. HTTP success is not permission to execute a business action. Examples use illustrative identifiers and fixed times; replace them with values from your authenticated workflow. Response excerpts show selected fields; linked schemas define the full response. Use credentials, WebAuthn assertions, and proofs obtained through your own integration.

API REFERENCE

Authentication and API conventions

The canonical API base is `https://app.graypass.org/api/v1`. The API origin `https://api.graypass.org` also supports the canonical JSON routes; documentation and application UI live on the app origin. Use one reviewed endpoint consistently in your configuration. The API version reported by metadata is `2026-10-01`.

Credential boundaries

Authenticated server requests carry `Authorization: Bearer` with a secret credential stored on the backend. Browser requests use the short-lived client token minted by that backend for the signed-in subject and a registered origin. Do not put credentials in query strings or expose a server credential in a browser bundle.

Credential or scope	Intended use
Browser client token	Its scoped evidence session, safe status, and permitted challenge ceremony
client scope	Restricted key scope; not sufficient for the canonical server operations in this manual, including client-token minting
server scope	Runtime subject, session, authorization, proof, and outcome operations
admin scope	Control-plane action/policy and evaluation operations in addition to permitted server work

The operation reference identifies which endpoints require server or admin authority. Credentials access resources within their own environment. Configure Console roles for people and API scopes for service credentials separately. See [Console](#).

Check identity and origin

Call the credential-identity endpoint before setup to confirm tenant, environment, environment kind, scope, key mode, and proof issuer. The proof verifier still requires a trusted expected issuer; configure it for the service rather than copying an issuer from an unverified proof.

Browser origins include scheme, hostname, and port when used. Register the exact intended origin before minting a token. A token is short-lived and subject-bound; refresh it through your own authenticated backend. Apply application authentication and CSRF protection alongside origin registration.

Request conventions

JSON request models reject undeclared fields where specified by the contract. Use the exact `snake_case` wire fields shown in the generated reference; the Node SDK translates its documented `camelCase` inputs. Omit optional values rather than sending an unintended type. Canonical action material accepts safe integers, not floating-point values.

List responses and subject export are bounded. Use each endpoint's documented pagination parameters and collection bounds. Date strings explicitly requiring timezone information must include it; `authorization requested_at` uses Unix seconds.

Most resource timestamps are ISO 8601 strings. The token-issuance response instead uses Unix seconds for `expires_at`; evidence frames use the SDK's documented clock convention. Read each field's unit rather than applying one date parser to every numeric timestamp.

Retry and idempotency

Authorization binds an idempotency key and nonce to one request. Reuse them only when retrying identical material. Reconcile a conflict or stale result before obtaining fresh authorization. Follow each operation's documented retry and idempotency contract.

The authorization request timestamp must be no older than 300 seconds and no more than 60 seconds in the future. It is excluded from the action hash and idempotency comparison, so a retry may refresh that timestamp while preserving the original nonce, key and business material. A lifecycle or policy change can still make the stored authorization stale.

Use bounded backoff for rate limiting and transient transport errors. Keep the operation pending or refuse it while the response is uncertain. A timeout after online proof consumption needs application-state reconciliation before any further execution.

Public discovery

Metadata and the proof JWKS are public discovery surfaces. Metadata identifies the contract and capability states. JWKS publishes public verification keys while keeping customer records and private signing material protected. Fetch it from the configured trusted origin. The remaining operations below document credential inspection and browser token issuance in detail.

GET /.well-known/graypass-proof-keys.json

Fetch the public keys used for Proof of Agency signature verification from the configured trusted service.

Credential: Public; no credential required.

Inputs: No request body or credential. Use the SDK verifier for algorithm, issuer and binding checks.

Effect and retry behavior: Read-only discovery. Refresh keys through your verifier's bounded cache, then check current proof state through the proof API.

Responses

HTTP status	Meaning	Body
200	JSON Web Key Set with public verification keys. Verification requires the key matching the proof's kid.	application/json: object

Response handling: Resolve kid against the trusted key set before verification. Keep the operation pending while the required key is being resolved.

Example

HTTP

```
GET /.well-known/graypass-proof-keys.json
```

Response - HTTP 200:

```
{
  "keys": []
}
```

POST /api/v1/client-tokens

Exchange backend authority for a short-lived browser token bound to one subject and exact registered origin.

Credential: Server credential.

Inputs: Resolve subject from your authenticated session. ttl_seconds defaults 900 and permits 60-3600.

Effect and retry behavior: Each accepted request issues a fresh token, including retries. Store and use the returned token for its subject, origin and lifetime.

Request body

application/json; required. [ClientTokenCreate](#).

Responses

HTTP status	Meaning	Body
201	New browser token, exact origin, lifetime and Unix-seconds expiry.	application/json: ClientTokenCreated
422	The canonical handler returns a Problem for invalid input; correct the request before retrying.	application/problem+json: Problem

Response handling: Confirm that the subject exists and the exact origin is valid and registered, then mint a browser token through your backend.

Example

HTTP

```
POST /api/v1/client-tokens
```

JSON request

```
{
  "subject": "sub_example",
  "origin": "https://merchant.example",
  "ttl_seconds": 900
}
```

Response excerpt (selected fields) - HTTP 201:

```
{
  "expires_at": 1790856900,
  "ttl_seconds": 900,
  "origin": "https://merchant.example"
}
```

GET /api/v1/credential

Resolve the calling credential's tenant, environment, authority and proof issuer for trusted backend configuration.

Credential: Server credential.

Inputs: Use the server credential you intend to use for the integration.

Effect and retry behavior: Read-only; returns credential metadata, never the secret credential value.

Responses

HTTP status	Meaning	Body
200	Credential id, tenant, environment, key mode, authority and configured proof issuer.	application/json: CredentialIdentity

Response handling: Use a server credential for this operation. For an environment mismatch, select the credential belonging to the intended environment.

Example

HTTP

```
GET /api/v1/credential
```

Response - HTTP 200:

```
{
  "credential": "key_example",
  "tenant": "tenant_example",
  "environment": "env_example",
  "environment_kind": "sandbox",
  "scope": "server",
  "mode": "test",
  "proof_issuer": "https://app.graypass.org"
}
```

GET /api/v1/meta

Discover the API version, capability availability and reason-code vocabulary applicable to this service.

Credential: Public; no credential required.

Inputs: No request body or credential. Use this response to inspect the service contract and reported capabilities.

Effect and retry behavior: Read-only discovery of the current contract. Configure action mode, policies and credentials through their dedicated operations.

Responses

HTTP status	Meaning	Body
200	API version, capability declarations and reason-code registry.	application/json: object

Response handling: Enable a feature after confirming compatible metadata and the required configuration.

Example

HTTP

```
GET /api/v1/meta
```

Response excerpt (selected fields) - HTTP 200:

```
{
  "api_version": "2026-10-01"
}
```

API REFERENCE

Errors and response handling

Canonical API errors are represented as problem documents. Use the HTTP status and stable code for program logic. `title` and `detail` are human-readable explanations and can change without changing the error's meaning. Execute the protected operation after a valid enforce-mode authorization and proof verification.

JSON

```
{
  "type": "about:blank",
  "title": "nonce reused",
  "status": 409,
  "detail": "The nonce was already used in this environment.",
  "code": "error.nonce_reused"
}
```

Authentication middleware, a reverse proxy, or a transport interruption may produce a response outside this JSON shape. Handle transport and parsing responses alongside API problem documents. The Node SDK raises `GrayPassApiError` for HTTP errors and also surfaces transport failures.

Validation responses

Canonical request-validation failures return HTTP 422 with `application/problem+json` and the `Problem` fields above. The stable code is `error.context_invalid`; `detail` identifies the invalid input. Correct the request before retrying. The downloadable OpenAPI and operation tables describe this response directly for generated clients.

Status classes

Status	Typical interpretation	Integration response
400 or 422	Malformed input or schema violation	Correct the request; do not blindly retry
401	Missing or invalid authentication	Obtain a valid credential through the correct authenticated path
403	Scope, caller, origin, audience, or environment restriction	Verify configuration and ownership; do not weaken checks
404	Object absent or not visible in the caller's boundary	Confirm the tenant, environment, and identifier
409	Request, lifecycle, idempotency, or concurrent-state conflict	Re-read state and follow the specific error code
429	Rate limit	Back off with jitter and preserve request identity
5xx	Service or dependency failure	Keep the operation unexecuted; retry only with safe idempotency

An authorization returning HTTP success may still contain challenge, hold, or deny. Conversely, the Express helper's HTTP responses are application responses and are not the same thing as raw authorization endpoint status codes. Distinguish transport success, policy decision, and business execution in your logs and metrics.

Actionable error families

Credential and isolation errors include `error.browser_credential_not_allowed`, `error.environment_mismatch`, `error.caller_not_allowed`, and `error.not_found`. Action configuration errors include `error.action_not_registered`, `error.policy_inactive`, `error.context_invalid`, and `error.audience_not_allowed`.

Request identity errors include `error.request_stale`, `error.nonce_reused`, `error.idempotency_conflict`, `error.request_in_progress`, and `error.idempotency_stale`. Lifecycle and proof errors include subject enrollment failures, challenge expiry or mismatch, unavailable signing, and inactive proof state.

`error.enforcement_not_approved` means the requested enforcement configuration was rejected; preserve the existing mode and contact support with the environment and trace ID. `error.commercial_access_unavailable` requires resolution of the account's commercial access before using the requested live service; keep the protected operation unexecuted.

See the generated [Reason codes](#) for the contract vocabulary and [Troubleshooting](#) for corrective procedures. Verifier failure reasons are separately documented in [Proofs](#).

Safe diagnostics

Record a timestamp, environment, operation, error code, and safe authorization or trace identifier. Do not record raw credentials, proofs, full action context containing sensitive material, or webhook secrets. If a response is ambiguous after consumption or a business side effect, consult the application's durable operation record before retrying execution.

API REFERENCE

Subject API

These operations create and manage pseudonymous subjects within the authenticated tenant and environment. Your backend resolves the subject from its own authenticated account mapping. Validate account ownership before accepting a subject identifier.

Create a subject with a stable external reference and the anchor established by your enrollment process. Begin an enrollment session through the session API and complete enrollment only after server readiness. Passkey registration uses a server-issued one-time grant. Revocation, recovery, adaptation freeze, export, and deletion are distinct lifecycle actions; choose the operation that matches your process.

Lifecycle constraints

The recovery policy controls the supported recovery path. Your application must establish the declared anchor and any required evidence reference through its own account-security process. GrayPass records the declared anchor; your account-security process establishes its provenance.

Record research consent separately from subject creation and ordinary product use. Deletion and revocation should also stop the customer's browser observer and update pending operations in the customer application.

Recording research consent returns the current `Subject`; the receipt is stored separately. Keep your own record of the person's choice and disclosure. Environment participation, current subject consent, per-frame opt-in and deployment eligibility all matter. Handle queued webhook deliveries and recipient copies through the separate delivery lifecycle described in [Privacy](#).

Subject exports return up to 500 records per listed collection and up to 100 enrollments in one response. Use the [export procedure](#) to match these bounds to an individual request and arrange additional records. Coordinate financial, audit, and backup handling through the agreed privacy process.

Read [Subject lifecycle](#) and [Privacy](#) before implementing these operations. The operation contracts below specify fields, credential requirements, parameters, and responses.

GET /api/v1/subjects

Read a bounded list of non-deleted subjects visible in this environment.

Credential: Server credential.

Inputs: Optional limit defaults 100 and is bounded to 1-500; there is no cursor parameter.

Effect and retry behavior: Read-only. Deleted records are filtered, so the returned count may be below the limit.

Parameters

Name / location	Required	Meaning, type and constraints
limit (query)	No	Maximum records requested; default 100. The service bounds it to 1-500. No cursor pagination is exposed. integer default: 100

Responses

HTTP status	Meaning	Body
200	Array of visible non-deleted subject records, up to the effective limit.	application/json: Array of Subject
422	The canonical handler returns a Problem for invalid input; correct the request before retrying.	application/problem+json: Problem

Response handling: Wrong credential scope or environment prevents access; an empty list is not evidence that another environment has no subjects.

Example

HTTP

```
GET /api/v1/subjects?limit=25
```

Response - HTTP 200:

```
[
  {
    "id": "sub_example",
    "environment": "env_example",
    "enrollment_state": "not_enrolled",
    "anchor": "customer_session",
    "recovery_policy": "re_enroll_with_anchor",
    "adaptation_frozen": true,
    "created_at": "2026-10-01T12:00:00Z",
    "revoked_at": null,
    "limitations": []
  }
]
```

POST /api/v1/subjects

Create the account-to-subject mapping in this environment, or return an existing subject with compatible security settings.

Credential: Server credential.

Inputs: Supply a stable external_ref. Stronger declared anchors require anchor_evidence_ref; only metadata.label is retained.

Effect and retry behavior: Stores a keyed reference and subject record. Identical reference/security settings return the existing subject; changed anchor or recovery policy conflicts.

Request body

application/json; required. [SubjectCreate](#).

Responses

HTTP status	Meaning	Body
201	New or existing subject record with enrollment state and recovery settings.	application/json: Subject
422	The canonical handler returns a Problem for invalid input; correct the request before retrying.	application/problem+json: Problem

Response handling: 422 for missing required anchor evidence; 409 idempotency_conflict for a reference already bound to different security settings.

Example

HTTP

```
POST /api/v1/subjects
```

JSON request

```
{
  "external_ref": "account-42",
  "anchor": "customer_session",
  "recovery_policy": "re_enroll_with_anchor"
}
```

Response - HTTP 201:

```
{
  "id": "sub_example",
  "environment": "env_example",
  "enrollment_state": "not_enrolled",
  "anchor": "customer_session",
  "recovery_policy": "re_enroll_with_anchor",
  "adaptation_frozen": true,
  "created_at": "2026-10-01T12:00:00Z",
  "revoked_at": null,
  "limitations": []
}
```

DELETE /api/v1/subjects/{subject_id}

Delete the subject's operational assurance data and make its record unavailable to normal subject lookup.

Credential: Server credential.

Inputs: Use the authenticated account's subject id after confirming the intended deletion in your application.

Effect and retry behavior: Destructive lifecycle operation that invalidates usable assurance. Financial, audit and backup retention are separate; see privacy and subject lifecycle.

Parameters

Name / location	Required	Meaning, type and constraints
subject_id (path)	Yes	Subject identifier resolved from your authenticated account mapping. string

Responses

HTTP status	Meaning	Body
204	Operational deletion completed; the 204 response has no body.	No response body
422	The canonical handler returns a Problem for invalid input; correct the request before retrying.	application/problem+json: Problem

Response handling: Reconcile state after an ambiguous response. A subsequent 404 can confirm completed deletion. Create a new subject only through an explicit application decision.

Example

HTTP

```
DELETE /api/v1/subjects/sub_example
```

HTTP 204; no response body.

GET /api/v1/subjects/{subject_id}

Retrieve the current subject enrollment and recovery state before starting a lifecycle operation.

Credential: Server credential.

Inputs: Use your stored subject identifier.

Effect and retry behavior: Read-only. Revoked records remain distinguishable from deleted or absent records.

Parameters

Name / location	Required	Meaning, type and constraints
subject_id (path)	Yes	Subject identifier resolved from your authenticated account mapping. string

Responses

HTTP status	Meaning	Body
200	Current subject state, declared anchor, recovery policy and limitations.	application/json: Subject

HTTP status	Meaning	Body
422	The canonical handler returns a Problem for invalid input; correct the request before retrying.	application/problem+json: Problem

Response handling: 404 for a missing, deleted or out-of-scope subject.

Example

HTTP

```
GET /api/v1/subjects/sub_example
```

Response - HTTP 200:

```
{
  "id": "sub_example",
  "environment": "env_example",
  "enrollment_state": "not_enrolled",
  "anchor": "customer_session",
  "recovery_policy": "re_enroll_with_anchor",
  "adaptation_frozen": true,
  "created_at": "2026-10-01T12:00:00Z",
  "revoked_at": null,
  "limitations": []
}
```

POST /api/v1/subjects/{subject_id}/adaptation/freeze

Ensure the subject's existing template is not silently updated through online adaptation.

