Network Working Group
Internet-Draft
Intended status: Informational
Expires: 23 November 2026

D. Smullen
B. Scriber
CableLabs

22 May 2026

Privacy Preference Declaration Protocol Specification
draft-dsmullen-ppd-protocol-03

Abstract

This document specifies a participant-facing protocol for Privacy Preference Declarations (PPDs) in home networks. The protocol is between a home-side PPD service endpoint and a device-side actor, formally the PPD participant, which is a device or a service acting on behalf of a device. It defines baseline operations for endpoint metadata confirmation, participant registration, optional participant declaration, effective-policy retrieval, policy acknowledgment, renewal, and reassociation. This document complements the PPD architecture and taxonomy documents by defining the message and sequencing behavior needed for interoperable policy signaling. The household policy instances carried by this protocol express privacy preferences for signaling and comparison; they do not by themselves define an enforcement mechanism that guarantees participant behavior.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://drspangle.github.io/draft-dsmullen-ppd-protocol/draft-dsmullen-ppd-protocol.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-dsmullen-ppd-protocol/.

Source for this draft and an issue tracker can be found at https://github.com/drspangle/draft-dsmullen-ppd-protocol.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Copyright Notice

Table of Contents

1. Introduction

[I-D.draft-dsmullen-ppd-architecture] defines the architectural roles, trust boundaries, and lifecycle meaning for Privacy Preference Declarations (PPDs) in home-network environments. [I-D.draft-dsmullen-ppd-taxonomy] defines the shared semantic floor used to express privacy rules and participant declarations. This document specifies the participant-facing protocol behavior that sits between those two companion documents. The broader relationship between PPD and earlier work such as DNT, P3P, MUD, and privacy-vocabulary or policy-expression efforts is discussed in [I-D.draft-dsmullen-ppd-architecture]. This document does not restate that comparison except where needed to explain protocol behavior.

The protocol defined here is intentionally narrow. It is designed to ensure that a device-side actor can discover or be provisioned with candidate home-side PPD service endpoints, confirm the selected endpoint, register, optionally describe itself, retrieve the current effective household policy that applies to it, and provide a

protected receipt acknowledgment for that exact policy instance. The protocol also defines how the home-side service and the device-side actor keep association current over time, including renewal and reassociation behavior.

The policy instances retrieved through this protocol express household privacy preferences and comparison inputs. They do not by themselves create or prove a separate enforcement action against participant behavior. Deployment-specific enforcement, gating, or control responses are outside the baseline participant-facing protocol defined here, because this document standardizes only the interoperable signaling path between the household-side endpoint and the participant, while enforcement depends on deployment-specific control surfaces and capabilities beyond that shared protocol baseline.

In the formal architecture terminology reused here, the device-side actor is the PPD participant. That term can be easy to misread, so this document makes the intended boundary explicit: the protocol-side participant is a device or a service acting for a device, not the homeowner, household member, or operator who set or review household policy.

2. Conventions and Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This document reuses the terminology defined in [I-D.draft-dsmullen-ppd-architecture]. In particular, it relies on the meanings of PPD participant, PPD service endpoint, policy authority, effective policy, association, current association, stale association, and needs reassociation.

For clarity in this document, PPD participant always means a device or a service acting on behalf of a device. It does not refer to a homeowner, household member, or other human actor on the household side of the system.

3. Scope

This document specifies:

* the participant-facing transport and serialization baseline;

* metadata confirmation for discovered candidate service endpoints;

* the baseline operation set for participant registration, optional declaration, policy retrieval, policy acknowledgment, and renewal;

* message-object expectations for those operations;

* reassociation behavior when current association can no longer be confirmed; and

* protocol-visible error and security behavior.

This document does not specify:

* operator-only status, dashboard, or diagnostics surfaces;

* household policy authoring interfaces;

* internal service-to-authority protocols;

* automated enforcement behavior; or

* non-HTTP transport profiles.

4. Protocol Model

4.1. Roles

This protocol defines a participant-facing contract between:

* a home-side PPD service endpoint, which presents effective policy instances and records protected policy acknowledgments; and

* a device-side PPD participant, which is a device or backend service acting on behalf of a device.

A policy authority may exist behind the PPD service endpoint, but this protocol does not require participants to discover or address that authority directly. When the service endpoint and policy authority are distinct, the deployment MUST preserve the authenticity and integrity of the policy information presented through the participant-facing endpoint.

The baseline end-to-end story is therefore:

1. the device-side participant learns or is provisioned with a home-side PPD service endpoint;

2. it confirms the endpoint and the applicable trust profile;

3. it registers and may optionally submit declaration data;

4. the home-side service endpoint returns the current effective policy for that participant;

5. the device-side participant acknowledges receipt of that exact policy instance; and

6. both sides use freshness and lifecycle state to determine whether association remains current or must be renewed or replayed.

