Workgroup:	RPP (RESTful Provisioning Protocol) Working Group
Internet-Draft:	draft-ietf-rpp-architecture-01
:	
Published:	6 November 2025
Intended Status:	Informational
Expires:	10 May 2026
Authors:	P. Kowalik	M. Wullink
---	---	---
	DENIC eG	*SIDN Labs*

RPP Architecture

Abstract

Advancements in development, integration, deployment environments and operational paradigms have led to a desire for an alternative for the Extensible Provisioning Protocol (EPP). This document defines the architecture for the RESTful Provisioning Protocol (RPP) an HTTP based provisioning protocol leveraging the REST architectural style and JSON data-interchange format, aiming to standardize a RESTful protocol for provisioning database objects. The architecture includes support for extensibility, allowing for multiple possible use cases. RPP is intended to co-exist with EPP, offering an alternative protocol including data model compatibility with EPP core objects and the benefits associated with the REST architectural style and widely adopted HTTP-based technologies.

Contributing

When contributing to this document, please use the following GitHub project: https://github.com/pawel-kow/RPP-architecture.

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/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 10 May 2026.

Copyright Notice

Table of Contents

1. Introduction

This document outlines the architecture of the RESTful Provisioning Protocol (RPP). RPP aims to provide a modern, standardised, and developer-friendly protocol for provisioning and managing objects in a shared database or registry, initially focusing on functional equivalents of EPP object mappings for domain names [RFC5731], hosts [RFC5732], and contacts [RFC5733]. RPP also considers provisioning of other objects as a potential use case, aiming for a uniform API layer for various registry operations.

RPP is designed to leverage the benefits of REST (REpresentational State Transfer), including statelessness, ease of integration, and compatibility with existing web infrastructure and tooling such as OpenAPI, API gateways, and web application firewalls. By adopting JSON as the data-interchange format, RPP seeks to align with current development practices and the successful deployment patterns observed in protocols such as RDAP [RFC9082]. The choice of REST and JSON also facilitates direct browser and mobile application integration including modern security mechanisms such as OAuth2.0.

This architecture document serves as a foundation for a series of specifications that will collectively define RPP. It details the layered approach, core components, and design considerations for building an interoperable and extensible provisioning protocol. RPP is intended to coexist with EPP, offering an alternative for implementers seeking a RESTful approach without aiming to replace EPP or define migration paths from EPP. RPP aims for data model compatibility with EPP core objects to allow automatic and mechanical mapping and conversion, especially for core objects (domain, contact, host).

2. Terminology

This document uses terminology from RFC5730 [RFC5730] and broadly adopts the REST architectural principles as defined in [REST] and related RFCs.

- **RPP:** RESTful Provisioning Protocol. The protocol being defined by the RPP working group.
- **EPP:** Extensible Provisioning Protocol as defined in [RFC5730].
- **REST:** Representational State Transfer architectural style [REST].
- **JSON:** JavaScript Object Notation [RFC8259].
- **JWT:** JSON Web Token [RFC7519].
- **OpenAPI:** The OpenAPI Specification (OAS) (formerly known as Swagger Specification) is an API description format for REST APIs [OpenAPI].
- **RPP client:** An entity or application that interacts with the RPP server to perform provisioning operations, such as creating, updating, or deleting resources.

3. Requirements

This document is based on the requirements defined by RPP WG in state from 7.3.2025 [RPPReq].

The actual state of the requirements is present on WG Wiki https://wiki.ietf.org/en/group/rpp/requirements.

4. Architectural Overview

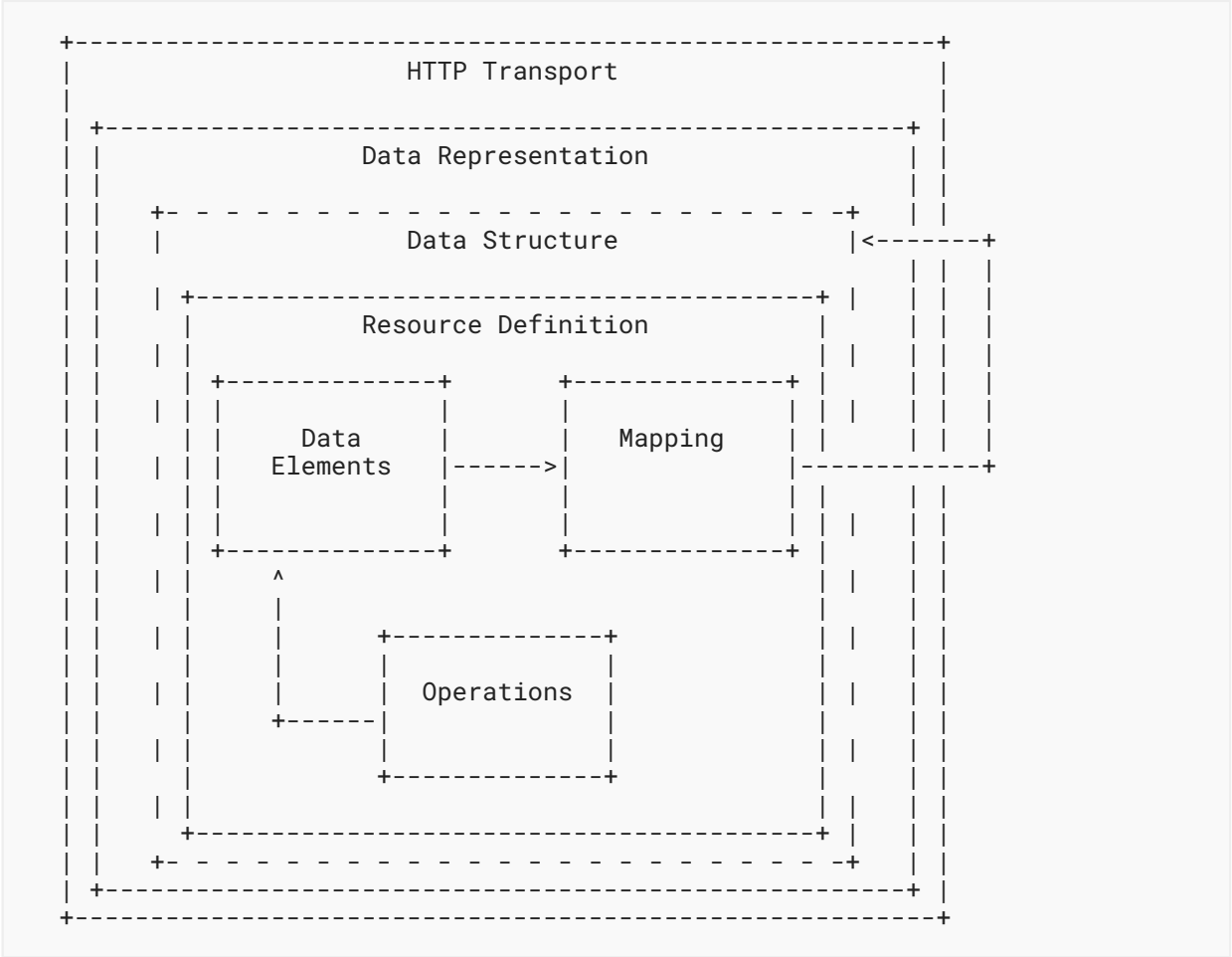
This chapter provides an overview of the RESTful Provisioning Protocol (RPP) architecture. A key design principle is to leverage existing web standards and principles, particularly HTTP and REST principles. This allows RPP to delegate functionality and features to the well-established infrastructure and semantics of the web, focusing its own definitions on the specific domain of object provisioning. Therefore, we assume:

- **HTTP and RESTful principles are foundational:** RPP leverages HTTP for transport and adheres to RESTful principles for resource management.
- **Domain-specific logic resides in data representations:** The specifics of resource provisioning are encoded within the data structures and semantics of the RPP message bodies.
- **Layered architecture for modularity:** The architecture is layered to promote modularity, separation of concerns, and independent evolution of different aspects of the protocol.

The architecture is divided into three main layers: **HTTP Transport**, **Data Representation**, and **Resource Definition**. Each layer defines specific aspects of the protocol. This layered approach allows for clear separation of concerns.

Data Structure is a sub-layer of Data Representation and described later in this document. It focuses on the structure of RPP messages.

Similarly **Data Elements**, their **Mapping** onto Data Structure and **Operations** are elements of Resource Definition. They focus on the semantic structure of RPP resources and transformation of those resources.



4.1. Resource Oriented Architecture

RPP adopts a Resource Oriented Architecture (ROA), aligning with RESTful principles. This approach defines all manageable entities as "resources," identified by unique URLs. Operations on these resources are performed through a uniform interface using the standard HTTP methods and their semantics. This contrasts with RPC-style protocols, which often define new and specific operations with custom parameters. ROA promotes a more standardised and interoperable approach, leveraging the existing web infrastructure and its well-defined semantics. Key aspects of ROA within RPP include:

- **Resource Identification:** Each resource is uniquely identifiable by a URL.
- **Uniform Interface:** HTTP methods (HEAD, GET, POST, PUT, DELETE, PATCH) are used to perform operations on resources in a consistent manner.
- **Operation Singularity** Operations, excluding collection retrieval, are defined to target a single resource. Operations intended to affect multiple resources, such as bulk operations (a single command applied to multiple resources) or command sets (multiple commands on multiple resources), should be modelled through dedicated "batch" or "bulk operation" resources.

- **Representation:** Resources can be represented in various formats (e.g., JSON, XML) through HTTP standard content negotiation.
- **Statelessness:** Each request to a resource is treated as independent of previous requests. The server does not maintain client state between requests.
- **Cacheability:** Responses can be cached to improve performance.

4.2. Architecture Layers

4.2.1. HTTP Transport Layer

This layer defines the transport mechanism for RPP messages, utilising HTTP as the underlying protocol.

The HTTP Transport Layer consists of two sub-layers:

4.2.1.1. Built-in HTTP Features

These are features that are fully specified by the HTTP standard itself. RPP leverages these features directly, specifying their use within the protocol without redefining their behaviour.

It encompasses aspects such as:

- **Authentication and Authorisation:** Mechanisms for verifying the identity of clients and controlling access to resources.
- **Resource Addressing using URLs:** Consistent and meaningful URL structures for identifying, accessing resources and enabling request routing.
- **Mapping of basic operations to HTTP uniform interface (verbs):** Mapping CRUD (Create, Read, Update, Delete) operations to POST, HEAD/GET, PUT/PATCH, and DELETE respectively.
- **Mapping of operations beyond HTTP uniform interface to URLs and verbs:** Handling more complex operations through appropriate URL structures and HTTP methods.
- **HTTP response codes:** Utilising standard HTTP status codes to indicate the outcome of requests.
- **Content negotiation for media types:** Supporting multiple data representation formats and using content negotiation to select the appropriate format.
- **Caching:** Leveraging HTTP caching mechanisms to improve performance.
- **Language negotiation for textual content:** Supporting multiple languages for textual content and using language negotiation to select the appropriate language.
- **Representation preferences:** The client may define whether a full representation of an object or only a limited representations is preferred as a response
- **Validation preferences:** Defining the approach to the input data validation that shall be applied by the server. The server may apply lenient validation where the server makes best effort to process the understood part of the request as opposed to strict processing where the server would reject a request where part of the data is not understood

4.2.1.2. RPP-Specific Extensions

These are protocol features where HTTP provides the necessary building blocks, but RPP defines additional rules, conventions, or mechanisms to address protocol-specific requirements.

