Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 16 September 2026

M. Wullink
SIDN Labs
P. Kowalik
DENIC

15 March 2026

RESTful Provisioning Protocol (RPP)
draft-wullink-rpp-core-05

Abstract

This document describes the endpoints for the RESTful Provisioning Protocol, used for the provisioning and management of objects in a shared database.

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 16 September 2026.

Copyright Notice

Table of Contents

1. Introduction

This document describes an Application Programming Interface (API) API based on the HTTP protocol [RFC2616] and the principles of [REST]. Conforming to the REST constraints is generally referred to as being "RESTful". Hence the API is dubbed: "'RESTful Provisioning Protocol" or "RPP" for short.

The RPP API is designed to be used for the provisioning and management of objects in a shared database, such as domain names, hosts, and entities.

2. Terminology

In this document the following terminology is used.

REST - Representational State Transfer ([REST]). An architectural style.

RESTful - A RESTful web service is a web service or API implemented using HTTP and the principles of [REST].

EPP RFCs - This is a reference to the EPP version 1.0 specifications [RFC5730], [RFC5731], [RFC5732] and [RFC5733].

RESTful Provisioning Protocol or RPP - The protocol described in this document.

URL - A Uniform Resource Locator as defined in [RFC3986].

Resource - An object having a type, data, and possible relationship to other resources, identified by a URL.

RPP client - An HTTP user agent performing an RPP request

RPP server - An HTTP server responsible for processing requests and returning results in any supported media type.

JWT - JSON Web Token as defined in [RFC7519].

3. Conventions Used in This Document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT","SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

In examples, indentation and white space in examples are provided only to illustrate element relationships and are not REQUIRED features of the protocol.

All example requests assume a RPP server using HTTP version 2 is listening on the standard HTTPS port on host rpp.example. An authorization token has been provided by an out of band process and MUST be used by the client to authenticate each request.

4. Request Headers

A RPP request does not always require a request message body. The information conveyed by the HTTP method, URL, and request headers may be sufficient for the server to be able to successfully processes a request. However, the client MUST include a request message body when the server requires additional attributes to be present in the request message. The RPP HTTP headers listed below use the "RPP-" prefix, following the recommendations in [RFC6648].

* RPP-Cltrid: The client transaction identifier is the equivalent of the clTRID element defined in [RFC5730] and MUST be used accordingly, when the HTTP message body does not contain an EPP request that includes a cltrid.

* RPP-Authorization: The client MAY use this header to send authorization information in the format <method> <authorization information>, similar to the HTTP Authorization header, defined in [RFC9110, Section 11.6.2]. The <method> indicates the type of authorization being used. For EPP object authorization information, for example the authorization information used for domain names described in [RFC5731, Section 2.3], a new authinfo method is defined and MUST be used. The <authorization information> defines the following comma separated fields:
  - value (REQUIRED): Base64 encoded EPP password-based authorization information. Base64 encoding is used to prevent problems when special characters are present that may conflict with the format rules for the Authorization header.
  - roid (OPTIONAL): A Roid as defined in [RFC5731], [RFC5733], and [RFC5730]. The roid is used to identify the object for which the authorization information is provided. If the roid is not provided, then the server MUST assume that the authorization information is linked to the object identified by the URL of the request.

Use of the RPP-Authorization header:

RPP-Authorization: authinfo value=TXkgU2VjcmFRva2Vu, roid=REG-X-123

The value of the RPP-Authorization header is case sensitive. The server MUST reject requests where the case of the header value does not match the expected case. The RPP-Authorization header is specific to the user agent and MUST NOT be cached, as recommended by Section 16.4.2, the server MUST use the correct HTTP cache directives to prevent caching of the RPP-Authorization header.

* RPP-Profile: The client MUST use this header to indicate the profiles is used in the request.

5. Response Headers

The server HTTP response contains a status code, headers, and MAY contain an RPP response message in the message body. HTTP headers are used to transmit additional data to the client and MAY be used to send RPP process related data to the client. HTTP headers used by RPP MUST use the "RPP-" prefix, the following response headers have been defined for RPP.

* RPP-Svtrid: This header is the equivalent of the "svTRID" element defined in [RFC5730] and MUST be used accordingly when the RPP response does not contain an EPP response in the HTTP message body. If an HTTP message body with the EPP XML equivalent "svTRID" exists, both values MUST be consistent.