Credential: Server credential.

Inputs: Subject id only; no request body.

Effect and retry behavior: Sets adaptation_frozen to true while preserving the subject, template and evidence collection. Repeated freezing is safe. Use the dedicated revoke or delete operations for those lifecycle changes.

Parameters

Name / location	Required	Meaning, type and constraints
subject_id (path)	Yes	Subject identifier resolved from your authenticated account mapping. string

Responses

HTTP status	Meaning	Body
200	Subject record with adaptation_frozen true.	application/json: Subject

HTTP status	Meaning	Body
422	The canonical handler returns a Problem for invalid input; correct the request before retrying.	application/problem+json: Problem

Response handling: 404 for a missing or out-of-scope subject.

Example

HTTP

```
POST /api/v1/subjects/sub_example/adaptation/freeze
```

Response - HTTP 200:

```
{
  "id": "sub_example",
  "environment": "env_example",
  "enrollment_state": "not_enrolled",
  "anchor": "customer_session",
  "recovery_policy": "re_enroll_with_anchor",
  "adaptation_frozen": true,
  "created_at": "2026-10-01T12:00:00Z",
  "revoked_at": null,
  "limitations": []
}
```

POST /api/v1/subjects/{subject_id}/enrollments/complete

Create a Brainprint enrollment from the subject's accepted enrollment-session evidence.

Credential: Server credential.

Inputs: Supply the owned enroll session after its evidence acknowledgement or status reports enrollment_ready=true. The server rechecks readiness at completion; accepted-window count, active seconds and modalities support progress reporting, not client-side thresholds. force defaults false.

Effect and retry behavior: Creates an enrollment attempt and may replace an existing template when force is true. Readiness requirements apply in both cases. Reconcile the subject state before retrying an uncertain response.

Parameters

Name / location	Required	Meaning, type and constraints
subject_id (path)	Yes	Subject identifier resolved from your authenticated account mapping. string

Request body

application/json; required. [EnrollmentComplete](#).

Responses

HTTP status	Meaning	Body
200	Enrollment attempt status, accepted-window count and completion time.	application/json: Enrollment
422	The canonical handler returns a Problem for invalid input; correct the request before retrying.	application/problem+json: Problem

Response handling: 409 for revoked subjects or insufficient readiness; 404 for mismatched subject/session ownership or purpose. Inspect the returned enrollment status.

Example

HTTP

```
POST /api/v1/subjects/sub_example/enrollments/complete
```

JSON request

```
{
  "session": "ses_example",
  "force": false
}
```

Response - HTTP 200:

```
{
  "id": "enr_example",
  "subject": "sub_example",
  "status": "completed",
  "anchor": "customer_session",
  "windows_accepted": 5,
  "created_at": "2026-10-01T12:00:00Z",
  "completed_at": "2026-10-01T12:00:00Z",
  "limitations": []
}
```

POST /api/v1/subjects/{subject_id}/export

Retrieve a bounded subject-linked export containing lifecycle records and whether a Brainprint exists.

Credential: Server credential.

Inputs: Use the subject id. No body is required.

Effect and retry behavior: Read-only export; no template vector or passkey credential material is returned. Several collections stop at 500 records and there is no complete-history pagination protocol.

Parameters

Name / location	Required	Meaning, type and constraints
subject_id (path)	Yes	Subject identifier resolved from your authenticated account mapping. string

Responses

HTTP status	Meaning	Body
200	Bounded subject, Brainprint-presence, enrollment, authorization, session, passkey and consent records.	application/json: object
422	The canonical handler returns a Problem for invalid input; correct the request before retrying.	application/problem+json: Problem

Response handling: 404 for an absent subject. Match the returned record bounds to the individual request and arrange additional history through the privacy process when needed.

Example

HTTP

```
POST /api/v1/subjects/sub_example/export
```

Response excerpt (selected fields) - HTTP 200:

```
{
  "subject": {
    "id": "sub_example"
  },
  "brainprint": {
    "present": false,
    "template_exported": false,
    "reason": "Templates are not exportable; they are deleted on request."
  },
  "sessions": [],
  "research_consent": []
}
```

POST /api/v1/subjects/{subject_id}/passkeys/register/complete

Verify the WebAuthn registration result and register the resulting credential for the subject.

Credential: Scoped browser token or server credential.

Inputs: Send registration_id from options.graypass_registration_id and credential as the browser's complete serialized PublicKeyCredential attestation response. The example shows the outer container; use the SDK to supply the full WebAuthn fields within it.

Effect and retry behavior: Stores the verified passkey for subsequent action-bound challenge ceremonies. Request action approval through a challenge.

Parameters

Name / location	Required	Meaning, type and constraints
subject_id (path)	Yes	Subject identifier resolved from your authenticated account mapping. string

Request body

application/json; required. object.

Responses

HTTP status	Meaning	Body
200	Registered passkey id and creation time.	application/json: PasskeyRegistered
422	The canonical handler returns a Problem for invalid input; correct the request before retrying.	application/problem+json: Problem

Response handling: 422 when registration_id is absent; expired/mismatched ceremony, RP/origin, attestation or credential failures are refused.

Example

HTTP

```
POST /api/v1/subjects/sub_example/passkeys/register/complete
```

JSON request

```
{
  "registration_id": "pkr_example",
  "credential": {
    "id": "illustrative-credential-id",
    "type": "public-key"
  }
}
```

Response - HTTP 200:

```
{
  "id": "pky_example",
  "created_at": "2026-10-01T12:00:00Z"
}
```

POST /api/v1/subjects/{subject_id}/passkeys/register/options

Consume a server-issued registration grant and return WebAuthn creation options for that subject.

Credential: Scoped browser token or server credential.

Inputs: Send registration_grant obtained by the authenticated backend. A browser caller must also match subject and origin scope.

Effect and retry behavior: Consumes the grant and stores a registration ceremony. The response is the WebAuthn creation-options object with graypass_registration_id added. Preserve that id for completion and use the browser SDK for conversion.

Parameters

Name / location	Required	Meaning, type and constraints
subject_id (path)	Yes	Subject identifier resolved from your authenticated account mapping. string

Request body

application/json; required. [PasskeyRegistrationOptionsRequest](#).

Responses

HTTP status	Meaning	Body
200	WebAuthn creation options with graypass_registration_id for the completion call.	application/json: object
422	The canonical handler returns a Problem for invalid input; correct the request before retrying.	application/problem+json: Problem

Response handling: Expired, reused or mismatched grants and disallowed origins are refused. Request a new backend grant for a genuinely new ceremony.

Example

HTTP

```
POST /api/v1/subjects/sub_example/passkeys/register/options
```

JSON request

```
{
  "registration_grant": "illustrative-registration-grant-not-valid"
}
```

Response excerpt (selected fields) - HTTP 200:

```
{
  "graypass_registration_id": "pkr_example",
  "rp": {
    "name": "GrayPass"
  },
  "timeout": 60000
}
```

POST /api/v1/subjects/{subject_id}/passkeys/registration-grants

Authorize one subject-bound passkey registration ceremony from your authenticated backend.

Credential: Server credential.

Inputs: Subject id only. Obtain application consent and validate account ownership before issuing the grant.

Effect and retry behavior: Issues an expiring one-time grant. It is consumed when registration options are requested; a retry may create a separate grant.

Parameters

Name / location	Required	Meaning, type and constraints
subject_id (path)	Yes	Subject identifier resolved from your authenticated account mapping. string

Responses

HTTP status	Meaning	Body
201	One-time registration token and ISO 8601 expiry.	application/json: PasskeyRegistrationGrant
422	The canonical handler returns a Problem for invalid input; correct the request before retrying.	application/problem+json: Problem

Response handling: Proceed with registration after subject lifecycle and scope checks pass and the grant is stored successfully.

Example

HTTP

```
POST /api/v1/subjects/sub_example/passkeys/registration-grants
```

Response excerpt (selected fields) - HTTP 201:

```
{
  "expires_at": "2026-10-01T12:05:00Z"
}
```

DELETE /api/v1/subjects/{subject_id}/passkeys/{passkey_id}

Remove a subject passkey from eligibility for future GrayPass action challenges.

Credential: Server credential.

Inputs: Provide the owning subject id and registered passkey id.

Effect and retry behavior: Revokes the credential within GrayPass. Authenticator removal and subject recovery follow their separate lifecycle procedures.

Parameters

Name / location	Required	Meaning, type and constraints
subject_id (path)	Yes	Subject identifier resolved from your authenticated account mapping. string
passkey_id (path)	Yes	Identifier of the subject's registered passkey to revoke. string

Responses

HTTP status	Meaning	Body
204	Passkey revocation completed; the 204 response has no body.	No response body
422	The canonical handler returns a Problem for invalid input; correct the request before retrying.	application/problem+json: Problem

Response handling: Missing, out-of-scope or mismatched subject/passkey records are refused.

Example

HTTP

```
DELETE /api/v1/subjects/sub_example/passkeys/pky_example
```

HTTP 204; no response body.

POST /api/v1/subjects/{subject_id}/recovery

Move a revoked subject into the permitted re-enrollment path after your application establishes the required anchor.

Credential: Server credential.

Inputs: Specify the supported anchor and required anchor_evidence_ref. The subject's stored recovery policy controls eligibility.

Effect and retry behavior: Begins a fresh enrollment lifecycle. Complete enrollment to create a new template, then authorize subsequent actions separately.

Parameters

Name / location	Required	Meaning, type and constraints
subject_id (path)	Yes	Subject identifier resolved from your authenticated account mapping. string

Request body

application/json; required. [RecoveryBegin](#).

Responses

HTTP status	Meaning	Body
200	Subject reset to not_enrolled under the accepted replacement anchor.	application/json: Subject
422	The canonical handler returns a Problem for invalid input; correct the request before retrying.	application/problem+json: Problem

Response handling: 409 when the subject is not revoked or its policy requires manual review; 422 when required anchor evidence is absent.

Example

HTTP

```
POST /api/v1/subjects/sub_example/recovery
```

JSON request

```
{
  "anchor": "customer_session"
}
```

Response - HTTP 200:

```
{
  "id": "sub_example",
  "environment": "env_example",
  "enrollment_state": "not_enrolled",
  "anchor": "customer_session",
  "recovery_policy": "re_enroll_with_anchor",
  "adaptation_frozen": true,
  "created_at": "2026-10-01T12:00:00Z",
  "revoked_at": null,
  "limitations": []
}
```

POST /api/v1/subjects/{subject_id}/research-consent

Record a separate, attributed subject decision about behavioral-model research participation.

Credential: Server credential.

Inputs: consented must be a JSON boolean. The API fixes purpose behavioral_model_research and disclosure version research-consent/1.

Effect and retry behavior: Appends a granted or revoked consent receipt; repeated calls append receipts. Research retention also requires environment participation, per-frame opt-in and deployment eligibility.

Parameters

Name / location	Required	Meaning, type and constraints
subject_id (path)	Yes	Subject identifier resolved from your authenticated account mapping. string

Request body

application/json; required. [ResearchConsentRequest](#).

Responses

HTTP status	Meaning	Body
200	Current subject record. The consent receipt is stored separately; session research eligibility is reported when creating a session.	application/json: Subject
422	The canonical handler returns a Problem for invalid input; correct the request before retrying.	application/problem+json: Problem

Response handling: 404 for an unavailable subject; 422 for a non-boolean value. Research collection also requires deployment and environment enablement and per-frame opt-in.

Example

HTTP

```
POST /api/v1/subjects/sub_example/research-consent
```

JSON request

```
{
  "consented": false
}
```

Response - HTTP 200:

```
{
  "id": "sub_example",
  "environment": "env_example",
  "enrollment_state": "not_enrolled",
  "anchor": "customer_session",
  "recovery_policy": "re_enroll_with_anchor",
  "adaptation_frozen": true,
  "created_at": "2026-10-01T12:00:00Z",
  "revoked_at": null,
  "limitations": []
}
```

POST /api/v1/subjects/{subject_id}/revoke

Revoke the subject's existing assurance and terminate its usable session and proof capabilities.

Credential: Server credential.

Inputs: Use the subject selected through your authenticated account-security workflow.

Effect and retry behavior: Changes enrollment state to revoked, removes usable template assurance and invalidates associated sessions and proofs. Stop the observer in your application too.

Parameters

Name / location	Required	Meaning, type and constraints
subject_id (path)	Yes	Subject identifier resolved from your authenticated account mapping. string

Responses

HTTP status	Meaning	Body
200	Subject record reflecting revoked enrollment state and revocation time.	application/json: Subject
422	The canonical handler returns a Problem for invalid input; correct the request before retrying.	application/problem+json: Problem

Response handling: Missing or out-of-scope subjects fail. Resolve any ambiguous response by reading lifecycle state before initiating recovery.

Example

HTTP

```
POST /api/v1/subjects/sub_example/revoke
```

Response - HTTP 200:

```
{
  "id": "sub_example",
  "environment": "env_example",
  "enrollment_state": "revoked",
  "anchor": "customer_session",
  "recovery_policy": "re_enroll_with_anchor",
  "adaptation_frozen": true,
  "created_at": "2026-10-01T12:00:00Z",
  "revoked_at": "2026-10-01T12:00:00Z",
  "limitations": []
}
```

API REFERENCE

Sessions and evidence API

A session belongs to a subject and has either `enroll` or `observe` purpose. Prefer the browser SDK for collection and signed evidence submission. It handles frame-signing material, feature extraction, windowing, and the supported transport lifecycle. Collect evidence from the authenticated user's interaction with your application.

`FeatureFrameRequest` defines the JSON window envelope; the SDK supplies its signing and modality handling. SDK frames use `purpose: "enroll"` for enrollment and `purpose: "score"` for observation. Use the separate consent workflow before setting the per-frame JSON field `research_optin`. The session's `research_enabled` flag reports environment and subject eligibility at session creation; storage also requires current consent and frame opt-in.

Status and assessment are different

Browser-safe session status reports accepted coverage, active seconds, modalities, freshness, readiness, and limitations. Use the server assessment endpoint for the full Assurance Profile. Complete enrollment when the server reports readiness.

Status reads can record qualifying Active Brainprint usage in a live environment. The browser-safe response contains the fields needed by the browser workflow.

The server-only assessment operation returns an Assurance Profile. **Retrieving an assessment can record qualifying assurance for commercial usage.** Call it when the integration needs an assessment. Use the health endpoints for monitoring, and load assessments through an intentional application request. Review [Usage and billing](#) before using it in live workflows.

Evidence integrity and shutdown

The service validates signatures, frame identity, time windows, shape, and evidence eligibility before accepting a frame. Accepted evidence contributes to coverage accounting. Signing protects session transport; protect the browser evidence producer through your page and endpoint controls.

End sessions when observation stops or the authenticated account changes. Stop an optional camera controller separately if it is shared with the observer. The operations below describe session creation, evidence submission, safe status, assessment, and session end. See [Browser SDK](#) for lifecycle methods and [Data flow](#) for collection boundaries.

POST /api/v1/sessions

Start an enroll or observe session for a subject using the scoped browser workflow.

Credential: Scoped browser token or server credential.

Inputs: `purpose` defaults `observe`. Optional `device_context` records application-supplied context. Validate endpoint trust through your application's attestation controls.

Effect and retry behavior: Creates a new session and scoped signing material. Enrollment sessions update enrollment lifecycle; retrying can create another session.

Request body

application/json; required. [SessionCreate](#).

Responses

HTTP status	Meaning	Body
201	Session id, purpose, feature version, scoped signing material and research eligibility flag.	application/json: SessionCreated
422	The canonical handler returns a Problem for invalid input; correct the request before retrying.	application/problem+json: Problem

Response handling: Revoked or unavailable subjects and mismatched browser-token subject/origin are refused.

Example

HTTP

```
POST /api/v1/sessions
```

JSON request

```
{
  "subject": "sub_example",
  "purpose": "enroll"
}
```

Response excerpt (selected fields) - HTTP 201:

```
{
  "id": "ses_example",
  "subject": "sub_example",
  "purpose": "enroll",
  "research_enabled": false,
  "created_at": "2026-10-01T12:00:00Z"
}
```

GET /api/v1/sessions/{session_id}/assessment

Build the current server assessment for the session and its subject.