4.2. Transport and Serialization

The baseline participant-facing protocol uses:

* HTTP over IP;

* the path prefix /ppd/v1; and

* JSON request and response bodies using application/json.

This document treats JSON as the baseline interoperable encoding. More compact encodings MAY be defined by future deployment profiles where resource constraints justify them, but such profiles need to preserve the same message semantics.

4.3. Security Profiles

This protocol defines explicit participant-facing security profiles. The metadata security_profile value identifies which profile a participant-facing service endpoint expects.

The following profile identifiers are defined:

* direct-constrained: authenticated direct-device participation for devices that can meet the minimum authenticated direct-participant bar without full certificate lifecycle expectations;

* direct-certificate: authenticated direct-device participation for devices that can support stronger certificate-capable deployments; and

* backend-mediated: authenticated participation by a service acting on behalf of a device.

The baseline interoperable profile set for this document consists of direct-constrained and direct-certificate. backend-mediated is an extension profile.

This document does not define an unauthenticated direct-participation profile. Extremely constrained devices that cannot satisfy the minimum authenticated direct-participant bar are expected to participate indirectly through a trusted intermediary, or remain non-participating.

Authenticated participation, regardless of mechanism family, MUST provide:

* endpoint authentication sufficient for the participant to authenticate the selected PPD service endpoint;

* participant authentication sufficient to bind registration and acknowledgment state to the same participant identity;

* confidentiality and integrity protection for participant-facing exchanges;

* policy-instance integrity sufficient to identify the acknowledged policy instance unambiguously; and

* freshness protection sufficient to prevent replay of old acknowledgments as evidence of current association.

This document does not require one universal credential mechanism across all participant classes. It is specific about required security properties first, while leaving room for deployment profiles to realize those properties differently for constrained direct devices, certificate-capable direct devices, and backend-mediated extensions.

4.4. Candidate Discovery and Metadata Confirmation

Every participant MUST support a configured or provisioned participant-facing PPD service endpoint. That is the minimum interoperable discovery floor.

This protocol does not standardize one universal automatic discovery mechanism. Participants MAY additionally learn candidate PPD service endpoints through local naming, DHCP-delivered hints, multicast service discovery, default-gateway probing, Wi-Fi onboarding hints, or comparable local-network mechanisms. Such mechanisms are optional discovery profiles unless a deployment profile requires them. Discovery yields candidate endpoints only; it does not establish authority.

A participant that learns a candidate endpoint through any discovery method MUST confirm that the endpoint supports this protocol before deeper interaction. For that purpose, the baseline protocol defines:

* GET /ppd/v1/meta

The metadata response is expected to identify at least:

* the participant-facing service URI;

* the protocol version or profile identifier;

* the taxonomy release or releases understood by the service;

* whether participant declarations are supported;

* whether protected acknowledgments are supported; and

* the expected security mode or trust profile.

The metadata response MUST NOT expose household policy contents, participant inventory, or acknowledgment history before the normal participant-facing trust checks succeed. The participant-facing discovery target is the PPD service endpoint itself, not an internal repository or policy-authority endpoint.

5. Participant Lifecycle

5.1. Initial Association

The baseline participant lifecycle is:

1. obtain one or more candidate PPD service endpoints;

2. confirm a selected endpoint using GET /ppd/v1/meta;

3. authenticate the selected endpoint according to the deployment’s trust profile;

4. register participant identity and metadata;

5. optionally submit a participant declaration;

6. retrieve the current applicable effective policy instance; and

7. acknowledge receipt of that specific policy instance.

Association is established only when the current applicable effective policy instance has been delivered and acknowledged. Acknowledgment is a receipt signal; it is not a claim of compatibility or compliance.

5.2. Renewal and Stale Association

Current association is freshness-bound. A participant MUST renew association often enough that the PPD service endpoint does not treat the participant as stale. The participant-facing protocol therefore needs a way to communicate renewal expectations.

The baseline effective-policy response and acknowledgment response MUST convey one of the following:

* an absolute renewal deadline; or

* a bounded renewal interval from the time of response.

For baseline interoperability, the minimum renewal procedure is:

1. the participant retrieves the current applicable effective policy instance using GET /ppd/v1/policy/effective/{device_id};

2. if the returned policy_id and policy_hash still identify the same policy instance the participant currently treats as associated, the participant renews by sending a fresh Policy Acknowledgment Object for that instance; and

3. if the returned policy instance differs, or if the service indicates reassociation-required, the participant MUST treat the renewal attempt as escalated to reassociation.

If the applicable effective policy instance remains unchanged but the participant does not complete that retrieval-and-acknowledgment renewal procedure before the conveyed freshness limit, the participant enters stale association. The participant no longer has current association until it successfully completes the minimum renewal procedure or, when required by the service, reassociation.

5.3. Reassociation Triggers

A participant enters needs reassociation when current association can no longer be confirmed because:

* the applicable effective policy instance changed;
* participant state relevant to effective-policy derivation changed;
* enough state was lost that the previous association can no longer be trusted; or
* another invalidating event defined by the applicable deployment profile occurred.