* RPP-Cltrid: This header is the equivalent of the "clTRID" element defined in [RFC5730] and MUST be used accordingly when the RPP response does not contain an EPP response in the HTTP message body. If the contents of the HTTP message body contains a "clTRID" value, then both values MUST be consistent.

* RPP-Code: This header is the equivalent of the EPP result code defined in [RFC5730] and MUST be used accordingly. This header MUST be added to all responses and MAY be used by the client for easy access to the result code, without having to parse the HTTP response message body.

For the EPP codes related to session management (1500, 2500, 2501 and 2502) there are no corresponding RPP codes.

In order for RPP to be backwards compatible with EPP, RPP will use 5-digit coding of the result codes, where first digit will denote origin specification of the result codes.

For [RFC5730] Result Codes the leading digit MUST be "0". For RPP result codes the leading digit MUST be "1". For avoidance of confusion RPP MUST not define new codes with the same semantic meaning as already defined in EPP.

For RPP codes the remaining 4 digits MUST keep the same semantics as [RFC5730] Result Codes.

* RPP-Queue-Size: Return the number of unacknowledged messages in the client message queue. The server MAY include this header in all RPP responses.

6. Error handling and relation between HTTP status codes and RPP codes

RPP leverages standard HTTP status codes to reflect the outcome of RPP operations. The RPP result codes are based on the EPP result codes defined in [RFC5730]. This allows clients to handle responses generically using common HTTP patterns. While the HTTP status code provides the primary, high-level outcome, the specific RPP result code MUST still be provided in the RPP-Code HTTP header for detailed diagnostics.

The mapping strategy is to use the most specific HTTP code that accurately reflects the operation’s result.

For common and well-defined outcomes, a specific HTTP status code is used. For example, an attempt to access a non-existent resource (EPP code 2302) MUST return 404 Not Found, and an attempt to create a resource that already exists (EPP code 2303) MUST return 409 Conflict. This allows a client to handle these common situations based on the HTTP code alone.

For all other failures, a generic HTTP status code is used. Client-side errors (e.g., syntax, parameter, or policy violations) MUST return 400 Bad Request. Server-side failures MUST return 500 Internal Server Error.

The server MUST return HTTP status codes, following the mapping rules in Table 1.

Table 1: RPP result code and HTTP Status-Code mapping.

HTTP Status-Code	Description	Corresponding RPP result code(s)
Success (2xx)		
200 OK	The request was successful (e.g., for GET or UPDATE).	01000 (in all cases not specified otherwise), 01300, 01301
201 Created	The resource was created successfully.	01000 for resource creating requests (POST/PUT)
202 Accepted	The request was accepted for asynchronous processing.	01001
204 No Content	The resource was deleted successfully.	01000 for DELETE
Client Errors (4xx)		
400 Bad Request	Generic client-side error (syntax, parameters, policy).	02000-02005, 02104-02106, 02300-02301, 02304-02308

403 Forbidden	Authentication or authorization failed.	02200-02202
404 Not Found	The requested resource does not exist.	02303
409 Conflict	The resource could not be created because it already exists.	02302
Server Errors (5xx)		
500 Internal Server Error	Generic server-side error; command failed.	02400
501 Not Implemented	The requested command or feature is not implemented.	02100-02103

Table 1

Some EPP result codes, like 01500, 02500, 02501 and 02502 are related to session management and therefore not applicable to a sessionless RPP protocol.

7. Problem Detail responses for errors

When an error occurs that prevents processing of the requested action, an RPP server MUST respond using a Problem Detail document [RFC9457] detailing what went wrong, or what was not acceptable to the server. The type field MUST be the urn:ietf:params:rpp:problem URN. The status field MUST reflect the HTTP status code. The document MUST contain an errors element, as a list of objects detailing individual errors.

This document consists of the following fields:

type (required, string) This field MUST always contain the fixed string value urn:ietf:params:rpp:error to indicate that this is an RPP Problem Detail response.
title (required, string) A short human-readable description of the Problem Detail type.
errors (optional, array of error objects) MUST contain objects

detailing information about the specific errors that occurred, including optional references to which values in the original request were not acceptable to the server.