Credential: Server credential.

Inputs: Server credential and session id. Use browser-safe status for progress UI.

Effect and retry behavior: This GET can record qualifying billable assurance. Request it for its assessment result and use health endpoints for monitoring; see usage and billing.

Usage side effect: this GET can record qualifying billable assurance. See [usage rules](#).

Parameters

Name / location	Required	Meaning, type and constraints
session_id (path)	Yes	Session identifier owned by the scoped subject in the authenticated environment. string

Responses

HTTP status	Meaning	Body
200	Server Assurance Profile with claim statuses, evidence freshness and limitations.	application/json: AssuranceProfile
422	The canonical handler returns a Problem for invalid input; correct the request before retrying.	application/problem+json: Problem

Response handling: 404 when session or subject state is unavailable. Read `insufficient_evidence` in the profile as a collection or approval requirement, and follow the returned policy decision.

Example

HTTP

```
GET /api/v1/sessions/ses_example/assessment
```

Response - HTTP 200:

```
{
  "id": "asr_example",
  "subject": "sub_example",
  "session": "ses_example",
  "assessed_at": "2026-10-01T12:00:00Z",
  "claims": [],
  "coverage": "none",
  "evidence_windows": 0,
  "active_seconds": 0.0,
  "modalities_present": [],
  "freshness_ms": null,
  "limitations": [
    "insufficient evidence"
  ],
  "model_version": "example-model-version",
  "rule_version": "example-rule-version",
  "feature_schema_version": "example-feature-version",
  "insufficient_evidence": true
}
```

POST /api/v1/sessions/{session_id}/end

Stop accepting further evidence in the selected session and return its accounting summary.

Credential: Scoped browser token or server credential.

Inputs: Use the session id and stop the customer observer when the session or signed-in account ends.

Effect and retry behavior: Marks the session ended while preserving the subject and template. Use the subject lifecycle operations for revocation or deletion.

Parameters

Name / location	Required	Meaning, type and constraints
session_id (path)	Yes	Session identifier owned by the scoped subject in the authenticated environment. string

Responses

HTTP status	Meaning	Body
200	ended receipt with elapsed session seconds and accounted evidence-window count.	application/json: SessionEnded
422	The canonical handler returns a Problem for invalid input; correct the request before retrying.	application/problem+json: Problem

Response handling: Missing or out-of-scope sessions are refused; reconcile an uncertain response rather than continuing to submit frames.

Example

HTTP

```
POST /api/v1/sessions/ses_example/end
```

Response - HTTP 200:

```
{
  "session": "ses_example",
  "status": "ended",
  "duration_seconds": 30,
  "evidence_windows": 5
}
```

POST /api/v1/sessions/{session_id}/evidence

Submit a session-signed feature window and receive accepted coverage accounting.

Credential: Scoped browser token or server credential.

Inputs: Use the browser SDK's exact-body signing and feature transport. JSON fields alone are insufficient. Match enroll/score purpose to the session; research_optin defaults false.

Effect and retry behavior: May contribute accepted non-overlapping evidence to coverage and enrollment readiness. Duplicate frames never count as fresh evidence.

Parameters

Name / location	Required	Meaning, type and constraints
session_id (path)	Yes	Session identifier owned by the scoped subject in the authenticated environment. string

Request body

application/json; required. [FeatureFrameRequest](#).

Responses

HTTP status	Meaning	Body
200	accepted, neutral or rejected status plus coverage, reasons and enrollment readiness.	application/json: EvidenceAck
422	The canonical handler returns a Problem for invalid input; correct the request before retrying.	application/problem+json: Problem

Response handling: Unsigned, stale, replayed, overlapping, malformed or purpose-mismatched windows are refused or neutralized. Opt-in without required consent/environment eligibility returns 422.

Example

HTTP

```
POST /api/v1/sessions/ses_example/evidence
```

JSON request

```
{
  "frame_id": "frame-example-7",
  "timestamp": 1790856000000,
  "duration_ms": 5000,
  "window_start": 1790855995,
  "window_end": 1790856000,
  "features": {},
  "purpose": "enroll",
  "research_optin": false
}
```

Response - HTTP 200:

```
{
  "status": "neutral",
  "coverage": "none",
  "evidence_windows": 0,
  "active_seconds": 0,
  "modalities_present": [],
  "reason_codes": [],
  "enrollment_ready": false
}
```

GET /api/v1/sessions/{session_id}/status

Inspect accepted evidence coverage, freshness and enrollment readiness without exposing a full server assessment.

Credential: Scoped browser token or server credential.

Inputs: Use the subject's scoped session id.

Effect and retry behavior: Returns status without changing enrollment state or authorizing a business action. A qualifying assurance profile can record Active Brainprint usage under the agreement's shadow-use terms. The enrollment_ready flag allows your backend to request enrollment completion.

Parameters

Name / location	Required	Meaning, type and constraints
session_id (path)	Yes	Session identifier owned by the scoped subject in the authenticated environment. string

Responses

HTTP status	Meaning	Body
200	Browser-safe coverage, accepted-window count, active duration, freshness and readiness.	application/json: StatusSummary
422	The canonical handler returns a Problem for invalid input; correct the request before retrying.	application/problem+json: Problem

Response handling: Use an active session belonging to the authenticated subject and environment. Inspect the response code when that session needs renewal or correction.

Example

HTTP

```
GET /api/v1/sessions/ses_example/status
```

Response - HTTP 200:

```
{
  "session": "ses_example",
  "coverage": "partial",
  "evidence_windows": 5,
  "active_seconds": 25,
  "modalities_present": [
    "keyboard"
  ],
  "freshness_ms": 500,
  "human_readable": "Evidence is available.",
  "enrollment_ready": true,
  "limitations": []
}
```

API REFERENCE

Actions and policy API

An action registers the exact consequential operation the customer may request. A policy version snapshots its security-relevant constraints: context schema, allowed audiences and callers, proof lifetime, assurance requirements, challenge behavior, and hold or deny conditions.

Administrative flow

Use an admin-scoped credential for action creation, supported updates, archival, policy creation, and activation. Runtime services ordinarily use server scope to read the relevant configuration and request decisions. Use the least privileged credential that covers each operation.

Create a version, review it, and activate it explicitly. Policy content is immutable and integrity checked. Activation changes the active version transactionally. Select a reviewed prior version for rollback rather than modifying immutable policy content.

Request constraints

Register a context schema that includes every security-relevant value needed to distinguish the operation. Use safe integers and the supported schema features for canonical action material. Validate business account ownership alongside the request schema.

New actions start in shadow mode. Use test and shadow decisions to inspect policy behavior. An eligible enforce-mode allow supplies a proof for backend verification and consumption. Check the returned mode after configuration changes and handle a rejected change without advancing the protected operation. See [Actions and policies](#) for mode configuration, policy examples, and decision precedence. The operations below specify the request and response contracts.

GET /api/v1/actions

Read non-archived action definitions in this environment.

Credential: Server credential.

Inputs: No body or pagination parameters; the list contains at most 500 non-archived actions.

Effect and retry behavior: Read-only bounded list. Use the returned record id for action administration and the key for authorization.

Responses

HTTP status	Meaning	Body
200	Array of up to 500 non-archived protected actions.	application/json: Array of ProtectedAction

Response handling: An empty list means the query returned no visible registered actions. Register and configure the intended action before authorization.

Example

HTTP

```
GET /api/v1/actions
```

Response excerpt (selected fields) - HTTP 200:

```
[
  {
    "id": "act_example",
    "key": "payout.destination.change",
    "mode": "shadow"
  }
]
```

POST /api/v1/actions

Create an action definition and its initial active policy in shadow mode.

Credential: Admin server credential.

Inputs: Supply a dotted key, display name, consequence and nonempty allowed_audiences. Use context_schema to bind sensitive business fields.

Effect and retry behavior: Creates a new action and initial policy in shadow mode. Defaults include risk_class high and proof_ttl_seconds 300. Configure the action mode separately.

Request body

application/json; required. [ActionCreate](#).

Responses

HTTP status	Meaning	Body
201	ProtectedAction in shadow mode with its initial active policy version.	application/json: ProtectedAction
422	The canonical handler returns a Problem for invalid input; correct the request before retrying.	application/problem+json: Problem

Response handling: Conflicting keys, invalid context schemas and unsupported settings are refused. Read existing configuration before retrying a create after a timeout.

Example

HTTP

```
POST /api/v1/actions
```

JSON request

```
{
  "key": "payout.destination.change",
  "name": "Change payout destination",
  "consequence": "Future payouts may be sent to a different destination.",
  "resource_type": "merchant",
  "allowed_audiences": [
    "merchant-api"
  ],
  "context_schema": {
    "type": "object",
    "properties": {
      "destination_fingerprint": {
        "type": "string"
      }
    }
  },
  "required": [
    "destination_fingerprint"
  ],
  "additionalProperties": false
}
```

Response excerpt (selected fields) - HTTP 201:

```
{
  "id": "act_example",
  "key": "payout.destination.change",
  "mode": "shadow",
  "active_policy_version": 1
}
```

DELETE /api/v1/actions/{action_id}

Stop using an action through the normal active action surface while retaining its historical records.

Credential: Admin server credential.

Inputs: Action record id only.

Effect and retry behavior: Archives the action. Historical authorizations and policies are not rewritten; integrations must handle subsequent refusal.

Parameters

Name / location	Required	Meaning, type and constraints
action_id (path)	Yes	Protected-action record identifier returned by action creation. Supply this record ID in the path; use the dotted action key in authorization requests. string

Responses

HTTP status	Meaning	Body
204	Action archival completed; the 204 response has no body.	No response body
422	The canonical handler returns a Problem for invalid input; correct the request before retrying.	application/problem+json: Problem

Response handling: Missing or unavailable actions are refused. Reconcile uncertain archive responses through action state before accepting new business requests.

Example

HTTP

```
DELETE /api/v1/actions/act_example
```

HTTP 204; no response body.

GET /api/v1/actions/{action_id}

Inspect the action's display metadata, mode and active policy settings before integration or a configuration change.

Credential: Server credential.

Inputs: Use the action record id, not its key.

Effect and retry behavior: Read-only; compare active_policy_version when reviewing changes.

Parameters

Name / location	Required	Meaning, type and constraints
action_id (path)	Yes	Protected-action record identifier returned by action creation. Supply this record ID in the path; use the dotted action key in authorization requests. string

Responses

HTTP status	Meaning	Body
200	ProtectedAction with mode and effective active policy settings.	application/json: ProtectedAction
422	The canonical handler returns a Problem for invalid input; correct the request before retrying.	application/problem+json: Problem

Response handling: 404 for missing, archived or out-of-scope actions.

Example

HTTP

```
GET /api/v1/actions/act_example
```

Response excerpt (selected fields) - HTTP 200:

```
{
  "id": "act_example",
  "key": "payout.destination.change",
  "mode": "shadow",
  "active_policy_version": 1
}
```

PATCH /api/v1/actions/{action_id}

Change an action's customer-facing name, consequence or test/shadow/enforce mode.

Credential: Admin server credential.

Inputs: Omit fields that should remain unchanged. Change audiences, caller restrictions, context or proof lifetime through a new policy version.

Effect and retry behavior: Updates the supplied action metadata or mode. Read the returned mode to confirm the effective configuration.

Parameters

Name / location	Required	Meaning, type and constraints
action_id (path)	Yes	Protected-action record identifier returned by action creation. Supply this record ID in the path; use the dotted action key in authorization requests. string

Request body

application/json; required. [ActionUpdate](#).

Responses

HTTP status	Meaning	Body
200	Updated ProtectedAction metadata and current mode.	application/json: ProtectedAction
422	The canonical handler returns a Problem for invalid input; correct the request before retrying.	application/problem+json: Problem

Response handling: For error.enforcement_not_approved, retain the effective mode and confirm the environment configuration with GrayPass. Action display metadata and immutable policy versions are configured separately.

Example

HTTP

```
PATCH /api/v1/actions/act_example
```

JSON request

```
{
  "name": "Change merchant payout destination"
}
```

Response excerpt (selected fields) - HTTP 200:

```
{
  "id": "act_example",
  "key": "payout.destination.change",
  "mode": "shadow"
}
```

GET /api/v1/actions/{action_id}/policy-versions

Inspect the action's immutable draft, active and retired policy snapshots.

Credential: Server credential.

Inputs: Action record id only; no pagination cursor.

Effect and retry behavior: Read-only. Only the active version controls new decisions.

Parameters

Name / location	Required	Meaning, type and constraints
action_id (path)	Yes	Protected-action record identifier returned by action creation. Supply this record ID in the path; use the dotted action key in authorization requests. string

Responses

HTTP status	Meaning	Body
200	Array of bounded policy snapshots with version numbers and lifecycle states.	application/json: Array of PolicyVersion
422	The canonical handler returns a Problem for invalid input; correct the request before retrying.	application/problem+json: Problem

Response handling: 404 for an unavailable action. Use the active policy version for a new authorization.

Example

HTTP

```
GET /api/v1/actions/act_example/policy-versions
```

Response excerpt (selected fields) - HTTP 200:

```
[
  {
    "id": "pol_example",
    "action": "act_example",
    "version": 1,
    "status": "active"
  }
]
```

POST /api/v1/actions/{action_id}/policy-versions

Prepare a new immutable policy snapshot without activating it.

Credential: Admin server credential.

Inputs: requirements is required. Omitted optional audiences, caller restrictions, context schema and proof lifetime inherit from the active version.

Effect and retry behavior: Creates a numbered draft; existing authorizations retain their original snapshot. Activate explicitly after review.

Parameters

Name / location	Required	Meaning, type and constraints
action_id (path)	Yes	Protected-action record identifier returned by action creation. Supply this record ID in the path; use the dotted action key in authorization requests. string

Request body

application/json; required. [PolicyVersionCreate](#).

Responses

HTTP status	Meaning	Body
201	New immutable draft policy and its assigned version number.	application/json: PolicyVersion
422	The canonical handler returns a Problem for invalid input; correct the request before retrying.	application/problem+json: Problem

Response handling: Invalid claim requirements, challenge methods, audiences or context schema are refused. A retry can create another draft; list versions after uncertainty.

Example

HTTP

```
POST /api/v1/actions/act_example/policy-versions
```

JSON request

```
{
  "requirements": {
    "coverage_min": "partial",
    "freshness_max_ms": 60000,
    "insufficient_evidence": "challenge",
    "challenge_methods": [
      "passkey"
    ]
  }
}
```

Response excerpt (selected fields) - HTTP 201:

```
{
  "id": "pol_example",
  "action": "act_example",
  "version": 2,
  "status": "draft"
}
```

POST /api/v1/actions/{action_id}/policy-versions/{version}/activate

Select an existing version as the action's active decision contract.

Credential: Admin server credential.

Inputs: Action record id and integer version number. Review the whole draft snapshot before activating.

Effect and retry behavior: Activates the selected version and retires the prior active version. Pending/retried authorization can fail as stale rather than silently changing meaning.

Parameters

Name / location	Required	Meaning, type and constraints
action_id (path)	Yes	Protected-action record identifier returned by action creation. Supply this record ID in the path; use the dotted action key in authorization requests. string
version (path)	Yes	Policy version number belonging to the action. Supply the numeric version in this path. integer

Responses

HTTP status	Meaning	Body
200	Selected policy version in active state.	application/json: PolicyVersion
422	The canonical handler returns a Problem for invalid input; correct the request before retrying.	application/problem+json: Problem

Response handling: Unavailable or incompatible versions and lifecycle conflicts are refused; read the active version after an uncertain response.

Example

HTTP

```
POST /api/v1/actions/act_example/policy-versions/2/activate
```

Response excerpt (selected fields) - HTTP 200:

```
{
  "id": "pol_example",
  "action": "act_example",
  "version": 2,
  "status": "active"
}
```

API REFERENCE

Authorization and outcome API

Authorization evaluates one request under an active policy. The request binds the authenticated subject, human actor, action, resource, context, audience, environment, nonce, and idempotency identity. Your backend supplies and freezes these values after checking application permissions.