When reassociation is required, the participant MUST retrieve and acknowledge the current applicable effective policy instance again before current association is restored.

5.4. Non-Participating Devices

This protocol does not require every device on a home network to participate in PPD. Devices that do not participate remain outside the active message exchange. Their presence may influence local management or enforcement decisions, but such decisions are out of scope for this protocol. Extremely constrained devices that cannot satisfy the minimum authenticated direct-participant bar MAY instead be represented indirectly by a trusted intermediary that participates on their behalf.

5.5. Comparison Outcome Categories

This protocol does not define a universal conflict-resolution procedure between participant-supplied descriptive material and household policy. That depends on household intent, participant capability, and deployment logic.

When a deployment compares participant-side descriptive or policy-related inputs against household policy and needs to expose the result at the protocol boundary in the baseline participant-facing protocol, it SHOULD return a Comparison Outcome Object on the declaration path and classify the result using one of the following coarse outcome categories:

* compatible: the compared inputs can coexist without further action;

* conditionally_satisfiable: the compared inputs can coexist if an allowed exception, alternate mode, or bounded refinement is applied;

* decision_required: household or operator choice is required before a compatible outcome can be determined;

* unsatisfiable: the compared inputs cannot be satisfied together under the currently known conditions; or

* indeterminate: the service cannot currently determine a reliable outcome.

This document defines the categories only. It does not define a universal resolution procedure.

6. Protocol Operations

6.1. Overview

The baseline participant-facing operation set is:

1. GET /ppd/v1/meta

2. POST /ppd/v1/device/register

3. POST /ppd/v1/device/declaration (optional)

4. GET /ppd/v1/policy/effective/{device_id}

5. POST /ppd/v1/device/ack

These operations form a narrow control path. They let a device-side participant confirm the home-side service, identify itself, optionally describe itself, retrieve the current effective household policy that applies to it, and acknowledge receipt of that specific policy instance. They do not define household policy authoring, repository-facing workflows, compliance attestation, or conflict-resolution procedure.

When the effective policy changes, when freshness expires, or when other invalidating events occur, the same narrow operation set is replayed as needed to restore current association.

A deployment MAY expose additional readback or manageability operations, but those are not required for baseline interoperability. This document also does not define internal repository-facing operations or operator-only status endpoints.

6.2. Metadata Confirmation

6.2.1. GET /ppd/v1/meta

Purpose:

* confirm that a candidate endpoint supports the expected PPD protocol profile;
* advertise baseline feature support; and
* communicate security expectations before registration or policy retrieval.

A successful response MUST be a Service Metadata Object.

6.3. Registration

6.3.1. POST /ppd/v1/device/register

Purpose:

* create or refresh the service endpoint’s stored registration for a participant; and
* bind the participant’s current protocol identity to the registration state.

The request body MUST be a Device Registration Object.

The request body SHOULD include, when available and appropriate for the deployment:

* manufacturer
* model
* firmware_version
* hostname

The following fields MAY be included when the deployment profile permits them:

* mac_address
* ip_address

A successful response MUST be a Registration Result Object. Registration success returns the canonical participant identity established or confirmed by the service. It MUST NOT repeat metadata-confirmation fields such as the participant-facing service URI, supported feature flags, or security profile.

6.4. Declaration

6.4.1. POST /ppd/v1/device/declaration

Purpose:

* provide optional participant-side declaration data that can inform effective policy derivation or later operator review.

Declarations are optional. A participant that does not submit a declaration can still establish association if it can retrieve and acknowledge the applicable policy instance.

The request body MUST be a Device Declaration Object. See Device Declaration Object and Declaration Statement Object below for the precise object shape.

A declaration carries one or more descriptive statements that use the taxonomy dimensions defined in [I-D.draft-dsmullen-ppd-taxonomy], such as data type, purpose, action, source, and handling context. In this model, handling context means the context in which a dataflow occurs or into which it is directed; it is not limited to transfer destinations. The taxonomy document defines the meaning and composition of those dimensions.

A successful declaration response without comparison detail SHOULD use 204 No Content. When the service chooses to expose the result of comparing the declaration against household policy or effective-policy constraints at this boundary, a successful response MUST be 200 OK with a Comparison Outcome Object. Services are not required to compute or return such an outcome synchronously. This document does not define a participant-controlled request flag for comparison outcomes, and this declaration path MUST NOT be treated as a baseline negotiation or homeowner-prompt channel.

6.5. Effective Policy Retrieval

6.5.1. GET /ppd/v1/policy/effective/{device_id}

Purpose:

* return the effective policy instance currently applicable to the participant;

* return enough policy-instance provenance information to identify what was acknowledged; and

* communicate the association-freshness limit for current association.

A successful response MUST be an Effective Policy Object. See Effective Policy Object and Policy Rule Object below for the precise object shape.