The error object consist of the following fields:

type (required, string) This field SHOULD be a URI that identifies the error type. This URI MAY be dereferenceable to obtain human-readable documentation about the error type.
result (required, string) This field MUST contain the RPP result code associated with the error.
paths (optional, list of strings) The JSONPath [RFC9535] to the value(s) in the request that caused the error. This field MAY be omitted if no specific value in the request can be identified as the cause of the error.
reason (required, string) A human-readable detailed description of the error.

RPP specifications and/or extensions MAY add extension fields to the error object, to convey additional information about the causes of the error. For example, to indicate the account balance on a billing failure, the following Problem Detail response could be used:

```
{
  "type": "urn:ietf:params:rpp:error",
  "title": "RPP Error",
  "errors": [{
    "type": "https://rpp.example/problems/billing/billing-failure",
    "result": "02104",
    "reason": "Not enough balance on account to create domain",
    "balance": 10.0,
    "action_cost": 25.0
  }]
}
```

Problem Detail response containing multiple errors for a domain create request using an invalid domain name (_$.example), JSONPaths are specified in the error objects to indicate which value in the request caused the error:

```
{
  "type": "urn:ietf:params:rpp:error",
  "title": "RPP Error",
  "errors": [{
    "type": "https://rpp.example/problems/domains/domain-syntax",
    "result": "02005",
    "paths": ["$.domain.name"],
    "reason": "Invalid character(s) in domain name",
  },
  {
    "type": "https://rpp.example/problems/domains/domain-allowed-length",
    "result": "02004",
    "paths": ["$.domain.name"],
    "reason": "Domain name length must be between 3 and 63 characters",
  }]
}
```

8. Bootstrapping

The client MUST be able to bootstrap itself by discovering the location of an RPP server. Not having a fixed location for the RPP server is a fundamental design principle of RPP, as it allows for a more flexible and scalable architecture. The client MUST use either the IANA registry for RPP servers or a DNS lookup using an SRV record as defined in [RFC2782]. The format and procedure for adding an RPP server to the IANA registry is defined in the IANA Considerations section below.

For DNS-based bootstrapping, an RPP server MUST publish an SRV record in each DNS zone that is served by the RPP server. The owner name of the SRV record MUST be _rpp._tcp.<zone>.

If multiple SRV records are returned, the client MUST select the RPP server according to the priority and weight rules in [RFC2782]. The client MUST ignore SRV records with a target of . (service not available).

The SRV record provides the target host and port of the RPP service. The client MUST construct the URL for the well-known endpoint (defined in the Discoverability section below) as:

* https://<target>:<port>/.well-known/rpp when <port> is not 443
* https://<target>/.well-known/rpp when <port> is 443

The client MUST use the constructed URL to discover the capabilities of the RPP server, as defined in the Discoverability section below.

Example SRV record for an RPP server for the TLD "example" running HTTPS on port 443 at rpp.example.:

```
_rpp._tcp.example. 3600 IN SRV 0 0 443 rpp.example.
```

In this example, the well-known endpoint URL is https://rpp.example/.well-known/rpp.

9. Discoverability

RPP server capabilities MUST be discoverable by clients. The server MUST provide a well-known endpoint at /.well-known/rpp at the root of the RPP server, this endpoint MUST return a JSON document containing the capabilities of the RPP server. The well-known endpoint MUST be accessible without authentication, and the client MUST be able to access this endpoint before authenticating with the server. The well-known endpoint MUST be accessible using the HTTP GET method and MUST return an HTTP status code 200 (OK) if the request was successful. The response message body MUST contain a JSON document describing the capabilities of the RPP server using the following fields:

* base_url: (required, string) The base URL for the RPP API, this is the URL that MUST be used as the base for all endpoint URL templates.
* version: (required, string) The version of the RPP API supported by the server, for example "1.0".
* tlds: (required, array of strings) A list of TLDs supported by the server, for example "example", "org".
* extensions: (optional, array of extension objects) A list of supported extensions, each extension object MUST contain the following fields:
  - name: (required, string) A short name for the extension, for example "registry fee extension".
  - id: (required, string) A unique URN identifier for the extension, for example "urn:ietf:params:rpp:extension:registry-fee".
  - version: (required, string) The version of the extension supported by the server, for example "1.0".
  - url: (required, string) for standard extensions, this MUST be the URL for the extension in the IANA registry, for private extensions, this MUST be the URL for the extension specification, the domain name used in the URL MUST resolve and the URL MUST be accessible to the client.
* profiles: (optional, array of profile objects) A list of supported profiles, each profile object MUST contain the following fields:
  - name: (required, string) A short name for the profile, for example "domain provisioning profile".