Interpret the response

A successful response returns `allow`, `challenge`, `hold`, or `deny`. Inspect mode as well as decision. An enforceable `allow` requires proof verification and online consumption before business execution. Use `test` and `shadow` responses for policy inspection and evaluation.

The immediate authorization response carries a compact proof string when applicable. The trace retrieval response carries a nested proof object with `compact`. A challenge or approved review can create a child authorization; verify its association with the original server-held pending request before using it.

Idempotency, review, and outcomes

Reuse one idempotency key only for retries of the same request material. A changed destination, subject, audience, or context is a new request. Preserve the nonce and request binding for a retry; resolve a binding conflict before submitting a new operation. A stale stored authorization can require fresh authorization after lifecycle or policy changes.

A review records an attributed approve or reject resolution. Your backend executes the action after the resulting authorization passes verification. Fetch and verify any resulting child authorization through the same binding and consumption checks.

Report the actual business outcome after execution or refusal. One outcome is stored per authorization; identical repeats are idempotent and contradictory updates are refused. Your application supplies outcome labels and maintains the evidence supporting them. Use your own retryable outbox for outcome reporting so a failed report never repeats the business mutation.

The [Integration guide](#) explains the complete pending-request and crash boundary. The operation contracts below describe authorization, trace retrieval, review, and outcome reporting.

GET /api/v1/authorizations/{authorization_id}

Retrieve the stored authorization identified by the request path. To resolve a challenge or review child, fetch the resulting child authorization ID explicitly.

Credential: Server credential.

Inputs: Backend-selected authorization id. Treat an id delivered by the browser as an untrusted candidate until bound to the original pending request.

Effect and retry behavior: Read-only. A challenge or review child carries parent; require it to equal the original authorization. The proof field is a nested object, unlike the immediate response string.

Parameters

Name / location	Required	Meaning, type and constraints
authorization_id (path)	Yes	Authorization identifier returned by a prior request or trusted challenge receipt. string

Responses

HTTP status	Meaning	Body
200	Full trace including original binding, parent relation and nested challenge/proof/review/outcome state.	application/json: AuthorizationTrace
422	The canonical handler returns a Problem for invalid input; correct the request before retrying.	application/problem+json: Problem

Response handling: 404 for an unavailable trace. Wrong parent, subject, audience, mode, action or independent action hash requires refusal even when lookup succeeded.

Example

HTTP

```
GET /api/v1/authorizations/az_child_example
```

Response excerpt (selected fields) - HTTP 200:

```
{
  "id": "az_child_example",
  "parent": "az_example",
  "subject": "sub_example",
  "action": "payout.destination.change",
  "decision": "hold",
  "mode": "shadow",
  "proof": null
}
```

POST /api/v1/authorizations/{authorization_id}/outcome

Record the outcome actually observed by the customer application after a decision.

Credential: Server credential.

Inputs: Choose the declared outcome label. Optional note is retained as a digest; use a minimal, non-confidential reference.

Effect and retry behavior: Stores one outcome per authorization. Identical repeats are idempotent; contradictory outcome updates conflict. Use an application outbox so reporting never repeats execution.

Parameters

Name / location	Required	Meaning, type and constraints
authorization_id (path)	Yes	Authorization identifier returned by a prior request or trusted challenge receipt. string

Request body

application/json; required. [OutcomeReport](#).

Responses

HTTP status	Meaning	Body
200	Recorded authorization id, outcome label and report timestamp.	application/json: OutcomeReceipt
422	The canonical handler returns a Problem for invalid input; correct the request before retrying.	application/problem+json: Problem

Response handling: 409 when a different outcome was already recorded; 404 for an unavailable authorization. Report executed after proof consumption and confirmed application execution.

Example

HTTP

```
POST /api/v1/authorizations/az_example/outcome
```

JSON request

```
{
  "outcome": "not_executed"
}
```

Response - HTTP 200:

```
{
  "authorization": "az_example",
  "outcome": "not_executed",
  "reported_at": "2026-10-01T12:00:00Z"
}
```

POST /api/v1/authorizations/{authorization_id}/review

Record an attributed administrator approve/reject resolution for manual review.

Credential: Admin server credential.

Inputs: Use the held authorization id. resolution is approve or reject; note is optional and retained as a digest.

Effect and retry behavior: Reject records a rejection outcome. Approve reevaluates a child request. Fetch the child decision and verify its proof before executing an eligible business operation.

Parameters

Name / location	Required	Meaning, type and constraints
authorization_id (path)	Yes	Authorization identifier returned by a prior request or trusted challenge receipt. string

Request body

application/json; required. [ManualReviewRequest](#).

Responses

HTTP status	Meaning	Body
200	Reject returns an outcome receipt; approve returns a fresh child decision that may still refuse execution.	application/json: ReviewApprovedReceipt or ReviewRejectedReceipt
422	The canonical handler returns a Problem for invalid input; correct the request before retrying.	application/problem+json: Problem

Response handling: Ineligible decision state, duplicate/conflicting resolution and lifecycle changes are refused. Require parent and original binding checks on any approved child.

Example

HTTP

```
POST /api/v1/authorizations/az_example/review
```

JSON request

```
{
  "resolution": "reject"
}
```

Response - HTTP 200:

```
{
  "authorization": "az_example",
  "resolution": "reject",
  "outcome": {
    "authorization": "az_example",
    "outcome": "manually_rejected",
    "reported_at": "2026-10-01T12:00:00Z"
  }
}
```

POST /api/v1/authorize

Evaluate one immutable request against the action's active policy and return allow, challenge, hold or deny.

Credential: Server credential.

Inputs: Derive subject, actor, resource and context on the backend. Persist requested_at (Unix seconds), a fresh nonce, idempotency_key and independently computed action hash before calling.

Effect and retry behavior: Stores a decision trace and may issue a challenge or enforce-mode proof and record qualifying usage. Identical retries reuse the request identity; changed material must be a new request.

Request body

application/json; required. [AuthorizeRequest](#).

Responses

HTTP status	Meaning	Body
200	Decision, mode, independent-binding digest, policy version, reasons and Assurance Profile; challenge or proof appears only when applicable.	application/json: AuthorizeResponse
422	The canonical handler returns a Problem for invalid input; correct the request before retrying.	application/problem+json: Problem

Response handling: Unregistered/inactive actions, invalid context, disallowed caller/audience, stale request, nonce reuse and conflicting idempotency are refused. An HTTP 200 deny/hold/challenge still requires non-execution.

Example

HTTP

```
POST /api/v1/authorize
```

JSON request

```
{
  "subject": "sub_example",
  "actor": {
    "id": "sub_example",
    "type": "human"
  },
  "action": "payout.destination.change",
  "resource": {
    "type": "merchant",
    "id": "merchant_42"
  },
  "context": {
    "destination_fingerprint": "bank-destination-fingerprint-7"
  },
  "audience": "merchant-api",
  "requested_at": 1790856000,
  "nonce": "2de95a66-712e-4a60-9ccf-3c9b53e442fa",
  "idempotency_key": "payout-change-42-request-7",
  "session": "ses_example"
}
```

Response - HTTP 200:

```
{
  "id": "az_example",
  "decision": "challenge",
  "mode": "shadow",
  "action": "payout.destination.change",
  "environment": "env_example",
  "policy_version": 1,
  "challenge": {
    "id": "ch_example",
    "methods": [
      "passkey"
    ],
    "expires_at": "2026-10-01T12:05:00Z"
  },
  "proof": null,
  "action_hash": "15fffd65a68fe273d7982c94152aa6e974e4f4b182e1733cc96fed169cfec22ea",
  "reason_codes": [
    "decision.insufficient_evidence"
  ],
  "human_readable": {
    "status": "challenge",
    "detail": "Additional action-bound confirmation is required."
  },
  "assurance": {
    "id": "asr_example",
    "subject": "sub_example",
    "session": "ses_example",
    "assessed_at": "2026-10-01T12:00:00Z",
    "claims": [],
    "coverage": "none",
    "evidence_windows": 0,
    "active_seconds": 0.0,
    "modalities_present": [],
    "freshness_ms": null,
  }
}
```

```
    "limitations": [  
      "insufficient_evidence"  
    ],  
    "model_version": "example-model-version",  
    "rule_version": "example-rule-version",  
    "feature_schema_version": "example-feature-version",  
    "insufficient_evidence": true  
  },  
  "proof_id": null,  
  "trace_id": "az_example",  
  "decided_at": "2026-10-01T12:00:00Z",  
  "latency_ms": 12  
}
```

API REFERENCE

Challenge API

A challenge binds an explicit approval step to the immutable original request. The supported canonical methods are action-bound passkey confirmation and a clearly labeled sandbox-only simulation when allowed by policy. Use the confirmation for this specific action in addition to the account's sign-in controls.

Browser workflow

Use the scoped browser token and SDK to retrieve challenge details, display them, obtain explicit confirmation, and complete the supported ceremony. Passkey registration is a separate process using a one-time grant issued by the authenticated backend.

Registration options are a WebAuthn creation-options object with `graypass_registration_id` added. Preserve that value as `registration_id` for registration completion. Action confirmation uses the separate challenge passkey-options operation and an authentication assertion. Keep registration, sign-in, and action-confirmation assertions associated with their respective ceremonies.

For browser callers, completion returns a receipt identifying the resulting authorization. Execution belongs to the backend verification flow. Send that ID to your backend; resolve it through the authenticated API, associate it with the original pending request, and verify its proof against the original locally computed action hash.

Response handling and lifecycle

Respect the challenge's expiry and status. Cancellation, expiry, duplicate or mismatched completion, denied confirmation, and unsupported methods preserve non-execution. If the business request changes, request a new authorization rather than reusing an old challenge.

Completing a challenge reevaluates the original request, including any active detector hold. Inspect the resulting child decision. Use the sandbox simulation to test lifecycle and binding, and the passkey ceremony for action-bound user confirmation.

See [Decisions and challenges](#) for UI and review guidance. The operation contracts below describe challenge retrieval, cancellation, passkey options, and completion.

GET /api/v1/challenges/{challenge_id}

Retrieve a challenge's current state, permitted methods, expiry and action display for explicit confirmation.

Credential: Scoped browser token or server credential.

Inputs: Use the challenge id from the original authorization; browser scope must match its subject and origin.

Effect and retry behavior: Read-only. Show the requesting application, action and consequence together with the exact business details saved by your backend. Use those details to display the destination associated with the returned digest.

Parameters

Name / location	Required	Meaning, type and constraints
<code>challenge_id</code> (path)	Yes	Challenge identifier tied to the original pending authorization. string

Responses

HTTP status	Meaning	Body
200	Current challenge state, available methods, expiry and browser-safe action display.	application/json: ChallengePublicView
422	The canonical handler returns a Problem for invalid input; correct the request before retrying.	application/problem+json: Problem

Response handling: 404 or scope refusal for an unavailable challenge. Start a fresh authorization when the intended request requires a new approval after expiry, cancellation or completion.

Example

HTTP

```
GET /api/v1/challenges/ch_example
```

Response excerpt (selected fields) - HTTP 200:

```
{
  "id": "ch_example",
  "status": "pending",
  "methods": [
    "passkey"
  ],
  "expires_at": "2026-10-01T12:05:00Z",
  "display": {
    "action_key": "payout.destination.change",
    "action": "Change payout destination",
    "consequence": "Future payouts may be sent to a different destination.",
    "audience": "merchant-api"
  }
}
```

POST /api/v1/challenges/{challenge_id}/cancel

Refuse the offered action confirmation and stop that pending challenge from completing.

Credential: Scoped browser token or server credential.

Inputs: Challenge id only; no body.

Effect and retry behavior: Marks the challenge cancelled. It never grants a proof or executes the underlying action.

Parameters

Name / location	Required	Meaning, type and constraints
challenge_id (path)	Yes	Challenge identifier tied to the original pending authorization. string

Responses

HTTP status	Meaning	Body
200	Challenge id and cancelled status.	application/json: ChallengeCancelled
422	The canonical handler returns a Problem for invalid input; correct the request before retrying.	application/problem+json: Problem

Response handling: Unavailable or invalid challenge lifecycle states are refused; keep the associated business request unexecuted.

Example

HTTP

```
POST /api/v1/challenges/ch_example/cancel
```

Response - HTTP 200:

```
{
  "id": "ch_example",
  "status": "cancelled"
}
```

POST /api/v1/challenges/{challenge_id}/complete

Verify an offered challenge method and reevaluate the immutable original request into a child authorization.

Credential: Scoped browser token or server credential.

Inputs: For passkey, supply the browser's assertion for this challenge. sandbox_confirm is simulation only and must be permitted in a sandbox policy.

Effect and retry behavior: Consumes the pending challenge. Browser callers receive a child-authorization receipt; server callers receive the decision. The backend must retrieve/bind/verify/consume before any mutation.

Parameters

Name / location	Required	Meaning, type and constraints
challenge_id (path)	Yes	Challenge identifier tied to the original pending authorization. string

Request body

application/json; required. [ChallengeCompleteRequest](#).

Responses

HTTP status	Meaning	Body
200	Scoped browser callers receive ChallengeCompletionReceipt; server callers receive AuthorizeResponse.	application/json: AuthorizeResponse or ChallengeCompletionReceipt
422	The canonical handler returns a Problem for invalid input; correct the request before retrying.	application/problem+json: Problem

Response handling: Expired, repeated, mismatched or unoffered methods and invalid assertions fail. Successful completion can still yield hold or deny if policy or attack signals require it.

Example

HTTP

```
POST /api/v1/challenges/ch_example/complete
```

JSON request

```
{
  "method": "sandbox_confirm"
}
```

Response - HTTP 200:

```
{
  "challenge": "ch_example",
  "status": "completed",
  "authorization": "az_child_example",
  "method": "sandbox_confirm"
}
```

POST /api/v1/challenges/{challenge_id}/passkey/options

Obtain WebAuthn authentication options bound to this challenge's immutable request digest.

Credential: Scoped browser token or server credential.

Inputs: Challenge id and correct browser origin. At least one active passkey must be registered for the subject.

Effect and retry behavior: Prepares the ceremony and records its RP/origin binding. Complete the ceremony and resolve the resulting authorization to obtain the action decision.

Parameters

Name / location	Required	Meaning, type and constraints
challenge_id (path)	Yes	Challenge identifier tied to the original pending authorization. string

Responses

HTTP status	Meaning	Body
200	WebAuthn request options with action-bound challenge bytes, RP id, eligible credentials and required user verification.	application/json: object
422	The canonical handler returns a Problem for invalid input; correct the request before retrying.	application/problem+json: Problem

Response handling: 409 for a nonpending challenge or no eligible registered passkey; scope/origin mismatches are refused.

Example

HTTP

```
POST /api/v1/challenges/ch_example/passkey/options
```

Response excerpt (selected fields) - HTTP 200:

```
{
  "timeout": 60000,
  "userVerification": "required"
}
```

API REFERENCE

Proof API

The proof APIs expose current proof state and online verification with optional one-time consumption. Combine local signature verification with online checks for current revocation and consumption state.

Verify before execution

The relying-party backend must already know its expected issuer, tenant, environment, audience, action key, and locally computed action hash. Obtain those values from trusted configuration and the original pending business request. Retain those independent expectations throughout challenge resumption and verification.

Use the Node verifier for local checks, then call the online verification operation with the same action expectations and `consume: true`. The server performs a scoped state check and conditional consumption. `consume: false` checks status; `consume: true` performs the conditional single-use consumption required before execution.

Respect the transaction boundary

If consumption fails or its response is ambiguous, keep the operation unexecuted until your application reconciles its durable request state. For an already-consumed proof, inspect the existing attempt in your durable request ledger. GrayPass consumption and your business database are separate transactions.

Raw proofs are sensitive short-lived authorization artifacts. Keep them out of URLs, analytics, support messages, and general logs. Use trace or proof identifiers for diagnostics when sufficient.

Read [Proofs and verification](#) for key handling and failure reasons and [Integration](#) for application recovery. The operations below document proof state and online verification.

GET /api/v1/proofs/{jti}/status

Look up whether a proof is active, consumed, revoked, expired or unknown in this environment.

Credential: Server credential.

Inputs: Use a proof id obtained from a trusted trace or verified proof.