It encompasses aspects such as:

- **Asynchronous Operation Management:** Facilitating the handling of operations that are not completed immediately, by defining an HTTP-based interaction pattern for status checking and deferred result retrieval.
- **RPP specific error codes and relation to HTTP error codes:** Defining RPP-specific error codes while relating them to standard HTTP error codes for consistency.
- **Transaction tracing and idempotency:** Mechanisms for tracking requests and ensuring idempotent operations where appropriate.
- **Versions and profiles:** Support signalling of versions of RPP protocol and other protocol elements as well as defining sets of protocol elements and their versions in the form of profiles.
- **Definition of special resources:** Defining specific resources for service discovery, metadata retrieval, etc.
- **Service discovery mechanisms:** Mechanisms for clients to discover available RPP services.

4.2.2. Data Representation Layer

This layer focuses on the data representation of RPP messages. It defines the media type used to carry RPP data and supports various data representation formats.

It encompasses aspects such as:

- **Data structure:** Defining the structure and schema of the RPP data, potentially using a specific schema language.
- **Data format:** Defining the specific format used to represent RPP data within the representation (e.g., JSON, XML or JWT).
- **Media Type definition:** Defining the specific media type to be used in RPP, including any constraints on the data format and structure

4.2.3. Resource Definition Layer

This layer defines the structure and operations for each resource type, independent of media type or representation. It ensures resources are well-defined and allows for easy extensibility and compatibility with different media types.

It encompasses aspects such as:

- **Data elements:** Defining the individual data elements that make up a resource, including their data types, formats, and any constraints.
- **Resource type definitions:** Defining the structure of specific resource types by combining data elements.

- **IANA registry definitions:** Potentially registering resource definitions with IANA for standardised and automated processing.
- **Mapping of data elements to media types:** Defining how the data elements of a resource type are represented in different media types (e.g., JSON, XML).
- **Extension mechanisms:** Providing mechanisms for creating new resource types and for extending existing resource types with new data elements or operations including potentially new response status codes.

5. Protocol Details

This section provides further details on each layer of the RPP architecture.

5.1. HTTP Transport Layer Details

The RPP architecture uses the best practices described in [RFC9205] for the HTTP transport layer.

5.1.1. Authentication and Authorisation

RPP is aimed to leverage scalable and modern authorisation standards, with a focus on OAuth 2.0 [RFC6749] and related frameworks. However, to maintain functional equivalence with EPP client authentication, RPP SHOULD also support authentication schemes that can carry a client identifier and a password, such as HTTP Basic Authentication. RPP should be able to support future authentication and authorisation standards defined for HTTP.

Specifications will define profiles for:

- HTTP Authentication schemes (e.g., HTTP Basic Authentication, Bearer Token [RFC6750] etc.)
- Authorisation frameworks (e.g., OAuth 2.0 [RFC6749])

Implementations will be able to choose authentication and authorisation methods appropriate for their security requirements.

5.1.1.1. Authorisation Scopes

RPP specifications will standardise authorisation scopes (like rpp:read or rpp:write) to define granular access control for different usage scenarios. These scopes will be defined for various operations and resource types, ensuring that clients can be granted only the necessary permissions.

5.1.1.2. Fine-Grained Authorisation

RPP authorisation models may become fine-grained, extending beyond simple auth-code based models used in EPP. Authorisation decisions will be able to consider the specific operation being performed (e.g., update vs. read), the resource being accessed (e.g., a specific domain name), and potentially even attributes within the resource.

Here solutions like OAuth2 RAR [RFC9396] could be considered to provide fine-grained access control.

5.1.1.3. Relationship between clients and authentication credentials

RPP authentication and authorisation model will make a clear distinction between the login credentials and the authorisation to act in context of a given RPP client. More than one credential might be authorised to act on behalf of the same RPP client. The same credential however must always be assigned to one and only one RPP client context.

In case of HTTP Basic Authentication, one user-id is always bound to at most one RPP client. For OAuth, the issued token is bound to the context of at most one RPP client, even though the OAuth client itself might have access to multiple RPP clients. The assignment of tokens to specific RPP clients can be controlled through the authorisation flow using the OAuth scope parameter. For example, if an OAuth client has access to two RPP clients (Client A and Client B), the scope parameter can specify which client the token applies to. A scope value like `scope=rpp:clientA` would ensure the token is valid only for Client A, while `scope=rpp:clientB` would apply to Client B.

5.1.1.4. Security

RPP will not explicitly define security related policies related to authentication or authorisation (such as password complexity, token lifetime or cryptography used) on the protocol level. Instead, these properties will be delegated to the best practices of the chosen authentication schemes, which may evolve over time and would have to be independent of the protocol itself.

5.1.1.5. Object level authorisation

RPP will define a mechanism for object-level authorisation, preventing unauthorised access to specific objects or resources. Each object will have an associated sponsor or owner with full control over an object, and the protocol will allow for the specification of which clients are authorised to access or modify non-sponsored/owned objects. This could be achieved through state-of-the-art standards like OAuth authorisation tokens, scopes, and resource-specific permissions but also shared secrets for backward compatibility with EPP password-based authorisation information.

5.1.2. Resource Addressing

RPP resources are addressed using URLs. Considerations include:

- Hierarchical URL structure to represent resources of different types (e.g., `/domains/{domain-name}`, `/contacts/{contact-id}`).
- URL structure to represent list of related resources (e.g., `/domains/{domain-name}/contacts/`)

RPP URL structure will be designed to be human-readable, intuitive, and RESTful, allowing clients to easily navigate and interact with resources.