A successful response SHOULD include policy-instance provenance fields that let later inspection distinguish the household baseline from any more specific inputs, such as:

* base_policy_id

* applied_policy_id when a more specific policy layer was applied

* computed_at

These fields describe the provenance of the returned policy instance itself. They do not describe the provenance of data later collected, transformed, or derived by participant devices or services.

This operation returns the policy instance the participant is expected to acknowledge. It is not required to expose the internal policy-authority topology or the full derivation algorithm.

6.6. Policy Acknowledgment

6.6.1. POST /ppd/v1/device/ack

Purpose:

* record a protected acknowledgment that a participant received a specific policy instance.

The request body MUST be a Policy Acknowledgment Object.

The acknowledgment payload is a receipt signal only. It MUST NOT be interpreted as a claim that the participant can satisfy every policy rule. If deployments need richer participant-side compatibility or status reporting, that behavior MUST be defined separately from the baseline acknowledgment.

A successful acknowledgment response MUST be an Acknowledgment Result Object. It returns the resulting association state and the next freshness value to be used for maintaining current association.

An acknowledgment that refers to a non-current or mismatched policy instance MUST be rejected.

7. Message Objects

The following object definitions are normative for baseline interoperability. Unless otherwise stated:

* identifiers such as device_id, declaration_id, policy_id, and rule_id are opaque text strings;

* timestamp fields use RFC 3339 date-time strings [RFC3339];

* policy_hash uses the form algorithm:value, and baseline implementations MUST support sha256. For the baseline JSON protocol, the hash value is computed over the UTF-8 octets of the Effective Policy Object serialized using the JSON Canonicalization Scheme (JCS) [RFC8785], after omitting the policy_hash member itself; and

* renewal_interval is a positive integer count of seconds.

7.1. Compact Term Identifiers

Taxonomy-bearing fields use compact term identifiers. A compact term identifier is a text string whose meaning is determined by:

* a reserved core prefix defined by the protocol or taxonomy work; or

* an explicit extension-prefix declaration in a Taxonomy Context Object.

The term identifier itself is the primary semantic hook. Taxonomy release metadata remains secondary validation context. Deployments MAY use company-specific or other non-core taxonomies when their terms are declared through the applicable taxonomy context and remain reducible to the shared core semantic model defined by the companion taxonomy work. For roles such as data_type, purpose, source, handling_context, and processing_boundary, that reduction can rely on equivalence or broader/narrower placement as defined by the taxonomy. For the flat action family, the reduction needs to preserve exact action meaning. For structured jurisdiction, the reduction needs to preserve the declared scope and the family-specific country or subdivision comparison model defined by the taxonomy.

For baseline interoperability, a compact term identifier MUST use the form prefix:term. The prefix identifies either a reserved core vocabulary or an explicitly declared non-core vocabulary. The baseline core prefix is ppd. A receiver MUST NOT silently reinterpret an unresolved compact term as some other known term.

7.2. Taxonomy Context Object

The Taxonomy Context Object carries optional vocabulary-release context and any required non-core prefix declarations.

It MAY include:

* release: a text identifier for the taxonomy release or profile in view when the object was produced; and

* prefixes: an object mapping non-core compact prefixes to stable namespace identifiers.

Reserved core prefixes MUST NOT be remapped in prefixes. A Taxonomy Context Object is REQUIRED whenever non-core compact prefixes appear in the containing object.

7.3. Term Resolution Behavior

Before processing a taxonomy-bearing field, a receiver MUST be able to deterministically expand each compact term identifier into the corresponding stable namespace-based term identifier.

For the baseline protocol:

* the core prefix ppd MUST be interpreted according to the companion taxonomy work;

* any non-core prefix used in the containing object MUST appear in the applicable Taxonomy Context Object;

* reserved core prefixes MUST NOT be redeclared or remapped; and

* a sender MUST NOT emit a taxonomy-bearing object whose compact identifiers it cannot itself deterministically resolve.

If deterministic expansion fails because a compact identifier is malformed, a required non-core prefix declaration is missing, or a reserved core prefix is redeclared or remapped, the receiver MUST treat the object as semantically unprocessable.

If deterministic expansion succeeds but the resulting stable term identifier is not supported for the relevant operation or deployment profile, the receiver MUST also treat the object as semantically unprocessable.

When a PPD service endpoint returns a taxonomy-bearing object, it MUST ensure that the terms it emits are consistent with any attached Taxonomy Context Object.

7.4. Service Metadata Object

The service metadata object describes a candidate endpoint before deeper interaction. It contains:

* service_uri (required, URI string): canonical participant-facing service URI;

* protocol_version (required, text): protocol version or profile identifier for the participant-facing contract;

* declaration_supported (required, boolean): whether the service accepts Device Declaration Objects;

* ack_supported (required, boolean): whether the service accepts Policy Acknowledgment Objects;

* security_profile (required, text): deployment security profile identifier, currently one of direct-constrained, direct-certificate, or the extension value backend-mediated; and

* supported_taxonomy_releases (optional, array of text): taxonomy release identifiers understood by the service for validation and reproducibility. These release identifiers are secondary validation context, not the primary semantic hook for taxonomy-bearing terms.