Effect and retry behavior: Read-only snapshot of proof state. Consume the proof through the verification operation to reserve its single use; state can change between lookup and consumption.

Parameters

Name / location	Required	Meaning, type and constraints
jti (path)	Yes	Proof identifier from a trusted authorization trace or verified proof. string

Responses

HTTP status	Meaning	Body
200	Proof id and active, consumed, revoked, expired or unknown state.	application/json: ProofStatus

HTTP status	Meaning	Body
422	The canonical handler returns a Problem for invalid input; correct the request before retrying.	application/problem+json: Problem

Response handling: Execute only after successful verification and consumption of an active proof within the expected tenant and environment. Unknown or inactive state keeps the operation pending.

Example

HTTP

```
GET /api/v1/proofs/proof_example/status
```

Response - HTTP 200:

```
{
  "jti": "proof_example",
  "status": "consumed",
  "expires_at": "2026-10-01T12:05:00Z",
  "revoked_at": null
}
```

POST /api/v1/verify

Verify signature/binding/current proof state and optionally consume the proof once for the pending operation.

Credential: Server credential.

Inputs: Supply audience, action and independent action hash from the original request. Perform offline subject/tenant/environment checks first. The example intentionally sends an invalid proof to show refusal.

Effect and retry behavior: consume defaults false; true conditionally consumes an active proof. Consumption and the customer business transaction are separate; ambiguous results require reconciliation, never blind replay.

Request body

application/json; required. [VerifyRequest](#).

Responses

HTTP status	Meaning	Body
200	valid flag with verified claims on success, or refusal reason on failure; both can use HTTP 200.	application/json: VerifyResponse
422	The canonical handler returns a Problem for invalid input; correct the request before retrying.	application/problem+json: Problem

Response handling: HTTP 200 can contain `valid:false` for malformed, expired, mismatched, revoked, unknown or consumed proof. Inspect `valid`; a successful transport response alone is insufficient.

Example

HTTP

```
POST /api/v1/verify
```

JSON request

```
{
  "proof": "not-a-jws",
  "audience": "merchant-api",
  "expected_action": "payout.destination.change",
  "expected_action_hash": "0000000000000000000000000000000000000000000000000000000000000000",
  "consume": true
}
```

Response - HTTP 200:

```
{
  "valid": false,
  "reason": "malformed",
  "claims": null
}
```

API REFERENCE

Evaluation API

Evaluations select shadow decisions over a defined observation window and compare integration metrics with recorded success criteria. Use them to measure policy behavior, evidence coverage, and application outcomes. Paid evaluation terms are handled separately through your commercial agreement.

Define and close a study

Create the evaluation with the intended action keys and success criteria before interpreting results. Supported criteria concern shadow-decision counts, abstention, and outcome coverage. Supply timezone-aware boundaries where used. Keep policy and application changes documented alongside the evaluation.

The report includes the selected decision distribution, coverage, insufficient-evidence reasons, challenge and outcome metrics, operator-change indicator counts, latency summaries, and criterion results. The operator-change count is returned as `experimental_operator_change_indicators` and counts decisions carrying `operator_change.detected`, rather than independently verified attacks.

The result is passed whenever all recorded criteria are met. Otherwise it is `in_progress` for an open study or `failed` for a closed study. An open study can pass before closure; close it to fix the observation cutoff.

`abstention_rate` is the fraction of selected shadow decisions carrying `decision.insufficient_evidence`; it is not simply the fraction labeled hold or deny. `false_challenge_proxy` is the fraction of challenge decisions subsequently labeled `executed` or `manually_approved`. That proxy does not establish that those challenges were unnecessary. Outcome coverage is the fraction of selected decisions with a timely recorded outcome.

Closing fixes the cutoff to the earlier of the planned end and closing time; repeated closing preserves it. Later decisions and outcomes are excluded. Save the report before relevant trace retention removes underlying data required for your review.

Interpretation boundary

Interpret results with the selected sample, outcome coverage, and individual criteria. Your review process establishes the basis for customer-reported labels. Identify synthetic fixtures separately from participant observations and retain exclusions that affect how you apply the result to your workflow.

See [Evaluate](#) for protocol design and [Commercial model](#) for paid evaluation terms. The operation contracts below describe creation, listing, reporting, and closure.

GET /api/v1/evaluations

Read up to 100 evaluation configurations in the authenticated environment.

Credential: Server credential.

Inputs: No body or pagination parameters.

Effect and retry behavior: Read-only; use each id to fetch metrics or close an open study.

Responses

HTTP status	Meaning	Body
200	Array of up to 100 stored evaluation configurations.	application/json: Array of Evaluation

Response handling: Results are scoped to the calling tenant and environment. An empty list describes that scope and the supplied query.

Example

HTTP

```
GET /api/v1/evaluations
```

Response - HTTP 200:

```
[
  {
    "id": "evl_example",
    "name": "Payout shadow study",
    "action_keys": [
      "payout.destination.change"
    ],
    "success_criteria": {
      "min_shadow_decisions": 50,
      "max_abstention_rate": 0.5,
      "min_outcome_coverage": 0.5
    },
    "status": "open",
    "starts_at": "2026-10-01T12:00:00Z",
    "ends_at": "2026-10-31T12:00:00Z",
    "created_at": "2026-10-01T12:00:00Z"
  }
]
```

POST /api/v1/evaluations

Register the action selection, observation window and thresholds for an operational shadow study.

Credential: Admin server credential.

Inputs: Use registered action_keys and optional success_criteria. Defaults are 50 decisions, maximum abstention 0.5 and minimum outcome coverage 0.5; end defaults 30 days after start.

Effect and retry behavior: Creates an open evaluation record with the selected actions and criteria. Paid evaluation terms are handled separately through your commercial agreement.

Request body

application/json; required. [EvaluationCreate](#).

Responses

HTTP status	Meaning	Body
201	Open evaluation with deduplicated action keys, recorded thresholds and effective observation window.	application/json: Evaluation

HTTP status	Meaning	Body
422	The canonical handler returns a Problem for invalid input; correct the request before retrying.	application/problem+json: Problem

Response handling: 422 for unknown action keys, invalid threshold ranges, timestamps without timezone or an end not later than start.

Example

HTTP

```
POST /api/v1/evaluations
```

JSON request

```
{
  "name": "Payout shadow study",
  "action_keys": [
    "payout.destination.change"
  ],
  "success_criteria": {
    "min_shadow_decisions": 50,
    "max_abstention_rate": 0.5,
    "min_outcome_coverage": 0.5
  }
}
```

Response - HTTP 201:

```
{
  "id": "evl_example",
  "name": "Payout shadow study",
  "action_keys": [
    "payout.destination.change"
  ],
  "success_criteria": {
    "min_shadow_decisions": 50,
    "max_abstention_rate": 0.5,
    "min_outcome_coverage": 0.5
  },
  "status": "open",
  "starts_at": "2026-10-01T12:00:00Z",
  "ends_at": "2026-10-31T12:00:00Z",
  "created_at": "2026-10-01T12:00:00Z"
}
```

POST /api/v1/evaluations/{evaluation_id}/close

Fix the study cutoff and return the resulting criteria report.

Credential: Admin server credential.

Inputs: Evaluation id and admin authority.

Effect and retry behavior: Closes at the earlier of the planned end and current time. Repeating close preserves the cutoff; later decisions and outcomes are excluded.

Parameters

Name / location	Required	Meaning, type and constraints
evaluation_id (path)	Yes	Evaluation identifier returned when the study was created. string

Responses

HTTP status	Meaning	Body
200	Closed evaluation report with fixed cutoff and final registered-criterion result.	application/json: EvaluationReport
422	The canonical handler returns a Problem for invalid input; correct the request before retrying.	application/problem+json: Problem

Response handling: 404 for an unavailable evaluation. Closing preserves the recorded criteria and their measured results.

Example

HTTP

```
POST /api/v1/evaluations/evl_example/close
```

Response excerpt (selected fields) - HTTP 200:

```
{
  "id": "evl_example",
  "status": "closed",
  "result": "failed",
  "ends_at": "2026-10-01T12:00:00Z"
}
```

GET /api/v1/evaluations/{evaluation_id}/report

Compute the study's selected shadow-decision metrics and compare them with registered success criteria.

Credential: Server credential.

Inputs: Evaluation id. Interpret the report with the study protocol and customer-label limitations.

Effect and retry behavior: Read-only report over retained traces. An open study can already have result passed when all criteria are met. Closing fixes the observation cutoff; trace retention can limit historical reconstruction.

Parameters

Name / location	Required	Meaning, type and constraints
evaluation_id (path)	Yes	Evaluation identifier returned when the study was created. string

Responses

HTTP status	Meaning	Body
200	Evaluation configuration, aggregated metrics, per-criterion checks and in_progress/passed/failed result.	application/json: EvaluationReport
422	The canonical handler returns a Problem for invalid input; correct the request before retrying.	application/problem+json: Problem

Response handling: 404 for an unavailable evaluation. Null ratios indicate insufficient denominator data; inspect the sample and individual criteria before interpreting the result.

Example

HTTP

```
GET /api/v1/evaluations/evl_example/report
```

Response excerpt (selected fields) - HTTP 200:

```
{
  "id": "evl_example",
  "status": "open",
  "result": "in_progress",
  "metrics": {
    "decisions": 0,
    "abstention_rate": null,
    "outcome_coverage": 0.0
  }
}
```

API REFERENCE

Activity API

Activity provides an operational view of authorization traces in the authenticated environment. Use its supported filters and bounded limits to locate decisions, reasons, policy versions, and outcome status. The Console Activity view is a companion for inspecting these records.

Use traces for diagnosis

Start with the authorization or trace ID stored in your own business request ledger. Compare the action, subject, audience, mode, decision, active policy version, reasons, and outcome with the application operation. Confirm actual execution from your business ledger and reconcile its result with the reported outcome.

Use trace retrieval for detailed challenge, review, and proof association. Resume through the original authenticated pending-request flow and apply complete proof binding and consumption checks.

Retention and data handling

Activity is subject to the environment's retention behavior and the endpoint's bounded response. Maintain your business transaction ledger alongside the retained Activity records. Preserve an appropriately minimized review record when a report must outlive trace retention.

Avoid copying subject-linked traces or raw proof material into broad analytics or support systems. Share only the identifiers and sanitized context needed for investigation. [Production operation](#) and [Privacy](#) describe the operational and data boundaries. The contract below specifies supported filters and response fields.

GET /api/v1/activity

Find recent authorization traces by action, decision or subject for operational diagnosis.

Credential: Server credential.

Inputs: Optional exact action_key, decision and subject_id filters. limit defaults 100 and is bounded to 1-500; no cursor is available.

Effect and retry behavior: Read-only, newest decision first. Use the trace for the policy decision and your application ledger for execution state. Reconcile an absent outcome with that ledger.

Parameters

Name / location	Required	Meaning, type and constraints
limit (query)	No	Maximum records requested; default 100. The service bounds it to 1-500. No cursor pagination is exposed. integer default: 100
action_key (query)	No	Optional exact dotted action-key filter. string or null
decision (query)	No	Optional exact decision filter, normally allow, challenge, hold or deny. string or null
subject_id (query)	No	Subject identifier resolved from your authenticated account mapping. string or null

Responses

HTTP status	Meaning	Body
200	Recent authorization traces matching the supplied exact filters, newest decision first.	application/json: Array of AuthorizationTrace
422	The canonical handler returns a Problem for invalid input; correct the request before retrying.	application/problem+json: Problem

Response handling: Wrong environment or overly narrow filters can yield no rows. Never resume an action merely because a list item has an allow decision.

Example

HTTP

```
GET /api/v1/activity?action_key=payout.destination.change&limit=25
```

Response excerpt (selected fields) - HTTP 200:

```
[
  {
    "id": "az_example",
    "subject": "sub_example",
    "action": "payout.destination.change",
    "decision": "challenge",
    "mode": "shadow",
    "outcome": "not_executed"
  }
]
```

API REFERENCE

Schemas

Request and response schemas with field meanings and constraints. Required fields must be present. Nullable fields explicitly permit null. Defaults apply when a field is omitted. Unknown request fields are rejected where `additionalProperties: false` is specified. Keep response parsers tolerant of additive fields within the same contract version.

ActionCreate

Register a protected action and its initial policy settings.

Field / type	Required	Meaning and constraints
key string	Yes	Stable dotted action key used by authorization, such as <code>payout.destination.change</code> . maxLength: 128 pattern: <code>^[a-z][a-z0-9_]*(\\. [a-z][a-z0-9_]*)*\$</code>
name string	Yes	Customer-visible display name for the action or evaluation. minLength: 1 maxLength: 120
consequence string	Yes	Plain-language consequence if the wrong person performs the protected action. minLength: 1 maxLength: 500
risk_class string	No	Customer-selected consequence category for the action: low, elevated, high or critical. enum: ["low", "elevated", "high", "critical"] default: "high"
resource_type string	No	Customer-defined category of the protected resource, such as merchant. default: "resource" maxLength: 64
context_schema object	No	JSON Schema object applied to authorization context. Include every business value that must be bound to approval. additionalProperties: true
allowed_audiences Array of string	Yes	Relying-party audience identifiers allowed to receive proofs for this action. minItems: 1
allowed_callers Array of string	No	Credential identifiers allowed to authorize this action. An empty list permits any eligible server credential in the same environment.
proof_ttl_seconds integer	No	Proof lifetime in seconds, measured from issuance; the accepted range is 30-900 seconds. default: 300 minimum: 30.0 maximum: 900.0

`additionalProperties: false`

ActionUpdate

Change action display metadata or evaluation mode; policy security settings are versioned separately.

Field / type	Required	Meaning and constraints
name string (maxLength: 120) or null	No	Customer-visible display name for the action or evaluation.
consequence string (maxLength: 500) or null	No	Plain-language consequence if the wrong person performs the protected action.
mode string (enum: ["test", "shadow", "enforce"]) or null	No	Action evaluation mode. Test and shadow return decisions without enforceable proofs. An eligible enforce-mode allow can carry a proof. Omit to keep the current mode.

additionalProperties: false

Actor

Actor bound to the protected business request.

Field / type	Required	Meaning and constraints
id string	Yes	Application actor identifier. Use the subject identifier for the supported human proof-verification path. minLength: 1 maxLength: 256
type string	No	Actor category; only human is supported in this API version. const: "human" default: "human"

additionalProperties: false

AssuranceClaim

A named evidence claim with categorical status, coverage, freshness, and assessment context.

Field / type	Required	Meaning and constraints
claim string	Yes	Claim name, such as operator_continuity or human_presence. Capability availability is separately reported by metadata.
status ClaimStatus	Yes	Status of this named claim. Use unsupported and insufficient_evidence to select the configured evidence-collection or approval path.
coverage string	Yes	Evidence-coverage category: none, sparse, partial or sufficient. Describes the amount and variety of accepted evidence. enum: ["none", "sparse", "partial", "sufficient"]
freshness_ms integer (minimum: 0.0) or null	No	Age of the relevant evidence in milliseconds; null when freshness cannot be established.

Field / type	Required	Meaning and constraints
evidence_refs Array of string	No	Opaque references to the evidence supporting this claim.
reason_codes Array of string	No	Stable reason identifiers explaining acceptance, limitations or a decision; resolve them in the reason-code catalog.
limitations Array of string	No	Applicable limitations that must accompany interpretation of the result.

additionalProperties: false

AssuranceProfile

Server assessment of available evidence for one subject and session. It contains no calibrated confidence score.

Field / type	Required	Meaning and constraints
id string	Yes	Identifier of this record. Treat it as opaque and retain it for subsequent operations.
subject string	Yes	Pseudonymous subject identifier in the authenticated tenant and environment.
session string or null	No	Observation or enrollment session identifier, as required by the operation.
assessed_at string	Yes	Assessment time as an ISO 8601 timestamp.
claims Array of AssuranceClaim	Yes	Individual assurance claims with their own status and limitations.
coverage string	Yes	Evidence-coverage category: none, sparse, partial or sufficient. Describes the amount and variety of accepted evidence. enum: ["none", "sparse", "partial", "sufficient"]
evidence_windows integer	No	Number of accepted, non-overlapping evidence windows accounted for in this session or profile. default: 0
active_seconds number	No	Duration of accepted active evidence in seconds. default: 0.0
modalities_present Array of string	No	Names of observed evidence modalities included in coverage accounting.
freshness_ms integer or null	No	Age of the relevant evidence in milliseconds; null when freshness cannot be established.
limitations Array of string	No	Applicable limitations that must accompany interpretation of the result.
model_version string	Yes	Version identifier for the model used in this assessment. Retain it when comparing assessment results.