RPP would not require all URLs to be hard wired to server's RPP root URL. Instead, it would allow for relative URLs to be defined and discovered by the client. This would allow servers to distribute resources across multiple servers and URLs and allow for easier scaling as described in [RFC9205]. At the same time the URLs shall be deterministic for the duration of the client session in order to minimise round trips and streamline the interaction.

As a matter of extensibility consideration RPP should allow for additional path segments to be added to the URLs and be discoverable by clients.

RPP responses will include URLs for related resources, allowing clients to navigate newly created resources easily. This is similar to the "links" concept in RESTful APIs, where related resources are linked together.

5.1.2.1. Internationalised Domain Names (IDN)

RPP will address the handling of Internationalised Domain Names (IDNs) in resource addressing. Specifications will define whether to use IDN or UTF-8 encoding directly in URLs and whether to employ redirects to canonical URLs or "see-also" linking for alternative representations. For example, a "see-also" link could point from a UTF-8 encoded URL to an IDN URL and vice versa, allowing clients to use either URL. Another way would be to always redirect to the canonical URL, which would be the IDN URL.

5.1.3. Mapping of basic operations to HTTP uniform interface (verbs)

RPP operations are mapped to standard HTTP methods to leverage the uniform interface and RESTful principles:

- **HEAD:** Retrieve resource state (e.g., retrieving domain existence information). This may be a candidate for equivalence of EPP check command, however it may come with a few caveats to consider:
  - EPP check is intended to check whether (future) resource provisioning is possible. This is not semantically the same as resource state. Overloading HEAD with EPP semantics may lead to confusion, especially as some frameworks implicitly implement HEAD out of GET handling.
  - a better equivalence of EPP check would be a POST with Expect header, however this header being a reserved header in browsers it may not be availble to all client implementations

  The conclusion is that RPP should not overload HTTP HEAD with own semantics, and support HEAD as it is defined in HTTP.
- **OPTIONS** RPP shall not define any new semantics for OPTIONS however the specifications should make implementers aware of its role in CORS and pre-flight requests typically made by web browsers
- **GET:** Retrieve resource state (e.g., retrieving domain or contact information) - EPP info command

- **POST:** Create a new resource (e.g., registering a domain or create contact object) - EPP create command
- **PUT:** Update an existing resource in its entirety (e.g., updating domain registration details) - not 100% equivalent of EPP update command
- **DELETE:** Delete a resource (e.g., deleting a domain registration) - EPP delete command
- **PATCH:** Partially modify a resource (e.g., updating specific attributes of a domain or contact) - EPP update command

5.1.4. Mapping of operations beyond HTTP uniform interface to URLs and verbs

Many of EPP commands do not map directly to the HTTP uniform interface. RPP will define how to handle these operations using appropriate URL structures and HTTP methods. In order to model additional operations RPP will define an abstraction of process, this being either a transient or a long running operations with state. In both cases such process may accept additional input data as well as have an outcome or result not being part of the resource state itself. Processes shall therefore be modelled as separate sub-resources of the resource being processed, with own uniform interface and set of operations (CRUD).

EPP transfer commands (query and transform), may be modelled by a subresource `/transfer` of the resource being transferred, or a collection `/transfers` with a PUT/POST operation correspondingly to initiate the transfer, GET operation to query the transfer status and POST operation to approve or reject the transfer.

Other transform operations like renew, or restore which are not directly addressable resources in terms of REST will modelled as a convention of URLs with processing resources with only POST interface, e.g. `/renewal`.

This pattern can be further applied to object operations not defined in the core protocol or EPP, allowing easy and uniform extension of allowed operations.

In order to minimise name collisions between process names and other kind of sub-resources, a distinct path segment shall be dedicated to processes, e.g. `/processes/{process-name}` or `/operations/{operation-name}`.

The path segments shall be appended to the resource path to create full URI of such processing resource (e.g. `/domains/{domain-name}/processes/renewal` for domain renew operation).

As discussed in Section 5.1.3 EPP check command may not be appropriate to be mapped directly to the HTTP uniform interface of the resource itself. EPP check command will be therefore modelled as sub-resource such as `/availability` offering both HEAD operation for quick yes/no response and GET operation for more detailed response allowing for example to extend on the dataset provided (e.g. pricing information).

This basic set of rules and guidelines will be further refined in the RPP specifications and give a universal toolset for extending RPP with new resources and commands.

5.1.5. HTTP response codes

In general RPP shall make use of HTTP status codes to indicate general response status categories (e.g., 2xx success responses, 4xx for client errors, 5xx for server errors) [RFC7231] as opposed to responding always 200 if the request was understood even if the operation itself failed. This allows clients and intermediaries to make first level of determination of the requests outcome based on the status code alone, without needing to parse the response body.

RPP shall use as specific HTTP codes as possible, using generic codes like 400 Bad Request or 500 Internal Server Error only if no corresponsing specific code exists (e.g. 404 for not existing resource or 401 for unauthenticated request).

More specific RPP codes are elaborated in Section 5.1.12.

5.1.6. Content negotiation for media types

RPP supports content negotiation to allow clients to specify preferred media types for request and response payloads using the HTTP 'Accept' and 'Content-Type' headers [RFC7231].

- 'application/rpp+json' as the primary media type.
- potential media type parameters for versioning, profiles, and other protocol elements.
- Potential support for other media types defined in the Section 5.2

5.1.7. Caching

RPP shall benefit from HTTP standard caching mechanisms to enable standard components like proxies and caches to improve performance and reduce load on servers. RPP shall define caching policies for different resources and operations, including cache-control headers and ETag support.

5.1.8. Language negotiation for textual content

RPP shall support language negotiation to enable clients to request responses in a preferred language using the HTTP 'Accept-Language' header [RFC7231].