7.5. Device Registration Object

The registration object identifies the participant and carries optional device metadata. The stable identifier is device_id. Other metadata fields are deployment-dependent and do not replace the stable participant identifier.

It contains:

* device_id (required, text): stable participant identifier for this device-side actor;

* manufacturer (optional, text): participant-reported vendor name;

* model (optional, text): participant-reported model name or number;

* firmware_version (optional, text): participant-reported software or firmware version;

* hostname (optional, text): participant-reported hostname when relevant to the deployment;

* mac_address (optional, text): participant-reported link-layer address when the deployment profile permits it; and

* ip_address (optional, text): participant-reported network address when the deployment profile permits it.

7.6. Registration Result Object

The registration result object confirms the canonical participant identity bound by registration.

It contains:

* device_id (required, text): canonical participant identifier established or confirmed by the service.

7.7. Device Declaration Object

The declaration object carries participant-supplied descriptive dataflow information. At minimum it contains device_id, declaration_id, and a non-empty statements array. Declaration statements use the shared taxonomy dimensions defined in [I-D.draft-dsmullen-ppd-taxonomy]. The taxonomy document defines the meaning of those fields, the qualifier families used with them, and the core semantic floor that keeps comparison computable across richer vocabularies; this protocol document defines only how such

statements are carried. This document uses operation for participant-facing protocol exchanges such as registration, retrieval, and acknowledgment. Terms such as handling, processing, and dataflow inside declaration statements and policy rules have the meanings defined by the companion taxonomy specification. The baseline declaration is intentionally minimal. Registration carries participant identity, declarations carry descriptive participant assertions, and Effective Policy and Acknowledgment Objects carry the lifecycle-critical policy binding and freshness semantics.

The declaration is descriptive only. It MUST NOT include normative policy verdicts such as allow or deny.

It contains:

* device_id (required, text): participant identifier to which the declaration applies;

* declaration_id (required, text): stable identifier for this declaration instance;

* taxonomy (optional, Taxonomy Context Object): release context and any required non-core prefix declarations;

* statements (required, non-empty array of Declaration Statement Objects): participant-supplied descriptive dataflow cases stating which taxonomy- defined combinations apply to this participant.

If a declaration uses any non-core compact prefix in its statements or constraints, the taxonomy object is REQUIRED.

7.8. Declaration Statement Object

A Declaration Statement Object is an atomic descriptive statement inside a Device Declaration Object. It mirrors the same core dimensions used by policy rules so that participant assertions can be compared at the same grain, but it MUST NOT include a normative effect.

It contains:

* statement_id (required, text): stable identifier for the statement within the declaration instance;

* data_type (required, compact term identifier): data category to which the statement applies;

* purpose (required, compact term identifier): why the dataflow occurs;

* action (required, compact term identifier): which privacy-relevant action the participant performs or may request;

* source (required, compact term identifier): immediate origin of the data in that dataflow;

* handling_context (required, compact term identifier): context in which the dataflow occurs or into which it is directed. For collection, this identifies the context into which collected data is brought. For use and inference, it identifies the context in which that dataflow occurs. For transfer, it identifies the recipient-side context into which data is transferred. The semantic meaning of this field is defined by [I-D.draft-dsmullen-ppd-taxonomy]; and

* constraints (optional, Constraints Object): structured qualifiers that refine the statement.

7.8.1. Example Device Declaration Object

```
{
  "device_id": "doorbell-7",
  "declaration_id": "doorbell-7-declaration-v1",
  "taxonomy": {
    "release": "ppd-core-2026-05"
  },
  "statements": [
    {
      "statement_id": "content-security-local-use",
      "data_type": "ppd:contentData",
      "purpose": "ppd:security",
      "action": "ppd:use",
      "source": "ppd:participantObserved",
      "handling_context": "ppd:householdContext",
      "constraints": {
        "processing_boundary": "ppd:onDeviceOnly"
      }
    },
    {
      "statement_id": "sensor-improvement-transfer",
      "data_type": "ppd:sensorData",
      "purpose": "ppd:analyticsAndImprovement",
      "action": "ppd:transfer",
      "source": "ppd:participantObserved",
      "handling_context": "ppd:vendorContext",
      "constraints": {
        "retention": "ppd:indefinite",
        "jurisdiction": {
          "scope": "transfer",
          "countrycode": ["us"]
        }
      }
    }
  ]
}
```

7.9. Comparison Outcome Object

The comparison outcome object carries an optional coarse result for declaration-to-policy comparison on the declaration path. It reports a coarse diagnostic result of comparing participant-side atomic declaration statements against household-side atomic policy rules. It does not change the meaning of the Effective Policy Object and it is not part of acknowledgment semantics. It MUST NOT be treated as a request for policy relaxation, an invitation to begin a participant-driven bargaining loop, or a trigger for baseline homeowner consent prompting.