    - id: (required, string) A unique URN identifier for the profile, for example "urn:ietf:params:rpp:profile:domain-provisioning-1.0".
    - version: (required, string) The version of the profile supported by the server, for example "1.0".
    - url: (required, string) for standard profiles, this MUST be the URL for the profile in the IANA registry, for private profiles, this MUST be the URL for the profile specification, the domain name used in the URL MUST resolve and the URL MUST be accessible to the client.
* objects: (required, array) A list of supported resource collections, for example "domains", "hosts", "entities".
* authentication: (optional, array of strings) A list of supported authentication methods, for example "Bearer", "Basic".
* endpoints: (required, array of endpoint objects) A list of available endpoints, each endpoint object MUST contain the following fields:
    - name: (required, string) A short name for the endpoint, for example "availability", "info", "poll", "create", "delete", "renewal" or "transfer".
    - url_template: (required, string) The URI template for the endpoint, using the syntax defined in [RFC6570].
* maintenance: (optional, array) An array containing information about upcoming planned maintenance windows of the server, with the following fields:
    - start_time: (required, string) The start time of the maintenance window in ISO 8601 format.
    - end_time: (required, string) The end time of the maintenance window in ISO 8601 format.
    - description: (optional, string) A human-readable description of the maintenance window.

The following template variables are defined for use in RPP endpoint URL templates:

* collection: The resource collection type (e.g., "domains", "hosts", "entities")
* id: The unique identifier for a resource instance within a collection
* process_name: The name of a process associated with a resource (e.g., "transfers", "renewals")
* process_id: The unique identifier for a specific process instance

Example discovery response document:

```
   {
     "base_url": "https://rpp.example/rpp/v1",
     "version": "1.0",
     "tlds": ["example", "org"],
     "extensions": [
       {
         "name": "RPP example extension",
         "id": "urn:ietf:params:rpp:extension:example-extension",
         "version": "1.0",
         "url": "https://www.iana.org/assignments/rpp-extensions/rpp-example-extension-1.
0"
       }
     ],
     "profiles": [
       {
         "name": "EPP compatibility profile",
         "id": "urn:ietf:params:rpp:profile:epp-compatibility",
         "version": "1.0",
         "url": "https://www.iana.org/assignments/rpp-profiles/epp-compatibility-provisio
ning-profile-1.0"
       }
     ],
     "objects": ["domains", "hosts", "entities"],
     "endpoints": [
       {
         "name": "availability",
         "url_template": "/{collection}/{id}/availability"
       },
       {
         "name": "info",
         "url_template": "/{collection}/{id}"
       },
       {
         "name": "poll",
         "url_template": "/messages"
       },
       {
         "name": "create",
         "url_template": "/{collection}"
       },
     ],
     "authentication": ["Bearer"],
     "maintenance": [
       {
         "start_time": "2026-06-01T00:00:00Z",
         "end_time": "2026-06-01T06:00:00Z",
         "description": "Planned maintenance for server upgrades"
       }
     ]
```

```
}
```

9.1. Workflow

The steps for a typical workflow of provisioning an object using RPP without knowing the location and capabilities of the server are as follows, the first three steps are optional, the client can choose to skip any of these steps if it already has the required information from a previous interaction or configuration.

1. Bootstrap (optional): The client discovers the location of the RPP server by looking up the IANA registry for RPP servers or by performing a DNS SRV lookup as defined in [RFC2782].
2. Discover capabilities (optional): The client retrieves the capabilities of the RPP server by sending a GET request to the well-known endpoint at /.well-known/rpp.
3. Extract RPP URLs (optional): The client extracts the base URL and endpoint URL templates from the discovery response, and uses this information to construct the URLs for the desired operations.
4. Perform provisioning operations: The client performs provisioning operations by sending HTTP requests to the appropriate endpoint URLs, using the HTTP method and request message body as required by the specific operation.

10. Versioning

RPP is designed to be extensible and backward compatible. The version of the RPP API is indicated in the URL path, for example: https://rpp.example/rpp/v1/. The server MUST support at least one version of the RPP API, and MUST return a 404 Not Found status code for requests using an unsupported version. The versioning scheme uses the Semantic Versioning format defined in [SemVer], but only the major version number is used to indicate breaking changes. The minor and patch version numbers are not used in an URL path, but can be used in the media type or in the message body to indicate non-breaking changes.