- Server implementations MAY support multiple languages for textual content in responses to provide human-readable localised responses.
- The default language and mechanisms for indicating supported languages will be defined, preferably using HTTP methods, like OPTIONS or HEAD requests.
- application/rpp+json media type may support multi-language representations, especially for writing operations involving user provided content. Other media types may have different mechanisms for language representation.

5.1.9. Client Signalling for Response Verbosity

RPP may utilise the HTTP `Prefer` header [RFC7240] with the "return" preference to allow clients to control the verbosity of responses. For example, clients not interested in full resource representations could use `Prefer: return=minimal` to request minimal responses, reducing payload sizes and improving efficiency. The default behaviour, without the `Prefer` header,

would be to return a full resource representation, similar to object info responses in EPP, especially after compound requests are completed. For certain use-cases it might be convenient for a client to receive also dereferenced full or partial representation of related objects. For example details about sponsoring client of domain name instead of just ID. "return" preference syntax alone is not sufficient for this purpose, therefore RPP would need to define custom preference and register it in "HTTP Preferences" IANA registry.

5.1.10. Client Signalling for Request Validation

RPP may utilise the HTTP `Prefer` header [RFC7240] for signalling the preference for either strict or lenient processing of requests. This allows clients to indicate whether they prefer strict validation of message payloads and rejection of requests with unknown properties or a more lenient approach ignoring unknown properties that may allow for additional flexibility in processing. The default behaviour, without the `Prefer` header, would be to process requests leniently.

5.1.11. Asynchronous Operation Processing

The RPP architecture accommodates operations that are potentially long-running or cannot be completed synchronously due to their nature (e.g., acting on multiple objects, resource-intensive tasks, or processes involving manual steps). This is achieved by leveraging standard HTTP mechanisms to provide an asynchronous interaction pattern. This pattern allows a client to initiate an operation and receive an immediate acknowledgement, with the means to check the operation's status and retrieve its outcome at a later point.

The typical interaction flow facilitated by the architecture is as follows: 1. A client initiates an operation via an HTTP request. 2. For operations processed asynchronously, the server typically responds immediately with an appropriate HTTP status code and an indication of a status resource where the client would be able to obtain result of the operation. The resource may be dedicated to the specific performed operation, be a subresource of the resource being processed, or be a separate message queue resource with a stream of operation results. 3. The server may also provide additional signalling in the response to indicate the expected time for completion or other relevant information using standard HTTP mechanisms. 4. The representation of the status resource reflects the operation's progress. Once the operation concludes, this representation indicates the final outcome, providing either the results directly, links to the results, or detailed error information in line with RPP's error reporting principles. It shall remain up to protocol design for certain operation and server policy which granularity of status information shall be offered. In some cases it might be sufficient to have one final message, in other cases intermediate statuses might be required. The lifetime of these resources might also be differentiated. Messages in the queue would exist until they are read out by the RPP client. Other status resources might exist for a specific time defined by the server after the processing reached its final state. Finally resources might virtually exist forever or require an explicit delete operation from the client.

This architectural approach to asynchronous operations allows client applications to remain responsive and manage extended processing times effectively, contributing to the overall scalability and robustness of interactions within the RPP ecosystem. Specific RPP operations intended for asynchronous execution will be designed to utilise this pattern.

5.1.12. RPP specific status codes and relation to HTTP status codes

RPP uses a dual-layer approach for signaling operation outcomes, leveraging both standard HTTP status codes and RPP-specific status codes. This allows for compatibility with generic HTTP components while providing detailed, application-level feedback for RPP clients.

- The HTTP status code [RFC7231] must be used to convey the overall outcome of an operation. Any HTTP-aware component, such as a proxy or monitoring tool, can determine if a request was successful (2xx), resulted in a client error (4xx), or a server error (5xx) by inspecting the HTTP status code alone.
- RPP-specific status codes, transmitted in a dedicated HTTP header, must be used to provide granular, application-level information about the operation's result.
- RPP responses must include both an HTTP status code and an RPP-specific status code, however in some cases request handling may be terminated already on HTTP level, for example due malformed HTTP message, in such cases only HTTP status code will be present.
- RPP-specific status codes should be mapped to the most semantically appropriate HTTP status code. For example, an RPP status code indicating "object already exists" during a creation attempt should map to HTTP 409 (Conflict), while a code for "object does not exist" during a lookup should map to HTTP 404 (Not Found). If no specific HTTP status code is a good semantic fit, a generic code (e.g., 400 for a general client-side business rule failure, 200 for a successful operation with additional information) should be used.
- This mechanism applies to both successful and unsuccessful operations. A successful response (e.g., HTTP 200 OK) may include an RPP-specific header to provide additional information, such as warnings, deprecation notices, or details about a partial success.
- In the case of an error, the response body should contain a machine-readable problem details document [RFC9457] to provide further information about the error.
- RPP status codes should be categorised as either temporary or permanent to guide client retry behaviour.
- RPP should also use other standardised HTTP signaling mechanisms where appropriate, for example for rate limiting.

5.1.13. Transaction tracing and idempotency

RPP shall support identification of requests and responses on both client side and server side with use of client provided identifiers and server provided identifiers. This will allow for tracking of requests and responses in case of errors, and for idempotency of requests.

Client provided identifier shall be returned in the corresponding synchronous response and shall be included in the asynchronous responses. This identifier shall be also used as idempotency identifier to allow clients to retry requests without risk of duplicate processing. The client provided identifier shall be unique for the client and the lifetime of the identifier shall be defined by the server, typically for a limited time after the request was processed.

Server shall always generate own unique trasaction identifier, regardless of nature of the transaction (reading or changing).

The transmission of transaction identifiers should be defined outside of the Data Representation Layer (e.g. as HTTP Headers), to assure clear separation of resource representation from performed actions. If possible existing mechanisms of HTTP shall be employed.