It contains:

* declaration_id (required, text): declaration instance to which this comparison result applies;

* outcome (required, text): one of compatible, conditionally_satisfiable, decision_required, unsatisfiable, or indeterminate; and

* detail (optional, text): brief human-readable explanation suitable for diagnostics or operator review.

7.10. Effective Policy Object

The effective policy object represents the policy instance the participant must acknowledge. It contains the policy identifier, hash, rule set, and freshness information. It SHOULD also contain policy-instance provenance fields that make later recordkeeping and inspection meaningful.

It contains:

* policy_id (required, text): stable identifier for the policy instance to be acknowledged;

* policy_hash (required, text): stable content hash for the policy instance. This hash binds the full returned policy instance, including freshness and provenance fields, as serialized under the baseline canonicalization rule above;

* rules (required, array of Policy Rule Objects): normative rule set for this effective policy instance;

* renew_by (optional, RFC 3339 date-time string): absolute deadline by which current association must be renewed if this field is used;

* renewal_interval (optional, positive integer seconds): bounded interval after response generation within which current association must be renewed if this field is used;

* taxonomy (optional, Taxonomy Context Object): release context and any required non-core prefix declarations for rule terms;

* base_policy_id (optional, text): identifier for the household baseline policy used in this effective result;

* applied_policy_id (optional, text): identifier for a more specific applied policy layer when present; and

* computed_at (optional, RFC 3339 date-time string): time at which the effective policy instance was computed or materialized.

An Effective Policy Object MUST contain exactly one of renew_by or renewal_interval. These fields govern association freshness for the participant-facing lifecycle. They do not define abstract policy validity outside that lifecycle. If any rule uses a non-core compact prefix, the taxonomy object is REQUIRED. A complete example appears below under Example Effective Policy and Acknowledgment.

7.11. Policy Rule Object

A Policy Rule Object is an atomic normative statement inside an Effective Policy Object.

The baseline rule model uses singular core dimensions. When multiple cases must be expressed, they are represented as multiple rules rather than array-valued core dimensions inside one rule.

It contains:

* rule_id (required, text): stable identifier for the rule within the policy instance;

* data_type (required, compact term identifier): data category to which the rule applies;

* purpose (required, compact term identifier): why the dataflow occurs;

* action (required, compact term identifier): which privacy-relevant action the rule covers;

* source (required, compact term identifier): immediate origin of the data in that dataflow;

* handling_context (required, compact term identifier): context in which the dataflow occurs or into which it is directed. For collection, this identifies the context into which collected data is brought. For use and inference, it identifies the context in which that dataflow occurs. For transfer, it identifies the recipient-side context into which data is transferred. The semantic meaning of this field is defined by [I-D.draft-dsmullen-ppd-taxonomy];

* effect (required, text): normative rule effect, currently one of allow or deny; and

* constraints (optional, Constraints Object): structured qualifiers that refine the rule.

An Effective Policy Object SHOULD NOT contain two Policy Rule Objects with the same core dimensions but different effect values. Such contradictions should be resolved before the effective policy is returned.

7.12. Constraints Object

The Constraints Object preserves a structured extension point for dataflow qualifiers without requiring a large qualifier language in the baseline draft. It is shared by declaration statements and policy rules. The wire container is named constraints, but the companion taxonomy work defines its members as qualifiers on atomic dataflows. [I-D.draft-dsmullen-ppd-taxonomy] also defines the semantic meaning of retention, processing_boundary, and jurisdiction; this document defines only their participant-facing wire representation.

The initial standardized members are:

* retention (optional, compact term identifier): baseline retention qualifier for the scoped dataflow. The baseline compact form is term-valued; future specifications or deployment profiles MAY define more structured bounded-retention forms.

* processing_boundary (optional, compact term identifier): processing-placement qualifier for use or inference within the declared handling context.

* jurisdiction (optional, object): declarative jurisdiction qualifier for the scoped dataflow or storage context.

When present, jurisdiction contains:

* scope (required, text): one of collection, use, inference, transfer, or storage;

* countrycode (optional, array of text): one or more lowercase country codes using the countrycode format [RFC8006]; and

* subdivisioncode (optional, array of text): one or more lowercase country subdivision codes using the subdivisioncode format [RFC9388].

At least one of countrycode or subdivisioncode MUST be present. A sender MUST NOT include an empty countrycode or subdivisioncode array. A receiver MUST treat an unknown scope, a malformed jurisdiction object, or an invalid country or subdivision code as invalid input.

Future specifications or deployment profiles MAY define additional structured constraint members. A Constraints Object MUST NOT be treated as an unstructured free-form text field.