Field / type	Required	Meaning and constraints
rule_version string	Yes	Version identifier for the assurance rules used in this assessment.
feature_schema_version string	Yes	Version identifier for the feature representation used by the session or assessment.
insufficient_evidence boolean	Yes	True when evidence is insufficient for the assessment's supported claims.

additionalProperties: false

AuthorizationTrace

Server-readable record of the original request binding, decision and subsequent challenge, proof, review and outcome state.

Field / type	Required	Meaning and constraints
id string	Yes	Identifier of this record. Treat it as opaque and retain it for subsequent operations.
action string	Yes	Registered action key, such as payout.destination.change.
subject string	Yes	Pseudonymous subject identifier in the authenticated tenant and environment.
actor string	Yes	Bound human actor identifier.
actor_type string	Yes	Bound actor category; only human is supported. const: "human"
resource_type string	Yes	Customer-defined category of the protected resource, such as merchant.
resource_digest string	Yes	One-way digest of the resource binding.
context_digest string	Yes	Digest of the context binding. Render your independently retained business details for user confirmation.
action_hash string	Yes	SHA-256 digest binding the canonical authorization material. Compare it with the hash computed from the original server-held request.
audience string	Yes	Expected relying-party audience. It must be permitted by the action and independently known to the verifier.
mode string	Yes	Action evaluation mode. Test and shadow return decisions without enforceable proofs. An eligible enforce-mode allow can carry a proof. enum: ["test", "shadow", "enforce"]
policy_version integer or null	No	Immutable policy version number evaluated for this decision.

Field / type	Required	Meaning and constraints
decision string	Yes	Policy result: allow, challenge, hold or deny. Inspect mode and proof requirements before performing any business mutation. enum: ["allow", "challenge", "hold", "deny"]
reason_codes Array of string	Yes	Stable reason identifiers explaining acceptance, limitations or a decision; resolve them in the reason-code catalog.
assurance AssuranceProfile	Yes	Assurance Profile used for this authorization decision.
challenge ChallengeTrace or null	No	Challenge associated with the request. Null means no challenge is included in this response.
proof_id string or null	No	Proof identifier (jti), or null when no proof was issued.
proof ProofTrace or null	No	Nested proof state. Its compact field carries the signed proof; AuthorizeResponse.proof carries the compact string directly.
parent string or null	No	Parent authorization identifier for a challenge or review child; null for an original request.
review_status string or null	No	Recorded manual-review state, or null when no review occurred.
review_authorization_id string or null	No	Child authorization produced by an approved review, or null.
reviewed_at string or null	No	Manual-review time as an ISO 8601 timestamp, or null.
reviewed_by string or null	No	Identifier of the credential attributed to the manual review, or null.
outcome string or null	No	Business outcome reported by the customer application, with supporting evidence maintained by the customer.
outcome_reported_at string or null	No	Recorded business-outcome time as an ISO 8601 timestamp, or null.
requested_at string	Yes	Recorded request time as an ISO 8601 timestamp in the trace, distinct from the caller's Unix-seconds input.
decided_at string	Yes	Decision time as an ISO 8601 timestamp.
latency_ms integer	Yes	Server-recorded decision latency in milliseconds.

AuthorizeRequest

Immutable business-request material supplied by the relying-party backend after its own authentication and permission checks.

Field / type	Required	Meaning and constraints
subject string	Yes	Pseudonymous subject identifier in the authenticated tenant and environment.
actor Actor or null	No	Optional explicit human actor. When omitted, the subject is the actor. Use that same subject identifier for the supported human proof-verification path.
action string	Yes	Registered action key, such as payout.destination.change. maxLength: 128
resource Resource	Yes	Resource type and identifier for the exact object being changed.
context object	No	Business values bound to the decision and validated against the active context schema. Use safe integers rather than floating-point values. additionalProperties: true
audience string	Yes	Expected relying-party audience. It must be permitted by the action and independently known to the verifier. minLength: 1 maxLength: 128
requested_at integer	Yes	Unix timestamp in seconds; the request must be no older than 300 seconds and no more than 60 seconds in the future. This timestamp is excluded from the action hash and idempotency comparison; a retry can refresh it without changing the original nonce or business material. minimum: 0.0
nonce string	Yes	Fresh unpredictable nonce for this request. Retain it with the pending operation and reuse it only for retries of that same request. minLength: 16 maxLength: 128
idempotency_key string	Yes	Stable application request identifier reused only for identical retries; changed material with the same key conflicts. minLength: 8 maxLength: 128
session string or null	No	Optional observation session for this subject. When omitted, the service selects the subject's latest applicable session.
mandate_ref string or null	No	Supported value: null. Omit the field or send null.

additionalProperties: false

AuthorizeResponse

Immediate policy result. Only an enforce-mode allow can include a compact Proof of Agency.

Field / type	Required	Meaning and constraints
id string	Yes	Identifier of this record. Treat it as opaque and retain it for subsequent operations.
decision string	Yes	Policy result: allow, challenge, hold or deny. Inspect mode and proof requirements before performing any business mutation. enum: ["allow", "challenge", "hold", "deny"]
mode string	Yes	Action evaluation mode. Test and shadow return decisions without enforceable proofs. An eligible enforce-mode allow can carry a proof. enum: ["test", "shadow", "enforce"]
action string	Yes	Registered action key, such as payout.destination.change.
action_hash string	Yes	SHA-256 digest binding the canonical authorization material. Compare it with the hash computed from the original server-held request.
environment string	Yes	Environment identifier that scopes this record; sandbox and live resources are separate.
policy_version integer	Yes	Immutable policy version number evaluated for this decision.
reason_codes Array of string	Yes	Stable reason identifiers explaining acceptance, limitations or a decision; resolve them in the reason-code catalog.
human_readable HumanReadable	Yes	Display status and detail explaining the result. Branch on the stable decision and reason-code fields.
assurance AssuranceProfile	Yes	Assurance Profile used for the decision.
challenge ChallengeDescriptor or null	No	Challenge associated with the request. Null means no challenge is included in this response.
proof string or null	No	Compact signed JWS, or null. Require enforce-mode allow, trusted binding checks and successful online consumption before execution.
proof_id string or null	No	Proof identifier (jti), or null when no proof was issued.
trace_id string	Yes	Identifier to retain for authorization trace lookup and operational diagnosis.
decided_at string	Yes	Decision time as an ISO 8601 timestamp.
latency_ms integer	Yes	Server-recorded decision latency in milliseconds.

ChallengeCancelled

Acknowledgment that the selected challenge was cancelled.

Field / type	Required	Meaning and constraints
id string	Yes	Identifier of this record. Treat it as opaque and retain it for subsequent operations.
status string	Yes	Lifecycle state of this record; use the operation-specific values listed in the schema. const: "cancelled"

ChallengeCompleteRequest

Complete an action-bound challenge using one of its permitted methods.

Field / type	Required	Meaning and constraints
method string	Yes	Chosen challenge method; use only a method offered by this challenge. enum: ["passkey", "sandbox_confirm"]
assertion object (additionalProperties: true) or null	No	WebAuthn authentication assertion produced for this challenge's options when method is passkey. Omit for sandbox_confirm.

additionalProperties: false

ChallengeCompletionReceipt

Browser-safe completion receipt. The backend must retrieve and validate the referenced child authorization.

Field / type	Required	Meaning and constraints
challenge string	Yes	Completed challenge identifier.
status string	Yes	Lifecycle state of this record; use the operation-specific values listed in the schema. const: "completed"
authorization string	Yes	Authorization identifier associated with this receipt. Resolve it on the backend and verify the bound proof before execution.
method string	Yes	Chosen challenge method; use only a method offered by this challenge. enum: ["passkey", "sandbox_confirm"]

ChallengeDescriptor

Challenge offered by an authorization decision, including methods and expiry.

Field / type	Required	Meaning and constraints
id string	Yes	Identifier of this record. Treat it as opaque and retain it for subsequent operations.

Field / type	Required	Meaning and constraints
methods Array of string	Yes	Permitted challenge methods. sandbox_confirm is a simulated ceremony restricted to the sandbox.
expires_at string	Yes	Expiry time as an ISO 8601 timestamp. Use the capability before this time; obtain a fresh one afterward.

ChallengeDisplay

Display information for explicit action confirmation. Digests complement the relying party's independently retained business details.

Field / type	Required	Meaning and constraints
action_key string	Yes	Stable registered key for the action being approved.
action string	Yes	Customer-visible protected-action name.
consequence string	Yes	Plain-language consequence if the wrong person performs the protected action.
resource_type string	Yes	Customer-defined category of the protected resource, such as merchant.
resource_digest string	Yes	One-way digest of the resource binding.
context_digest string	Yes	Digest of the context binding. Render your independently retained business details for user confirmation.
audience string	Yes	Expected relying-party audience. It must be permitted by the action and independently known to the verifier.
requesting_application string	Yes	Relying-party application identified in the challenge display.

ChallengePublicView

Browser-safe view of a scoped challenge and the action it asks the subject to approve.

Field / type	Required	Meaning and constraints
id string	Yes	Identifier of this record. Treat it as opaque and retain it for subsequent operations.
status string	Yes	Lifecycle state of this record; use the operation-specific values listed in the schema.
methods Array of string	Yes	Permitted challenge methods. sandbox_confirm is a simulated ceremony restricted to the sandbox.
expires_at string	Yes	Expiry time as an ISO 8601 timestamp. Use the capability before this time; obtain a fresh one afterward.

Field / type	Required	Meaning and constraints
display ChallengeDisplay	Yes	Action display metadata and digests bound to the original request.

ChallengeTrace

Challenge lifecycle state embedded in an authorization trace.

Field / type	Required	Meaning and constraints
id string	Yes	Identifier of this record. Treat it as opaque and retain it for subsequent operations.
status string	Yes	Lifecycle state of this record; use the operation-specific values listed in the schema.
method_used string or null	No	Method used to complete the challenge, or null before completion.
strength string or null	No	Recorded challenge assurance label. Distinguishes action-bound passkey confirmation from sandbox simulation.
expires_at string	Yes	Expiry time as an ISO 8601 timestamp. Use the capability before this time; obtain a fresh one afterward.
completed_at string or null	No	Completion time as an ISO 8601 timestamp, or null before completion.

ClaimStatus

Permitted statuses for a named assurance claim. Read the status with the claim definition, coverage, freshness, and assessment context.

string

enum: ["confirmed", "likely", "uncertain", "not_confirmed", "detected", "not_detected", "not_present", "insufficient_evidence", "unsupported"]

ClientTokenCreate

Mint a short-lived, subject-bound browser credential for a registered origin.

Field / type	Required	Meaning and constraints
subject string	Yes	Pseudonymous subject identifier in the authenticated tenant and environment.
origin string	Yes	Exact browser origin including scheme, hostname and non-default port. It must be registered for the tenant. minLength: 1 maxLength: 2048

Field / type	Required	Meaning and constraints
ttl_seconds integer	No	Browser-token lifetime in seconds; default 900, permitted range 60-3600. default: 900 minimum: 60.0 maximum: 3600.0

additionalProperties: false

ClientTokenCreated

Browser credential issuance result. Treat client_token as a secret capability.

Field / type	Required	Meaning and constraints
client_token string	Yes	Short-lived bearer token for the selected subject and origin. Keep it out of URLs, logs and persistent browser storage.
expires_at number	Yes	Token expiry as Unix time in seconds, which differs from the ISO timestamps used by most resource records.
ttl_seconds integer	Yes	Browser-token lifetime in seconds; default 900, permitted range 60-3600.
origin string	Yes	Exact browser origin including scheme, hostname and non-default port. It must be registered for the tenant.

CredentialIdentity

Non-secret identity and scope of the authenticated server credential, used to establish trusted verification expectations.

Field / type	Required	Meaning and constraints
credential string	Yes	Public identifier of the calling credential.
tenant string	Yes	Tenant identifier to bind proof verification to the intended customer.
environment string	Yes	Environment identifier that scopes this record; sandbox and live resources are separate.
environment_kind string	Yes	Environment category: sandbox or live. enum: ["sandbox", "live"]
scope string	Yes	Credential authority category. Canonical backend routes require server or admin; client scope is insufficient. enum: ["client", "server", "admin"]
mode string	Yes	Credential key mode, test or live; distinct from an action's test, shadow or enforce mode. enum: ["test", "live"]

Field / type	Required	Meaning and constraints
proof_issuer string	Yes	Expected proof issuer reported by the trusted configured service; never learn an issuer from an unverified proof.

Enrollment

Enrollment attempt and completion state for a subject's behavioral reference.

Field / type	Required	Meaning and constraints
id string	Yes	Identifier of this record. Treat it as opaque and retain it for subsequent operations.
subject string	Yes	Pseudonymous subject identifier in the authenticated tenant and environment.
status string	Yes	Lifecycle state of this record; use the operation-specific values listed in the schema. enum: ["pending", "completed", "failed"]
anchor string	Yes	Account anchor established by the customer identity process and recorded by GrayPass.
windows_accepted integer	Yes	Count of accepted evidence windows recorded for this enrollment.
created_at string	Yes	Creation time as an ISO 8601 timestamp.
completed_at string or null	No	Completion time as an ISO 8601 timestamp, or null before completion.
limitations Array of string	No	Applicable limitations that must accompany interpretation of the result.

EnrollmentComplete

Ask the server to complete a subject's enrollment from its ready enrollment session.

Field / type	Required	Meaning and constraints
session string	Yes	Enrollment-purpose session owned by the subject in the path.
force boolean	No	Whether to replace an existing template after accepted-window, duration and modality readiness checks pass. default: false

additionalProperties: false

Evaluation

Stored selection, thresholds and observation window for a shadow integration study.

Field / type	Required	Meaning and constraints
id string	Yes	Identifier of this record. Treat it as opaque and retain it for subsequent operations.
name string	Yes	Customer-visible display name for the action or evaluation.
action_keys Array of string	Yes	Registered action keys included in the evaluation; the service deduplicates this selection.
success_criteria EvaluationCriteria	Yes	Recorded thresholds for shadow-decision count, abstention rate and outcome coverage, evaluated against the selected traces.
status string	Yes	Lifecycle state of this record; use the operation-specific values listed in the schema. enum: ["open", "closed"]
starts_at string	Yes	Timezone-aware ISO 8601 study start. If omitted during creation, the service uses the current time.
ends_at string or null	No	Timezone-aware ISO 8601 study end, later than starts_at. Creation defaults to 30 days after the start; closing can shorten this cutoff.
created_at string	Yes	Creation time as an ISO 8601 timestamp.

EvaluationCreate

Create a shadow integration study for registered actions with explicit success criteria.

Field / type	Required	Meaning and constraints
name string	Yes	Customer-visible display name for the action or evaluation. maxLength: 120
action_keys Array of string	Yes	Registered action keys included in the evaluation; the service deduplicates this selection. minItems: 1 maxItems: 100
success_criteria EvaluationCriteria	No	Recorded thresholds for shadow-decision count, abstention rate and outcome coverage, evaluated against the selected traces.
starts_at string or null	No	Timezone-aware ISO 8601 study start. If omitted during creation, the service uses the current time.
ends_at string or null	No	Timezone-aware ISO 8601 study end, later than starts_at. Creation defaults to 30 days after the start; closing can shorten this cutoff.

additionalProperties: false

EvaluationCriteria

Thresholds for operational integration metrics used to evaluate the selected shadow decisions.

Field / type	Required	Meaning and constraints
min_shadow_decisions integer	No	Minimum number of selected shadow decisions; default 50. default: 50 minimum: 1.0 maximum: 1000000.0
max_abstention_rate number	No	Maximum permitted fraction of selected decisions carrying decision.insufficient_evidence, from 0 to 1; default 0.5. default: 0.5 minimum: 0.0 maximum: 1.0
min_outcome_coverage number	No	Minimum fraction of selected decisions with recorded customer outcomes, from 0 to 1; default 0.5. default: 0.5 minimum: 0.0 maximum: 1.0

additionalProperties: false

EvaluationMetrics

Aggregated trace and customer-outcome metrics within the study window. Missing denominators can produce null ratios.