5.1.14. Protocol Versioning

RPP will define a versioning schema for the protocol itself, the extensions and other protocol elements such as profiles as appropriate. The versioning schema shall on one side allow for independent introduction of new features in a non-breaking manner on both client and server side, and on the other side allow the opposite party of the communication to determine if the version is compatible or not. One of potential approaches having this property might be use of Semantic Versioning [SemVer], but also other versioning schema shall be possible.

Signalling of the versions will be preferably realised using parameters of the media type.

5.1.15. Profiles

In real operational conditions different RPP server operators may have different requirements regarding set of protocol elements and their versions necessary to be supported by the client to enable reliable communication. Such requirements may also be defined by external policies. For this purpose RPP will define a concept of profiles, being identifiers translated into a certain minimum configuration of protocol version, extensions and their versions. The profiles themselves will be versioned in the same way as other protocol elements.

RPP may define a machine-readable definition of profiles to allow automatic processing by the clients, but may also refer to other forms of profile specification.

Signalling of the profiles will be preferably realised using parameters of the media type.

5.1.16. Definition of special resources

RPP may define special resources for specific purposes:

- Service Discovery endpoints to advertise protocol capabilities and supported features (see Section 5.1.17).
- Metadata endpoints to provide schema information or other protocol-level metadata, potentially including OpenAPI definitions for documentation and code generation.

5.1.17. Service discovery mechanisms

RPP will define mechanisms for service discovery, allowing clients to dynamically discover RPP service endpoints and capabilities, reducing coupling between clients and servers.

- Potential discovery of RPP server location, like IANA bootstrapping document or a special DNS TXT RR with location of RPP service for the tld.
- Potential use of well-known URIs (e.g., `/.well-known/rpp-capabilities`) for service discovery.
- Advertising supported protocol versions, extensions, available resource types, authentication methods, and supported features.

- It may be considered for RPP to distribute service discovery for each resource type separately for better scalability and management. For example instead of having a single service discovery endpoint for the whole registry on `/.well-known/rpp-capabilities` there might be a separate discovery placed under `/{resource-type}/.well-known/rpp-capabilities` e.g. `/domains/.well-known/rpp-capabilities`.
- Service discovery shall utilise standardised methods, like URI templates [RFC6570] to allow easy navigation of resources and avoid hard-coding of URLs, same time allowing clients to navigate directly to a known resource without additional server queries.
- As a matter of principle service discovery shall be used to bootstrap the communication between client and server, its capabilities and operational policies. The server configuration shall be considered static between reconfiguration of the server and not be used for any dynamic configuration, like load balancing etc. It would enable the right balance between discoverability and reduction of round trips between clients and servers.

5.2. Data Representation Layer

This layer focuses on the data representation of RPP messages. It defines the media type used to carry RPP data and supports various data representation formats.

5.2.1. Data structure

RPP will define the overall structure of the message payload carried by the chosen media type. By default one data structure will be defined, however RPP should be able to support multiple data structures, especially for compatibility with EPP and other standards.

- **'RPP' Structure:** Defining a new, dedicated data structure specifically for RPP messages. This would be the default in core specifications.

Other future possibilities:

- **'EPP' Structure Adaptation:** Reusing or adapting to the existing EPP XML schemas, to maintain data model compatibility with EPP core objects and simplify mapping from EPP.
- **'JSContact' Structure Adaptation:** Adapting to the existing JSON representation for Contact Information [RFC9553], to maintain alignment with RDAP.
- **'VC' Structure Adaptation:** Adapting to existing Verifiable Credentials ([W3C-VC], [SD-JWT]) data structures, especially for representing identity or authorisation information, allowing for integration with external identity systems.

5.2.2. Data format

The primary format for RPP data representations shall be JSON, however RPP should be able to be extended to support other formats like XML, JWT, JWT-SD or CBOR.

- **JSON:** Standard JSON format [RFC8259].
- **XML:** eXtensible Markup Language [XML] (considered for potential compatibility with EPP).
- **JWT:** JSON data encapsulated within a JSON Web Token [RFC7519] for potential use-cases when verifiable data consistency is required

- **JWT-SD:** JSON data with Selective Disclosure using JWTs [SelDiscJWT] for minimisation of exposed data.
- **CBOR:** Concise Binary Object Representation for specific use cases requiring compact binary encoding [RFC8949].

Some data formats can be optionally represented in other encapsulations, for example JSON data can be represented also in JWT or CBOR. Change of encapsulation shall not affect the data structure. This might be beneficial if RPP is to be extended to support different data formats in the future that only require additional properties provided by encapsulation, like signing, encryption or binary representation.

5.2.3. Data Validation

Data structures and formats will be described using a schema language, such as JSON Schema, OpenAPI, CDDL or other appropriate stable and open standard for JSON data structures. It will enable data validation to be performed by both client and servers on received requests and responses. For example, JSON Schema can define the expected structure of a domain object, including required fields and data types, allowing clients to validate their requests before sending them and servers to ensure incoming data conforms to the expected format. The schemas must support both strict and lenient processing of requests and responses and support protocol extensibility.

5.2.4. Media Type definition

Together data structure and data format would define the whole media type. So application/rpp+json would be the primary media type with "rpp" payloads in plain json format. application/epp+xml would be epp payload as per [RFC5730].

5.3. Resource Definition Layer

Each resource type, no matter if on a top level, being an independent provisioning object, or a subresource, being a part of another resource, shall be well defined including data elements and possible operations. A resource definition shall on the first level of abstraction be composable out of data elements, without any reference to the media type or representation. This will allow for easy extensibility and compatibility with different media types.

All resource types shall be defined in IANA registry in a way that allows fully automated processing of the resource definition, including data elements, operations and media type representation.