7.13. Policy Acknowledgment Object

The acknowledgment object binds a participant identifier to a specific policy instance and policy hash. Deployments that claim strong accountability properties MUST protect the acknowledgment against forgery, replay, and stale-policy confusion.

It contains:

* device_id (required, text): participant identifier acknowledging receipt;

* policy_id (required, text): policy instance identifier being acknowledged; and

* policy_hash (required, text): content hash of the acknowledged policy instance, computed according to the baseline policy_hash definition above.

This object is evidentiary only. It is a receipt for a specific policy instance and MUST NOT be interpreted as a claim of compatibility or compliance.

7.14. Acknowledgment Result Object

The acknowledgment result object confirms the lifecycle state after the service records a protected acknowledgment.

It contains:

* association_status (required, text): resulting association state, currently one of not_associated, associated, needs_reassociation, stale_association, or broken;

* renew_by (optional, RFC 3339 date-time string): absolute deadline by which current association must be renewed if this field is used; and

* renewal_interval (optional, positive integer seconds): bounded interval after response generation within which current association must be renewed if this field is used.

An Acknowledgment Result Object MUST contain exactly one of renew_by or renewal_interval.

7.14.1. Example Effective Policy and Acknowledgment

An Effective Policy Object example:

```
{
  "policy_id": "effective-doorbell-7-v3",
  "policy_hash": "sha256:8de72af3c4d6d8c9f0b0f6a4a13c8df0f716c9c0a1130d27c855a2dd8dd8e
8c7",
  "renewal_interval": 900,
  "taxonomy": {
    "release": "ppd-core-2026-05"
  },
  "base_policy_id": "home-default-v2",
  "applied_policy_id": "doorbell-exception-v1",
  "computed_at": "2026-05-13T18:00:00Z",
  "rules": [
    {
      "rule_id": "r1",
      "data_type": "ppd:contentData",
      "purpose": "ppd:security",
      "action": "ppd:use",
      "source": "ppd:participantObserved",
      "handling_context": "ppd:householdContext",
      "effect": "allow",
      "constraints": {
        "processing_boundary": "ppd:onDeviceOnly"
      }
    }
  ]
}
```

The matching Policy Acknowledgment Object example is:

```
{
  "device_id": "doorbell-7",
  "policy_id": "effective-doorbell-7-v3",
  "policy_hash": "sha256:8de72af3c4d6d8c9f0b0f6a4a13c8df0f716c9c0a1130d27c855a2dd8dd8e
8c7"
}
```

An Acknowledgment Result Object example is:

```
{
  "association_status": "associated",
  "renewal_interval": 900
}
```

7.15. Error Object

Error responses SHOULD use application/problem+json and a structured error object with at least:

* type: problem type identifier, including PPD-specific problem types when applicable;
* title: short problem summary;
* status: HTTP status code for this error; and
* detail: human-readable explanation when useful.

A deployment MAY include:

* instance: problem-instance identifier; and
* retryable: boolean hint about whether retry is appropriate.

Error responses MUST NOT leak more household or participant metadata than is necessary to explain the failure.

For PPD-specific problems, the type member SHOULD be one of the following relative references:

* invalid-request
* invalid-participant-binding
* reassociation-required
* stale-association
* policy-instance-mismatch
* unsupported-taxonomy-term
* term-resolution-failed
* policy-authority-unavailable

8. Error Handling

The baseline protocol uses conventional HTTP status codes. At minimum, participants need to handle:

* 200 OK for successful retrieval or update with a response body;
* 204 No Content for successful declaration acceptance when no diagnostic response body is returned;
* 400 Bad Request for invalid payloads or missing required fields;
* 401 Unauthorized for failed authentication;
* 403 Forbidden for authenticated participants that are not authorized for the requested operation;
* 404 Not Found for missing participant or policy state;
* 409 Conflict for lifecycle or policy-instance conflicts, such as acknowledgments that do not match the current policy instance;
* 422 Unprocessable Content for well-formed content that cannot be processed semantically, such as unsupported or unresolvable taxonomy terms;
* 503 Service Unavailable for transient service or policy-authority unavailability; and
* other 5xx errors for unexpected service failures.

The initial PPD-specific problem vocabulary is:

* invalid-request: malformed request payload, missing required fields, or invalid field shape;
* invalid-participant-binding: authenticated participant identity does not match the bound registration or requested participant identifier;
* reassociation-required: current association is no longer valid and the participant must replay the required lifecycle steps;
* stale-association: the acknowledged policy instance may still be current, but freshness expired and renewal is required;
* policy-instance-mismatch: the supplied policy_id or policy_hash does not identify the current policy instance the service expects;

* unsupported-taxonomy-term: the service recognizes the request shape and can resolve the supplied compact term identifier or identifiers, but does not support one or more resulting taxonomy terms for the relevant operation or deployment profile;

* term-resolution-failed: the service cannot deterministically resolve a supplied compact term identifier to usable protocol semantics, such as because the identifier is malformed, a required non-core prefix declaration is missing, or a reserved core prefix was redeclared or remapped; and

* policy-authority-unavailable: the participant-facing service cannot currently obtain or materialize the effective policy instance it needs to serve.