The following RPP elements include versioning support:

* Endpoints: The server MUST support at least one version of the RPP API, and MUST return a 404 Not Found status code for requests using an unsupported version.
* Messages: A request and response message MUST include the version of the RPP API it is compatible with.
* Extensions: RPP extensions MUST include the version of the RPP API they are compatible with.
* Profiles: RPP profiles MUST include the version of the RPP API they are compatible with.

* Media types: RPP media types MUST include the version of the RPP API they are compatible with.
* Result codes: RPP result codes may be added by extensions or updates to the core RPP specification.

10.1. Endpoints

The base_url element of the RPP Discovery response MAY include the version of the RPP API supported by the server. The client MUST use this base_url for all subsequent requests to the server. For example, if the version is 1.2.3, the base_url is https://rpp.example/rpp/v1/, then the client MUST use this URL for all subsequent requests to the server, and MUST not use a different version in the URL path.

10.2. Messages

The version element of the RPP request and response messages MUST include the version of the RPP API that the message is compatible with. The server MUST reject requests with a version that is not supported by the server, and MUST return a RPP Client error code.

10.3. Extensions

The RPP server MUST include the version for each extension in the RPP Discovery response. The client MUST use this version information to determine which extensions are supported by the server, and to ensure that it uses the correct version of the extension when making requests to the server. A request using an extension MUST include the version of the extension. The server MUST reject requests using extensions with a version that is not supported by the server, and MUST return a RPP Client error code. The following is an example of how the version information for an extension can be included in the RPP Discovery response:

```
"extensions": [
    {
      "name": "RPP example extension",
      "id": "urn:ietf:params:rpp:extension:example-extension",
      "version": "1.0",
      "url": "https://www.iana.org/assignments/rpp-extensions/rpp-example-extension-1.
0"
    }
  ],
```

10.4. Profiles

The RPP server MUST include the version for each profile in the RPP Discovery response. The client MUST use this version information to determine which profiles are supported by the server, and to ensure that it uses the correct version of the profile when making requests to the server. A request using a profile MUST include the version of the profile. The server MUST reject requests using profiles with a version that is not supported by the server, and MUST return a RPP Client error code. The following is an example of how the version information for a profile can be included in the RPP Discovery response:

```
"profiles": [
    {
      "name": "EPP compatibility profile",
      "id": "urn:ietf:params:rpp:profile:epp-compatibility",
      "version": "1.0",
      "url": "https://www.iana.org/assignments/rpp-profiles/epp-compatibility-provisio
ning-profile-1.0"
    }
  ]
```

11. Media types

RPP media types are used to indicate the format of the request and response messages, and MUST include a parameter indicating the name of the profile they are compatible with. The server MUST use the profile information in the media type to determine which features, extensions and versions to use when processing the request, and to ensure that it returns a response that is compatible with the client. The client MUST use the profile information in the media type to determine which features, extensions and versions to use when processing the response, and to ensure that it can correctly interpret the response.

The definition of profile parameters in media types is described in section ....

12. Profiles

A profile is a named set of protocol features and versions that are used to define the compatibility and capabilities of RPP server implementations, allowing for better interoperability between different implementations. Using profiles helps to simplify the implementation and deployment of RPP by providing a clear and concise way for the client and server to communicate their capabilities and requirements.

A profile is identified by a unique name and may be published as a standard profile in the IANA registry for RPP profiles, to promote interoperability and standardization across implementations. Standard profiles names MUST use the RPP URN namespace defined in this document, for example: urn:ietf:params:rpp:profile:{profile_name}. The profile name MUST be unique within the RPP URN namespace and SHOULD be descriptive of the features and versions included in the profile.

A profile can also be defined as a private profile, which is not published in the IANA registry, but is used by specific server implementations only. A private profile can be defined by a server operator to specify the features and versions supported by their implementation. If the profile is not published in the IANA registry, then the server operator MUST ensure that the profile name is globally unique to avoid conflicts with other profiles, the use of reverse domain name notation is RECOMMENDED for private profiles to ensure uniqueness.