5.3.1. Data Elements

This part defines logical data elements for each resource type, which can also be re-used across resource types. It is abstracted from the actual transport and media type, focusing on the structure and constraints of data elements. Data element definition includes:

- Identification of logical data units (e.g. a stable identifier of a data element, which is independent of the representation)
- Definition of logical data units (e.g., domain name, contact details)

- Format and schema for primitive data elements or reference to other resource type definitions
- Constraints on data elements (e.g., data type, length, allowed values)
- Mechanisms for extensibility, if applicable

Data elements shall be defined in IANA registry in a way that allows for automated processing of the data element definition, including constraints and references to other data elements.

5.3.2. Mapping

This layer defines the mapping of Data Elements onto the Data Representation Layer. For example in case of application/rpp+json media type, the mapping layer would define how the logical data units are represented in JSON format.

This additional level of indirection would allow usage of data formats defined outside of rpp specifications - for example usage of Verifiable Credentials or Verifiable Presentations as first class resource types for contacts in RPP, and mapping appropriate data elements.

The mapping layer shall be defined in IANA registry in a way that allows for automated processing of the mapping definition, including reading and writing operations. Mechanisms, such as defined for JavaScript Object Notation (JSON) Patch [RFC6902], may be used to define the mapping.

5.3.3. Operations

Each resource type shall define operations possible on this resource type. This may encompass any of the mechanisms defined on the HTTP transport layer and be constrained by those extensibility rules.

Operations shall be defined in IANA registry in a way that allows for automated processing of the operation definition, including constraints and references to other resource types.

5.4. Extension mechanisms

The RPP architecture is designed to be extensible, allowing for the addition of new resource types, data elements, and operations without breaking existing implementations. This extensibility is achieved through:

- **Layered Design:** Each architectural layer (HTTP Transport, Data Representation, Resource Definition) is defined independently, allowing new features or technologies to be introduced at one layer without impacting others.
- **IANA Registries:** Resource types, data elements, mappings, and operations are registered in IANA registries using machine-readable formats. This enables automated processing, discovery, and extension of protocol elements without requiring changes to the core specifications.
- **Resource and Operation Extension:** New resource types and operations can be defined and registered. Existing resources can be extended with additional data elements or operations in a backward-compatible manner.

- **Profile and Compatibility Layers:** Compatibility profiles can be defined to support subsets of RPP for specific use cases (such as EPP compatibility)
- **Discovery and Negotiation:** Service discovery endpoints and content negotiation mechanisms allow clients and servers to dynamically discover and utilise new capabilities, resource types, and representations as they are introduced.
- **Status codes:** New RPP status codes can be defined and registered including their mapping to HTTP status codes.

RPP shall facilitate standardisation and reuse by requiring that extensions be registered with IANA. These registration requirements shall be fine-grained, applying independently to protocol elements such as resource types, data elements, operations, and status codes. This approach promotes consistency and interoperability, avoids fragmentation from conflicting definitions, and allows protocol elements to evolve independently.

These extensibility mechanisms ensure that RPP can evolve to meet future requirements, integrate with emerging technologies, and support a wide range of provisioning scenarios while maintaining interoperability and stability.

5.4.1. Name Management and Collision Avoidance

RPP extensions MUST define unique names for all extension elements to prevent conflicts with other extensions and with core protocol elements. These names are used consistently in resource identifiers, data element identifiers, and URL path segments.

Standardised RPP extensions MUST register their names in a dedicated IANA registry for RPP extensions to ensure global uniqueness and avoid collisions. Private (non-standardised) extensions are also required to use unique names, but are not required to register with IANA. This allows private extensions to be developed and used within specific implementations or organisations without impacting the global RPP ecosystem. Private extensions should use names that are unlikely to conflict with other extensions or with RPP core elements, for example by using reverse domain notation as a prefix (e.g., `org.example.rpp`).

This naming mechanism ensures that new resource types, data elements, and operations can be introduced independently and safely, supporting the extensibility goals of the RPP architecture while maintaining interoperability and clarity across implementations.

6. Change History

6.1. -00 to -01

6.2. -03 to draft-ietf-rpp-architecture-00

- No changes. Document adopted by RPP WG.

6.3. -02 to -03

- split into HTTP built-ins and custom RPP in Section 4.2.1. Reorganised Section 5.1 to reflect that and added/rewritten some parts to cover all aspects.
- Clarified the relationship and mapping between HTTP status codes and RPP-specific status codes in Section 5.1.12.
- Added paragraph on promotion of standard extensions to Section 5.4.
- Fixed broken references to non-RFC documents
- Explicitly stated that RPP SHOULD support client_id/password authentication for EPP compatibility in Section 5.1.1.

6.4. -01 to -02

- Added responses must contain links to relevant object to Section 5.1.2
- Added round trip minimisation principle to Section 5.1.17
- Added description of lenient versus strict request validation to Section 4.2.3.
- Added description of asynchronous handling to Section 4.2.1 and Section 5.1.
- Added Operation Singularity to Section 4.1.
- Added Versioning chapter to Section 5.1.
- Added Profiles chapter to Section 5.1.
- Added Security section to Section 5.1.1.
- Added Relationship between clients and authentication credentials to Section 5.1.1.
- Add Data Validation (Section 5.2.3) section with schema language support for RPP to Section 5.2
- Added Name Management and Collision Avoidance section to Section 5.4.
- Added Section about dereferenced related object representation to Section 5.1.16.

6.5. -00 to -01

- Removed requirements and replaced with a reference to RPP WG
- Encapsulation removed as a primary extension point and part of architecture
- Added reference to JSContact as a possible contact representation
- Added HEAD verb to basic operations
- Updated RPP specific status codes and relation to HTTP status codes
- Added Extension mechanisms section to Protocol Details

7. References

7.1. Normative References

[RFC9457]

Nottingham, M., Wilde, E., and S. Dalal, "Problem Details for HTTP APIs", IETF, DOI 10.17487/RFC9457, RFC 9457, July 2023, <https://www.rfc-editor.org/info/rfc9457>.