The following HTTP status mappings are RECOMMENDED:

* 400 Bad Request with invalid-request;

* 401 Unauthorized with invalid-participant-binding when authentication fails to establish the expected participant identity;

* 403 Forbidden with invalid-participant-binding when the participant is authenticated but not authorized for the targeted participant state;

* 409 Conflict with reassociation-required, stale-association, or policy-instance-mismatch;

* 422 Unprocessable Content with unsupported-taxonomy-term or term-resolution-failed; and

* 503 Service Unavailable with policy-authority-unavailable.

A participant that receives an error during renewal or reassociation MUST NOT assume that it still has current association unless the service endpoint has explicitly confirmed that state.

9. Security Considerations

Candidate discovery and endpoint trust are separate concerns. A participant MUST authenticate the selected PPD service endpoint according to the deployment’s security profile before treating policy information as authoritative.

All normative PPD participation defined by this document is authenticated participation. Deployments MUST NOT present unauthenticated local testing or demo operation as equivalent to direct-constrained or direct-certificate participation when making claims about protected current association.

All normative PPD participation defined by this document MUST provide:

* participant authentication sufficient to bind registration and acknowledgment state to the same participant identity;

* participant-facing confidentiality and integrity protection for registration, policy retrieval, and acknowledgment exchanges;

* policy integrity sufficient to identify the acknowledged policy instance unambiguously;

* freshness protection sufficient to prevent replay of old acknowledgments as evidence of current association; and

* protected storage or export of acknowledgment records when those records are used for later inspection or accountability.

When a PPD service endpoint fronts a distinct policy authority, the deployment MUST preserve the authenticity and integrity of policy instances, policy hashes, and freshness metadata across that internal boundary. This document does not standardize the internal protocol used for that purpose.

The protocol SHOULD minimize metadata exposure during discovery, registration, and policy retrieval. In particular, discovery metadata and unauthenticated error responses SHOULD avoid exposing household policy contents, participant inventories, or acknowledgment history. [RFC7258] remains relevant to these design choices.

10. Internationalization Considerations

Where policy tags, labels, or other string identifiers are exchanged in this protocol, future profiles SHOULD define comparison and storage behavior that is consistent across vendors and locales. Where internationalized strings are used, alignment with [RFC7564] SHOULD be considered.

11. IANA Considerations

This document has no IANA actions.

12. References

12.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

12.2. Informative References

[I-D.draft-dsmullen-ppd-architecture] Smullen, D. and B. Scriber, "Privacy Preference Declaration for Home Networks", Work in Progress, Internet-Draft, draft-dsmullen-ppd-architecture-08, 20 May 2026, <https://datatracker.ietf.org/doc/html/draft-dsmullen-ppd-architecture-08>.

[I-D.draft-dsmullen-ppd-taxonomy] Smullen, D. and B. Scriber, "Privacy Preference Declaration Taxonomy", Work in Progress, Internet-Draft, draft-dsmullen-ppd-taxonomy-04, 20 May 2026, <https://datatracker.ietf.org/doc/html/draft-dsmullen-ppd-taxonomy-04>.

[RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <https://www.rfc-editor.org/rfc/rfc3339>.

[RFC7258] Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is an Attack", BCP 188, RFC 7258, DOI 10.17487/RFC7258, May 2014, <https://www.rfc-editor.org/rfc/rfc7258>.

[RFC7564] Saint-Andre, P. and M. Blanchet, "PRECIS Framework: Preparation, Enforcement, and Comparison of Internationalized Strings in Application Protocols", RFC 7564, DOI 10.17487/RFC7564, May 2015, <https://www.rfc-editor.org/rfc/rfc7564>.

[RFC8006] Niven-Jenkins, B., Murray, R., Caulfield, M., and K. Ma, "Content Delivery Network Interconnection (CDNI) Metadata", RFC 8006, DOI 10.17487/RFC8006, December 2016, <https://www.rfc-editor.org/rfc/rfc8006>.

[RFC8785] Rundgren, A., Jordan, B., and S. Erdtman, "JSON Canonicalization Scheme (JCS)", RFC 8785, DOI 10.17487/RFC8785, June 2020, <https://www.rfc-editor.org/rfc/rfc8785>.

[RFC9388] Sopher, N. and S. Mishra, "Content Delivery Network Interconnection (CDNI) Footprint Types: Country Subdivision Code and Footprint Union", RFC 9388, DOI 10.17487/RFC9388, July 2023, <https://www.rfc-editor.org/rfc/rfc9388>.

Authors' Addresses

Daniel Smullen
CableLabs
Email: d.smullen@cablelabs.com

Brian Scriber
CableLabs
Email: brian.scriber@computer.org