12.1. Definition

A profile definition MUST contain the following fields, private profiles may contain additional fields as needed:

* name: A unique name that identifies the profile.
* description: A human-readable description of the profile and its intended use.
* version: The version of the profile.
* rpp_version: The minimum version of the RPP protocol that is supported or required for the profile.
* objects: A list of objects allowed for provisioning operations, for example "domains", "hosts", "entities", etc.
* extensions: A list of extensions, including their versions, that are supported or required for the profile.
* profile: A base profile that is extended by this profile. The base profile MUST be published in the IANA registry for RPP profiles, or be made available to the client using other means.

Example JSON representation for a standard profile named "example-profile" that supports RPP version 1.0 and includes two extensions, "rppExample" version 1.0 and "rppOther" version 1.1. The profile also "extends" the "base-profile" profile, which is defined in the IANA registry for RPP profiles and supports RPP version 1.0.

```
{
  "name": "urn:ietf:params:rpp:profile:example-profile",
  "description": "An example profile for provisioning objects using the RPP protocol."
,
  "version": "1.0",
  "rpp_version": "1.0",
  "objects": ["domains", "hosts", "entities"],
  "extensions": [
    {
      "rppExample": {
        "version": "1.0"
      },
      "rppOther": {
        "version": "1.1"
      }
    }
  ],
  "profile": {
    "name": "urn:ietf:params:rpp:profile:base-profile",
    "version": "1.0"
  }
}
```

12.2. Inheritance

A profile can include another profile, which is referred to as "inheriting" from that profile. When a profile inherits from another profile, it means that the features and versions defined in the inherited profile are also included in the inheriting profile. This allows for the creation of more complex profiles by building upon existing profiles. For example, a profile named "example-profile" could inherit from a base profile named "base-profile", which includes a set of common features and versions. The "example-profile" could then add additional features and versions on top of the ones defined in the "base-profile", or it can override some of the features and versions from the "base-profile". This allows for greater flexibility and modularity in defining profiles, as well as promoting reuse of common features and versions across different profiles. However, it is important to note that the use of inheritance in profiles can also make things more complicated, as it can create dependencies between profiles and make it harder to understand the features and versions included in a profile. Therefore, it is recommended to use inheritance with caution and to clearly document the relationships between profiles. The depth of inheritance MUST be limited to 1 to provent excessive complexity, and profile designers MUST NOT create circular dependencies between profiles, where a profile inherits from itself directly.

This is an example of the parent base-profile used in the previous example, it contains a single extension "rppOther" version 1.0 and does not include any other profiles:

```
{
  "name": "urn:ietf:params:rpp:profile:base-profile",
  "description": "A base profile for provisioning objects using the RPP protocol.",
  "version": "1.0",
  "rpp_version": "1.0",
  "objects": ["domains", "hosts", "entities"],
  "extensions": [
    {
      "rppOther": {
        "version": "1.0"
      }
    }
  ],
  "profile":
}
```

The example profile definition shown in Section 12.1, Paragraph 3 uses the "base-profile" as its base and inherits its features and also overrides the version of the "rppOther" extension defined in the base profile.

12.3. Signalling

This document descibes two distinct methods for signalling the profile used in a RPP request or response, the first method uses a dedicated HTTP header, the second method uses media type parameters. The two methods MUST not be used simultaneously in a single request or response. If both methods are used in a single request or response, then the server MUST return an HTTP error response and include a Problem Detail response in the message body.

12.3.1. Header signalling

The client MUST use the RPP-Profile header to indicate the name of the profile that is to be used for the request. The value of this header MUST be of the type parameter described in [RFC8941], the first parameter MUST uniquely identify a profile, for example urn:ietf:params:rpp:profile:example-profile, followed by the version parameter. If the server does not support the indicated profile or version, then the server MUST return an HTTP error response and include a Problem Detail response in the message body.

The ABNF for Profile header value is as follows:

```
profile-header = "profile" "=" profile-name ";" OWS "version" "=" version
profile-name   = token
version        = 1*DIGIT "." 1*DIGIT
```

Example:

```
RPP-Profile: profile=urn:ietf:params:rpp:profile:example-profile;version=1.0
```

12.3.2. Media type parameter signalling

When using Media type parameter signalling, the client and the server MUST use media type parameters in the Accept and Content-Type headers to indicate the name and version of the profile used in the request. The media type parameters MUST be defined as follows:

* profile: The value of this parameter MUST uniquely identify the profile, for example urn:ietf:params:rpp:profile:example-profile.
* version: The value of this parameter MUST indicate the version of the profile used in the request.