7.2. Informational References

[RFC5730] Hollenbeck, S., "Extensible Provisioning Protocol (EPP)", IETF, DOI 10.17487/RFC5730, BCP 69, RFC 5730, August 2009, <https://www.rfc-editor.org/info/rfc5730>.

[RFC5731] Hollenbeck, S., "Extensible Provisioning Protocol (EPP) Domain Name Mapping", IETF, DOI 10.17487/RFC5731, BCP 69, RFC 5731, August 2009, <https://www.rfc-editor.org/info/rfc5731>.

[RFC5732] Hollenbeck, S., "Extensible Provisioning Protocol (EPP) Host Mapping", IETF, DOI 10.17487/RFC5732, BCP 69, RFC 5732, August 2009, <https://www.rfc-editor.org/info/rfc5732>.

[RFC5733] Hollenbeck, S., "Extensible Provisioning Protocol (EPP) Contact Mapping", IETF, DOI 10.17487/RFC5733, BCP 69, RFC 5733, August 2009, <https://www.rfc-editor.org/info/rfc5733>.

[RFC7231] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", IETF, DOI 10.17487/RFC7231, RFC 7231, June 2014, <https://www.rfc-editor.org/info/rfc7231>.

[RFC7240] Snell, J., "Prefer Header for HTTP", IETF, DOI 10.17487/RFC7240, RFC 7240, June 2014, <https://www.rfc-editor.org/info/rfc7240>.

[RFC8259] Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", IETF, DOI 10.17487/RFC8259, BCP 90, RFC 8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

[RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", IETF, DOI 10.17487/RFC6570, RFC 6570, March 2012, <https://www.rfc-editor.org/info/rfc6570>.

[RFC6749] Hardt, D., "The OAuth 2.0 Authorization Framework", IETF, DOI 10.17487/RFC6749, RFC 6749, October 2012, <https://www.rfc-editor.org/info/rfc6749>.

[RFC6750] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", IETF, DOI 10.17487/RFC6750, RFC 6750, October 2012, <https://www.rfc-editor.org/info/rfc6750>.

[RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", IETF, DOI 10.17487/RFC7519, RFC 7519, May 2015, <https://www.rfc-editor.org/info/rfc7519>.

[RFC9082] Hollenbeck, S. and A. Newton, "Registration Data Access Protocol (RDAP) Query Format", IETF, DOI 10.17487/RFC9082, BCP 95, RFC 9082, June 2021, <https://www.rfc-editor.org/info/rfc9082>.

[RFC6902] Bryan, P. and M. Nottingham, "JavaScript Object Notation (JSON) Patch", IETF, DOI 10.17487/RFC6902, RFC 6902, April 2013, <https://www.rfc-editor.org/info/rfc6902>.

[RFC9396] Lodderstedt, T., Richer, J., and B. Campbell, "OAuth 2.0 Rich Authorization Requests", IETF, DOI 10.17487/RFC9396, RFC 9396, May 2023, <https://www.rfc-editor.org/info/rfc9396>.

[RFC9205] Nottingham, M., "Building Protocols with HTTP", IETF, DOI 10.17487/RFC9205, BCP 56, RFC 9205, June 2022, <https://www.rfc-editor.org/info/rfc9205>.

[RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", IETF, DOI 10.17487/RFC8949, BCP 94, RFC 8949, December 2020, <https://www.rfc-editor.org/info/rfc8949>.

[RFC9553] Stepanek, R. and M. Loffredo, "JSContact: A JSON Representation of Contact Data", IETF, DOI 10.17487/RFC9553, RFC 9553, May 2024, <https://www.rfc-editor.org/info/rfc9553>.

[SD-JWT] Terbu, O., Fett, D., and B. Campbell, "SD-JWT-based Verifiable Credentials (SD-JWT VC)", July 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-oauth-sd-jwt-vc-10>.

[SelDiscJWT] Fett, D., Yasuda, K., and B. Campbell, "Selective Disclosure for JWTs (SD-JWT)", May 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-oauth-selective-disclosure-jwt-22>.

[RPPReq] Wullink, M. and P. Kowalik, "RESTful Provisioning Protocol (RPP) - Requirements", December 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-rpp-requirements-03>.

[XML] Bray, T., Paoli, J., Sperberg-McQueen, C., Maler, E., and F. Yergeau, "Extensible Markup Language (XML) 1.0 (Fifth Edition)", REC-xml-20081126, November 2008, <https://www.w3.org/TR/2008/REC-xml-20081126/>.

[SemVer] "Semantic Versioning 2.0.0", Misc SemVer 2.0.0, <https://semver.org/spec/v2.0.0.html>.

[REST] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", Doctoral Dissertation University of California, Irvine, September 2000, <http://roy.gbiv.com/pubs/dissertation/top.htm>.

[OpenAPI] "OpenAPI Specification", Misc OpenAPI, <https://swagger.io/specification/>.

[W3C-VC] Sporny, M., Longley, D., Chadwick, D., Herman, I., Sporny, M., Thibodeau, T., Herman, I., Cohen, G., and M. Jones, "Verifiable Credentials Data Model v2.0", vc-data-model-2.0, <https://www.w3.org/TR/vc-data-model-2.0/>.

Authors' Addresses

P Kowalik
DENIC eG
Theodor-Stern-Kai 1
Frankfurt am Main
Germany
Email: pawel.kowalik@denic.de
URI: https://denic.de

M Wullink
SIDN Labs
Netherlands
Email: maarten.wullink@sidn.nl
URI: https://sidn.nl/