Field / type	Required	Meaning and constraints
decisions integer	Yes	Number of selected shadow authorization decisions.
by_decision object	Yes	Counts by decision label. additionalProperties: integer
by_mode object	Yes	Counts by action evaluation mode in the selected records. additionalProperties: integer
abstention_rate number or null	No	Fraction of selected decisions carrying decision.insufficient_evidence, or null when the selection is empty.
experimental_operator_change_indicators integer	Yes	Number of selected decisions carrying the operator_change.detected indicator. Interpret with the sample and customer-reviewed outcomes.
challenges integer	Yes	Count of selected decisions that requested a challenge.
false_challenge_proxy number or null	No	Fraction of challenge decisions whose reported outcome is executed or manually_approved, or null with no challenges. Review individual outcomes to assess whether the challenge was appropriate.
coverage_distribution object	Yes	Counts by assurance coverage category. additionalProperties: integer
outcomes object	Yes	Counts by customer-reported outcome label. additionalProperties: integer

Field / type	Required	Meaning and constraints
outcome_coverage number	Yes	Fraction of selected decisions with an outcome, from 0 to 1.
decision_latency_ms object	Yes	Server decision latency summaries in milliseconds; entries may be null when no samples exist. additionalProperties: integer or null
note string	Yes	Interpretation limitations supplied with these operational metrics.

EvaluationReport

Evaluation configuration, current metrics and registered-criterion results.

Field / type	Required	Meaning and constraints
id string	Yes	Identifier of this record. Treat it as opaque and retain it for subsequent operations.
name string	Yes	Customer-visible display name for the action or evaluation.
action_keys Array of string	Yes	Registered action keys included in the evaluation; the service deduplicates this selection.
success_criteria EvaluationCriteria	Yes	Recorded thresholds for shadow-decision count, abstention rate and outcome coverage, evaluated against the selected traces.
status string	Yes	Lifecycle state of this record; use the operation-specific values listed in the schema. enum: ["open", "closed"]
starts_at string	Yes	Timezone-aware ISO 8601 study start. If omitted during creation, the service uses the current time.
ends_at string or null	No	Timezone-aware ISO 8601 study end, later than starts_at. Creation defaults to 30 days after the start; closing can shorten this cutoff.
created_at string	Yes	Creation time as an ISO 8601 timestamp.
metrics EvaluationMetrics	Yes	Operational metrics computed from selected retained traces.
checks object	Yes	Per-criterion pass/fail results against the recorded thresholds. additionalProperties: boolean
result string	Yes	passed when all recorded criteria are met; otherwise in_progress for an open study or failed for a closed study. An open study can pass before closure; close it to fix the cutoff. enum: ["in_progress", "passed", "failed"]

EvidenceAck

Receipt for one evidence window, including whether it contributed to accepted coverage.

Field / type	Required	Meaning and constraints
status string	Yes	accepted contributes to coverage. Interpret neutral and rejected through the associated reason codes. enum: ["accepted", "neutral", "rejected"]
reason_codes Array of string	No	Stable reason identifiers explaining acceptance, limitations or a decision; resolve them in the reason-code catalog.
coverage string	Yes	Evidence-coverage category: none, sparse, partial or sufficient. Describes the amount and variety of accepted evidence. enum: ["none", "sparse", "partial", "sufficient"]
evidence_windows integer	Yes	Number of accepted, non-overlapping evidence windows accounted for in this session or profile.
active_seconds number	Yes	Duration of accepted active evidence in seconds.
modalities_present Array of string	No	Names of observed evidence modalities included in coverage accounting.
enrollment_ready boolean	No	Whether the server's accepted coverage meets enrollment readiness. Client elapsed time alone is insufficient. default: false

FeatureFrameRequest

A feature window submitted by the supported SDK, which handles extraction, session signing and transport.

Field / type	Required	Meaning and constraints
frame_id string	Yes	Unique identifier for this frame within the session; repeated identifiers are refused by replay controls. minLength: 1 maxLength: 128
timestamp number	Yes	Frame timestamp. The browser SDK emits Unix milliseconds; native-compatible Unix seconds are also normalized. Use the SDK's clock convention consistently.
duration_ms number	Yes	Frame duration in milliseconds; finite values from 0 through 600000 are accepted by the schema. Accepted coverage applies additional server checks. minimum: 0.0 maximum: 600000.0
window_start number (minimum: 0.0) or null	No	Optional evidence-window start as Unix seconds. When omitted, bounds are derived from timestamp and duration_ms.
window_end number (minimum: 0.0) or null	No	Optional evidence-window end as Unix seconds. Bounds must describe a fresh, non-overlapping window for this session.
features object	Yes	SDK-produced modality feature object containing bounded derived features. Raw key content, images, video and audio stay outside the evidence submission. additionalProperties: true

Field / type	Required	Meaning and constraints
purpose string	No	Frame purpose: enroll for enrollment evidence; score for observation. It must match the session's purpose. default: "score" pattern: "^(score enroll)\$"
research_optin boolean	No	Separate per-frame JSON opt-in, default false. Retention requires this field plus current subject consent, environment participation and deployment eligibility. default: false

HumanReadable

Customer-display wording explaining a decision; stable reason codes carry the programmatic meaning.

Field / type	Required	Meaning and constraints
status string	Yes	Short display label for the decision; use reason codes for stable programmatic behavior.
detail string	Yes	Human-readable explanation. Use stable codes and typed decision fields for application branching.

ManualReviewRequest

Record an attributed resolution for a held authorization.

Field / type	Required	Meaning and constraints
resolution string	Yes	Attributed manual review result: approve or reject. Approval still requires a fresh verified child decision. enum: ["approve", "reject"]
note string (maxLength: 500) or null	No	Optional review note; retained as a digest rather than a free-text business record. Keep your review evidence in your own authorized system.

additionalProperties: false

OutcomeReceipt

Acknowledgment of the customer's recorded business-outcome label.

Field / type	Required	Meaning and constraints
authorization string	Yes	Authorization identifier associated with this receipt. Resolve it on the backend and verify the bound proof before execution.
outcome string	Yes	Business outcome reported by the customer application, with supporting evidence maintained by the customer.
reported_at string	Yes	Time the outcome was recorded, as an ISO 8601 timestamp.

OutcomeReport

Report what the customer application actually did after a decision; reporting never executes or authorizes the action.

Field / type	Required	Meaning and constraints
outcome string	Yes	Business outcome reported by the customer application, with supporting evidence maintained by the customer. enum: ["executed", "not_executed", "manually_approved", "manually_rejected", "confirmed_attack", "false_challenge", "customer_override", "unknown"]
note string (maxLength: 500) or null	No	Optional outcome note stored as a one-way digest. Keep a readable case narrative in your own application.

additionalProperties: false

PasskeyRegistered

Identifier and creation time of the registered subject passkey.

Field / type	Required	Meaning and constraints
id string	Yes	Identifier of this record. Treat it as opaque and retain it for subsequent operations.
created_at string	Yes	Creation time as an ISO 8601 timestamp.

PasskeyRegistrationGrant

Single-use backend grant permitting the intended subject to begin passkey registration.

Field / type	Required	Meaning and constraints
token string	Yes	Short-lived, one-time passkey-registration grant. Deliver only to the authenticated subject's intended browser and keep out of logs.
expires_at string	Yes	Expiry time as an ISO 8601 timestamp. Use the capability before this time; obtain a fresh one afterward.

PasskeyRegistrationOptionsRequest

Exchange a subject-bound one-time grant for WebAuthn registration options.

Field / type	Required	Meaning and constraints
registration_grant string	Yes	One-time server-issued grant scoped to this subject's passkey registration. minLength: 32 maxLength: 256

additionalProperties: false

PolicyRequirements

Requirements and refusal precedence applied by one immutable policy version.

Field / type	Required	Meaning and constraints
claims object	No	Minimum status by claim name. The default requests operator_continuity at uncertain. Choose a requirement supported by the named claim. additionalProperties: string
coverage_min string	No	Minimum accepted evidence coverage category; default partial. enum: ["none", "sparse", "partial", "sufficient"] default: "partial"
freshness_max_ms integer	No	Maximum acceptable evidence age in milliseconds; default 60000 (one minute). default: 60000 minimum: 1000.0 maximum: 3600000.0
insufficient_evidence string	No	Decision used for insufficient evidence: challenge, hold or deny. Default challenge. enum: ["challenge", "hold", "deny"] default: "challenge"
challenge_methods Array of string	No	Permitted compensating methods; default passkey. sandbox_confirm is sandbox-only simulation.
explicit_approval_required boolean	No	Whether action-bound explicit approval is required even if other evidence satisfies policy. default: false
challenge_compensates Array of string	No	Evidence shortfalls a completed challenge may compensate: operator_continuity, human_presence, coverage and freshness by default. Positive attack indicators remain subject to hold/deny rules.
max_challenges integer	No	Maximum challenges in one request chain before escalation to hold; default 2. default: 2 minimum: 1.0 maximum: 5.0
hold_on Array of string	No	Reason codes that force hold under this policy. Default empty list.
deny_on Array of string	No	Reason codes that force deny under this policy. Default empty list.

additionalProperties: false

PolicyVersion

Immutable decision contract and its draft, active or retired lifecycle state.

Field / type	Required	Meaning and constraints
id string	Yes	Identifier of this record. Treat it as opaque and retain it for subsequent operations.
action string	Yes	Identifier of the protected-action record owning this version.
version integer	Yes	Monotonically assigned policy version number for the action.
status string	Yes	Lifecycle state of this record; use the operation-specific values listed in the schema. enum: ["draft", "active", "retired"]
requirements PolicyRequirements	Yes	Assurance, freshness and challenge requirements evaluated by this policy version.
context_schema object	Yes	JSON Schema object applied to authorization context. Include every business value that must be bound to approval. additionalProperties: true
allowed_audiences Array of string	Yes	Relying-party audience identifiers allowed to receive proofs for this action.
allowed_callers Array of string	Yes	Credential identifiers allowed to authorize this action. An empty list permits any eligible server credential in the same environment.
proof_ttl_seconds integer	Yes	Proof lifetime in seconds, measured from issuance; the accepted range is 30-900 seconds.
content_hash string	Yes	Digest of the immutable policy content; retain it when comparing policy configurations.
created_at string	Yes	Creation time as an ISO 8601 timestamp.
activated_at string or null	No	Policy activation time as an ISO 8601 timestamp, or null for a draft.
retired_at string or null	No	Policy retirement time as an ISO 8601 timestamp, or null when not retired.

PolicyVersionCreate

Create a draft policy. Omitted optional security settings inherit from the active version; activation is a separate operation.

Field / type	Required	Meaning and constraints
requirements PolicyRequirements	Yes	Required complete requirement set for the new version; omitted fields inside it use the documented requirement defaults.
context_schema object (additionalProperties: true) or null	No	JSON Schema object applied to authorization context. Include every business value that must be bound to approval.

Field / type	Required	Meaning and constraints
allowed_audiences Array of string (minItems: 1) or null	No	Relying-party audience identifiers allowed to receive proofs for this action.
allowed_callers Array of string or null	No	Credential identifiers allowed to authorize this action. An empty list permits any eligible server credential in the same environment.
proof_ttl_seconds integer (minimum: 30.0 maximum: 900.0) or null	No	Proof lifetime in seconds, measured from issuance; the accepted range is 30-900 seconds.

additionalProperties: false

Problem

Canonical problem-details response. Inspect status and code; detail and title are explanatory text.

Field / type	Required	Meaning and constraints
type string	No	Problem type URI; about:blank for the standard problem envelope. default: "about:blank"
title string	Yes	Short human-readable error summary.
status integer	Yes	HTTP status code represented as an integer.
detail string or null	No	Optional explanatory detail; wording is not a stable programmatic contract. default: null
code string	Yes	Stable canonical error identifier used for integration behavior.
instance string or null	No	Optional identifier for this particular problem occurrence. default: null

ProofStatus

Current scoped server state for a proof identifier. Use proof verification and consumption before execution.

Field / type	Required	Meaning and constraints
jti string	Yes	Proof identifier from a verified proof or trusted authorization trace.
status string	Yes	active, consumed, revoked, expired or unknown. Only active is eligible for attempted consumption. enum: ["active", "consumed", "revoked", "expired", "unknown"]
expires_at string or null	No	Expiry time as an ISO 8601 timestamp. Use the capability before this time; obtain a fresh one afterward.

Field / type	Required	Meaning and constraints
revoked_at string or null	No	Revocation time as an ISO 8601 timestamp, or null when not revoked.

ProofTrace

Signed proof and current state associated with an authorization trace.

Field / type	Required	Meaning and constraints
id string	Yes	Identifier of this record. Treat it as opaque and retain it for subsequent operations.
compact string or null	No	Compact JWS when available; keep it out of logs and browser-controlled execution decisions.
status string	Yes	Lifecycle state of this record; use the operation-specific values listed in the schema.
expires_at string or null	No	Expiry time as an ISO 8601 timestamp. Use the capability before this time; obtain a fresh one afterward.

ProtectedAction

Protected-action metadata together with the currently selected policy settings and evaluation mode.

Field / type	Required	Meaning and constraints
id string	Yes	Identifier of this record. Treat it as opaque and retain it for subsequent operations.
environment string	Yes	Environment identifier that scopes this record; sandbox and live resources are separate.
key string	Yes	Stable dotted action key used by authorization, such as payout.destination.change.
name string	Yes	Customer-visible display name for the action or evaluation.
consequence string	Yes	Plain-language consequence if the wrong person performs the protected action.
risk_class string	Yes	Customer-selected consequence category for the action: low, elevated, high or critical. enum: ["low", "elevated", "high", "critical"]
resource_type string	Yes	Customer-defined category of the protected resource, such as merchant.
context_schema object	Yes	JSON Schema object applied to authorization context. Include every business value that must be bound to approval. additionalProperties: true

Field / type	Required	Meaning and constraints
mode string	Yes	Action evaluation mode. Test and shadow return decisions without enforceable proofs. An eligible enforce-mode allow can carry a proof. enum: ["test", "shadow", "enforce"]
allowed_audiences Array of string	Yes	Relying-party audience identifiers allowed to receive proofs for this action.
allowed_callers Array of string	Yes	Credential identifiers allowed to authorize this action. An empty list permits any eligible server credential in the same environment.
proof_ttl_seconds integer	Yes	Proof lifetime in seconds, measured from issuance; the accepted range is 30-900 seconds.
active_policy_version integer or null	No	Currently active policy version number, or null when no version is active.
created_at string	Yes	Creation time as an ISO 8601 timestamp.
updated_at string	Yes	Most recent update time as an ISO 8601 timestamp.

RecoveryBegin

Start re-enrollment for a revoked subject under its configured recovery policy.

Field / type	Required	Meaning and constraints
anchor string	Yes	Account anchor established by the customer identity process and recorded by GrayPass. enum: ["none", "customer_session", "idp", "passkey", "verified_identity"]
anchor_evidence_ref string (minLength: 8 maxLength: 256) or null	No	Reference to the customer's anchor evidence; required for idp, passkey and verified_identity anchors. The customer identity process establishes the underlying claim. GrayPass retains the digest without independently verifying that identity claim.

additionalProperties: false

ResearchConsentRequest

Record or revoke subject research consent separately from ordinary service use and subject creation.

Field / type	Required	Meaning and constraints
consented boolean	Yes	Strict JSON boolean: true records consent and false records withdrawal. The server fixes the receipt purpose and disclosure version. Deployment eligibility is configured separately.

additionalProperties: false

Resource

Exact business resource bound to an authorization.

Field / type	Required	Meaning and constraints
type string	Yes	Application resource category, such as merchant. minLength: 1 maxLength: 64
id string	Yes	Application resource identifier, derived from the authenticated pending business request. minLength: 1 maxLength: 256

additionalProperties: false

ReviewApprovedReceipt

Approved review receipt with a newly evaluated child decision. Inspect that decision and complete its required verification before execution.

Field / type	Required	Meaning and constraints
authorization string	Yes	Authorization identifier associated with this receipt. Resolve it on the backend and verify the bound proof before execution.
resolution string	Yes	Attributed manual review result: approve or reject. Approval still requires a fresh verified child decision. const: "approve"
decision AuthorizeResponse	Yes	Fresh child authorization response, requiring the usual parent, request binding and proof checks.

ReviewRejectedReceipt

Rejected manual-review receipt and its recorded rejection outcome.

Field / type	Required	Meaning and constraints
authorization string	Yes	Authorization identifier associated with this receipt. Resolve it on the backend and verify the bound proof before execution.
resolution string	Yes	Attributed manual review result: approve or reject. Approval still requires a fresh verified child decision. const: "reject"
outcome OutcomeReceipt	Yes	Outcome receipt for the manual rejection.

SessionCreate

Open an enrollment or observation session for the authenticated subject.

Field / type	Required	Meaning and constraints
subject string	Yes	Pseudonymous subject identifier in the authenticated tenant and environment.
purpose string	No	enroll collects enrollment evidence; observe assesses continuity. The default is observe. enum: ["observe", "enroll"] default: "observe"
device_context object (additionalProperties: true) or null	No	Optional customer-provided device context, without independent device integrity attestation. Apply your endpoint controls to establish device trust.

additionalProperties: false

SessionCreated

Session identifier, feature contract and short-lived evidence-signing material returned to its scoped caller.

Field / type	Required	Meaning and constraints
id string	Yes	Identifier of this record. Treat it as opaque and retain it for subsequent operations.
subject string	Yes	Pseudonymous subject identifier in the authenticated tenant and environment.
purpose string	Yes	Session purpose, enroll or observe.
feature_schema_version string	Yes	Version identifier for the feature representation used by the session or assessment.
frame_signing_key string or null	No	Session-specific signing secret for SDK evidence transport, when issued. Never log or expose it to another subject.
frame_signing_alg string or null	No	Algorithm identifier for the session's evidence-signing material.
research_enabled boolean	No	Environment participation and a current subject consent receipt are both present at session creation. Storage also requires deployment enablement, current consent and per-frame opt-in. default: false
created_at string	Yes	Creation time as an ISO 8601 timestamp.

SessionEnded

Receipt for an ended evidence session and its accounted duration and windows.

Field / type	Required	Meaning and constraints
session string	Yes	Observation or enrollment session identifier, as required by the operation.

Field / type	Required	Meaning and constraints
status string	Yes	Lifecycle state of this record; use the operation-specific values listed in the schema. const: "ended"
duration_seconds number	Yes	Elapsed session duration in seconds, distinct from accepted active evidence duration. minimum: 0.0
evidence_windows integer	Yes	Number of accepted, non-overlapping evidence windows accounted for in this session or profile. minimum: 0.0

StatusSummary

Browser-safe coverage and enrollment-readiness summary without a score or full server assessment.

Field / type	Required	Meaning and constraints
session string	Yes	Observation or enrollment session identifier, as required by the operation.
coverage string	Yes	Evidence-coverage category: none, sparse, partial or sufficient. Describes the amount and variety of accepted evidence. enum: ["none", "sparse", "partial", "sufficient"]
evidence_windows integer	Yes	Number of accepted, non-overlapping evidence windows accounted for in this session or profile.
active_seconds number	Yes	Duration of accepted active evidence in seconds.
modalities_present Array of string	No	Names of observed evidence modalities included in coverage accounting.
freshness_ms integer or null	Yes	Age of the relevant evidence in milliseconds; null when freshness cannot be established.
human_readable string	Yes	Human-readable coverage/readiness explanation for the user interface.
enrollment_ready boolean	No	Whether the server's accepted coverage meets enrollment readiness. Client elapsed time alone is insufficient. default: false
limitations Array of string	No	Applicable limitations that must accompany interpretation of the result.

Subject

Pseudonymous subject lifecycle record within one environment.

Field / type	Required	Meaning and constraints
id string	Yes	Identifier of this record. Treat it as opaque and retain it for subsequent operations.
environment string	Yes	Environment identifier that scopes this record; sandbox and live resources are separate.
enrollment_state string	Yes	Current enrollment lifecycle state. Revoked subjects need the configured recovery path; deleted subjects are no longer operationally retrievable. enum: ["not_enrolled", "enrolling", "enrolled", "revoked", "deleted"]
anchor string	Yes	Account anchor established by the customer identity process and recorded by GrayPass.
recovery_policy string	Yes	Recovery rule governing whether re-enrollment requires an anchor or a separate manual-review process.
adaptation_frozen boolean	Yes	Whether online template adaptation is frozen; true prevents silent template adaptation.
created_at string	Yes	Creation time as an ISO 8601 timestamp.
revoked_at string or null	No	Revocation time as an ISO 8601 timestamp, or null when not revoked.
limitations Array of string	No	Applicable limitations that must accompany interpretation of the result.

SubjectCreate

Create or retrieve the subject mapped to a stable customer reference and compatible security settings.

Field / type	Required	Meaning and constraints
external_ref string	Yes	Stable customer account reference, 1-256 characters. It is received by the API and retained as a keyed hash; avoid using direct personal information. minLength: 1 maxLength: 256
anchor string	No	Account anchor established by the customer identity process and recorded by GrayPass. enum: ["none", "customer_session", "idp", "passkey", "verified_identity"] default: "customer_session"
anchor_evidence_ref string (minLength: 8 maxLength: 256) or null	No	Reference to the customer's anchor evidence; required for idp, passkey and verified_identity anchors. The customer identity process establishes the underlying claim. GrayPass retains the digest without independently verifying that identity claim.

Field / type	Required	Meaning and constraints
recovery_policy string	No	Recovery rule governing whether re-enrollment requires an anchor or a separate manual-review process. enum: ["re_enroll_with_anchor", "re_enroll_any", "manual_review"] default: "re_enroll_with_anchor"
metadata object	No	Optional customer metadata object. This version retains only the label key; avoid sensitive information. additionalProperties: true

additionalProperties: false

VerifyRequest

Online verification against expected action binding, with optional atomic one-time consumption.

Field / type	Required	Meaning and constraints
proof string	Yes	Compact JWS obtained through your trusted authorization workflow. First perform the documented offline and subject/tenant/environment binding checks.
audience string	Yes	Expected relying-party audience. It must be permitted by the action and independently known to the verifier.
expected_action string	Yes	Action key independently retained with the pending business request.
expected_action_hash string	Yes	64-character SHA-256 hex action hash computed from the original server-held material, never learned from an unverified proof. minLength: 64 maxLength: 64
consume boolean	No	Default false performs a state check. Set true immediately before the protected operation to attempt one-time consumption; it is a separate transaction from your business database. default: false

additionalProperties: false

VerifyResponse

Verification result. HTTP 200 can still mean valid is false; never authorize based on transport status alone.

Field / type	Required	Meaning and constraints
valid boolean	Yes	True only when all online checks succeed and, if requested, one-time consumption succeeds.
reason string or null	No	Stable verification refusal reason when invalid, otherwise null.

Field / type	Required	Meaning and constraints
claims object (additionalProperties: true) or null	No	Verified proof claims when valid, otherwise null. Compare them with the application's independent expectations.

API REFERENCE

Reason codes

Stable identifiers returned on assurance claims, decisions, evidence acknowledgements, and errors. Adding a code is a contract change; removing one requires a new API version. Match on the code, never on the human-readable text.

Claim statuses

Each Assurance Profile claim carries one of these statuses. Unsupported statuses are reported so a policy can never depend on a claim GrayPass does not make.

Code	Meaning
<code>operator_continuity.insufficient_evidence</code>	This session lacks sufficient fresh, template-backed evidence for continuity assessment.
<code>operator_continuity.likely</code>	Current behavior is consistent with the enrolled operator's Brainprint.
<code>operator_continuity.not_confirmed</code>	Behavior diverged from the enrolled operator.
<code>operator_continuity.uncertain</code>	Behavior is only partly consistent with the enrolled operator.
<code>operator_change.detected</code>	The change detector flagged a shift from the session's own operator baseline.
<code>operator_change.insufficient_evidence</code>	There is not enough fresh evidence to compare current activity with the session's operator baseline.
<code>operator_change.not_detected</code>	No shift from the session baseline was detected.
<code>operator_change.uncertain</code>	Reserved for an inconclusive change assessment. Current assessments use <code>operator_change.insufficient_evidence</code> when a comparison cannot be made.
<code>automation.detected</code>	An indicator flagged timing or movement patterns consistent with automation or replay; this does not establish general bot detection.
<code>automation.insufficient_evidence</code>	The evidence does not support an automation conclusion. Absence of an indicator is not proof of human origin.
<code>automation.not_detected</code>	Reserved for an assessment with no detected automation indicator. Current assessments use <code>automation.insufficient_evidence</code> when no indicator fires; absence does not establish human origin.
<code>human_presence.unsupported</code>	Derived interaction activity does not establish human presence or liveness.
<code>explicit_action_approval.confirmed</code>	This exact request has approval from a completed bound challenge or an authorized manual review. Check the approval method and final decision.

Code	Meaning
<code>explicit_action_approval.not_present</code>	No action-bound approval exists for this request.
<code>device_integrity.unsupported</code>	Current assurance does not establish device integrity.
<code>identity_anchor_validity.unsupported</code>	GrayPass did not verify the identity anchor declared at enrollment.
<code>personhood.unsupported</code>	GrayPass does not assess personhood.
<code>agent_mandate_validity.unsupported</code>	This assessment provides no assurance of agent-mandate validity.
<code>approval_method.manual_review</code>	Approval used an attributed manual review; behavioral limitations remain in the profile.
<code>approval_method.passkey</code>	Approval used a phishing-resistant WebAuthn passkey with user verification.
<code>approval_method.sandbox_confirm</code>	Approval used a labeled sandbox-only confirmation, not a passkey.

Coverage, freshness, and evidence

Coverage and freshness describe the evidence behind a claim. Evidence codes explain why a submitted window was not counted.

Code	Meaning
<code>coverage.none</code>	No accepted evidence windows are available.
<code>coverage.partial</code>	Accepted activity provides partial coverage below the sufficient category. The policy's required coverage is evaluated separately.
<code>coverage.sparse</code>	Accepted activity is sparse in windows, active time, or observed modalities. The policy's required coverage is evaluated separately.
<code>coverage.sufficient</code>	Accepted activity meets the sufficient coverage category. This does not by itself satisfy the policy's other requirements.
<code>freshness.stale</code>	Fresh template-backed evidence is absent or too old for the applicable freshness requirement.
<code>evidence.rejected_quality</code>	The evidence window failed integrity or quality checks and was not counted.
<code>evidence.rejected_replay</code>	The evidence window repeated, overlapped, or predated already counted evidence and was not counted.
<code>evidence.rejected_unsigned</code>	The evidence window carried no valid frame signature and was not counted.

Limitations

Limitations travel with the profile and remain visible in the decision trace. A challenge never removes them.

Code	Meaning
<code>limitation.adaptation_frozen</code>	Automatic updates to the subject's Brainprint from new observations are disabled. Freezing does not delete an existing template or stop evidence collection.
<code>limitation.compromised_endpoint_not_detectable</code>	A fully compromised device can fabricate evidence.
<code>limitation.enrollment_unanchored</code>	Enrollment declared no anchor or only a customer-session anchor; GrayPass did not independently verify the original enrollee.
<code>limitation.experimental_assessment</code>	Behavioral categories summarize observed patterns, not verified identity or calibrated accuracy. Apply the stated coverage, freshness and policy requirements.
<code>limitation.legacy_enrollment</code>	This enrollment uses an earlier contract; re-enrollment may be required for current features.
<code>limitation.no_calibrated_confidence</code>	Interpret claim statuses with their coverage, freshness and limitations, rather than as identity probabilities.
<code>limitation.single_modality</code>	Only one behavioral modality contributed.

Decision reasons

Every decision carries at least one reason code explaining which policy branch produced it.

Code	Meaning
<code>decision.challenge_completed</code>	Explicit approval satisfied the evaluated policy. Inspect the approval method to distinguish a bound challenge from an authorized manual review.
<code>decision.deny_condition</code>	A deny condition in the policy matched.
<code>decision.hold_condition</code>	A hold condition in the policy matched.
<code>decision.insufficient_evidence</code>	Required coverage, freshness, or continuity evidence is missing or insufficient. Follow the returned decision for the next step.
<code>decision.manual_review_approved</code>	An authorized reviewer explicitly approved this exact held request; behavioral limitations remain unchanged.
<code>decision.manual_review_rejected</code>	Reserved for rejection of an exact held request by an authorized reviewer. Current rejections are recorded in review state and the manually_rejected outcome rather than emitting this code.
<code>decision.manual_review_superseded</code>	The review did not override a deny condition or attack indicator; no enforceable proof was issued.

Code	Meaning
<code>decision.policy_satisfied</code>	All policy requirements were met.
<code>decision.requirement_unmet</code>	A required claim status was not reached.
<code>decision.session_missing</code>	No active session for the subject.
<code>decision.shadow_mode</code>	The action is in shadow mode; no enforceable proof was issued.
<code>decision.subject_revoked</code>	The subject is revoked.
<code>decision.test_mode</code>	The action is in test mode inside a sandbox environment.

Errors

Errors are returned as RFC 9457 problem details with the code in the `code` field. See the troubleshooting guide for the usual causes.

Code	Meaning
<code>error.action_archived</code>	Reserved for an archived action. Current authorization requests for archived actions return <code>error.action_not_registered</code> .
<code>error.action_not_registered</code>	No active registered action with this key is available in the environment; the action may be missing or archived.
<code>error.audience_not_allowed</code>	The audience is not permitted for this action.
<code>error.browser_credential_not_allowed</code>	Browser credentials cannot perform this operation.
<code>error.caller_not_allowed</code>	The authenticated credential is not permitted for this action or request chain.
<code>error.challenge_expired</code>	The challenge has expired or is no longer pending for this operation. Retrieve its current state before continuing.
<code>error.challenge_mismatch</code>	The challenge or passkey ceremony could not be validated for the intended subject and request. Read the problem detail for the failed prerequisite or validation.
<code>error.commercial_access_unavailable</code>	Confirm commercial access for this tenant and environment with GrayPass, then request fresh authorization for the pending operation.
<code>error.companion_unavailable</code>	Optional companion evidence is unavailable. No comparison result was returned.
<code>error.context_invalid</code>	A request field, context, or operation prerequisite is invalid. Read the problem detail to identify the required correction.
<code>error.enforcement_not_approved</code>	The requested mode requires an eligible environment configuration. Retain the effective action mode and contact support with the environment and trace ID.

Code	Meaning
<code>error.environment_mismatch</code>	The credential's environment does not match the resource.
<code>error.idempotency_conflict</code>	The request conflicts with an existing operation or its current state, including reuse of an idempotency key with a different payload. Inspect the existing record before retrying.
<code>error.idempotency_stale</code>	A stored idempotent result is no longer safe after subject, action, or policy state changed.
<code>error.nonce_reused</code>	The nonce was already used in this environment.
<code>error.not_found</code>	The resource does not exist in this environment. Returned identically for a resource owned by another tenant.
<code>error.policy_conflict</code>	The current action or policy state conflicts with this request or continuation. Retrieve the current state before starting a fresh authorization.
<code>error.policy_inactive</code>	The action has no active policy version.
<code>error.policy_integrity</code>	Immutable policy content did not match its recorded digest.
<code>error.proof_unavailable</code>	Obtain a valid proof before proceeding with this enforce-mode request. Keep the operation pending while you resolve the proof response.
<code>error.rate_limited</code>	The caller reached the request rate limit for this operation.
<code>error.request_in_progress</code>	Another request with this idempotency key is still being decided.
<code>error.request_stale</code>	The request timestamp or evidence-window bounds are missing, invalid, or outside the accepted freshness window.
<code>error.subject_not_enrolled</code>	The operation requires an enrolled subject and enrollment has not completed.
<code>error.subject_not_found</code>	The subject does not exist in this environment.