The ABNF for media type parameter signalling is as follows:

```
profile-parameter = "profile" "=" profile-name ";" OWS "version" "=" version
profile-name      = token
version           = 1*DIGIT "." 1*DIGIT
```

Example for the media type application/rpp+json with profile parameters indicating the use of the "example-profile" profile version 1.0.:

```
Accept: application/rpp+json; profile="urn:ietf:params:rpp:profile:example-profile"; v
ersion="1.0"
Content-Type: application/rpp+json; profile="urn:ietf:params:rpp:profile:example-profi
le"; version="1.0"
```

13. Endpoints

Endpoints are described using URI Templates [RFC6570] relative to a discoverable base URL, as recommended by [RFC9205]. Some RPP endpoints do not require a request and/or response message.

The RPP endpoints are defined using the following URI Template syntax:

* {c}: An abbreviation for {collection}: this MUST be substituted with "domains", "hosts", "entities" or another collection of objects.
* {i}: An abbreviation for an object identifier, this MUST be substituted with the value of a domain name, hostname, contact-id or a message-id or any other defined object.

A RPP client MAY use the HTTP GET method for executing informational request only when no request data has to be added to the HTTP message body. Sending content using an HTTP GET request is discouraged in [RFC9110], there exists no generally defined semantics for content received in a GET request. When an RPP object requires additional information, the client MUST use the HTTP POST method and add the query command content to the HTTP message body.

13.1. Availability for Creation

The Availability for Creation endpoint is used to check whether an object can be successfully provisioned. Two distinct methods are defined for checking the availability of provisioning of an object, the first method uses the HEAD method for a quick check to find out if the object can be provisioned. The second method uses the GET method to retrieve additional information about the object's availability for provisioning, for example about pricing or additional requirements to be able to provision the requested object.

When the client uses the HTTP HEAD method, the server MUST respond with an HTTP status code 200 (OK) if the object can be provisioned or with an HTTP status code 404 (Not Found) if the object cannot be provisioned.

When the client uses the HTTP GET method, the server MUST respond with an HTTP status code 200 (OK) if the object can be provisioned. The server MUST include a message body containing more detailed availability information, for example about pricing or additional requirements to be able to provision the requested object. The message body MAY be and empty JSON object if no additional information is applicable.

If the object cannot be provisioned then the server MUST return an HTTP status code 404 (Not Found) and include a problem statement in the message body.

As an extension point the server MAY define and the client MAY use additional HTTP query parameters to further specify the check operation or the kind of response information that shall be returned. For example Registry Fee Extension [RFC8748] defines a possibility to request certain currency, only certain commands or periods. Such functionality would add query parameters, which could be used with GET request to receive additional pricing information with the response. HEAD request would not be affected in this case.

The server MUST respond with the same HTTP status code if the same URL is requested with HEAD and with GET.

- Request: HEAD|GET {collection}/{id}/availability
- Request message: None
- Response message: Optional availability response

Example request for a domain name that is not available for provisioning:

```
HEAD domains/foo.example/availability HTTP/2
Host: rpp.example
Authorization: Bearer <token>
Accept-Language: en
RPP-Cltrid: ABC-12345
```

Example response:

```
HTTP/2 404 Not Found
Date: Wed, 24 Jan 2024 12:00:00 UTC
Server: Example RPP server v1.0
RPP-Cltrid: ABC-12345
RPP-Svtrid: XYZ-12345
RPP-code: 01000
Content-Length: 0
```

13.2. Resource Information

The Object Info request MUST use the HTTP GET method on a resource identifying an object instance. If the object has authorization information attached then the client MUST use an empty message body and include the RPP-Authorization HTTP header. If the authorization is linked to a database object the client MUST also include the roid in the RPP-Authorization header. The client MAY also use a message body that includes the authorization information, the client MUST then not use the RPP-Authorization header.

* Request: GET {collection}/{id}
* Request message: Optional
* Response message: Info response

Example request for an object not using authorization information.

```
GET domains/foo.example HTTP/2
Host: rpp.example
Authorization: Bearer <token>
Accept: application/rpp+json
Accept-Language: en
RPP-Cltrid: ABC-12345
```

Example request using RPP-Authorization header for an object that has attached authorization information.

```
GET domains/foo.example HTTP/2
Host: rpp.example
Authorization: Bearer <token>
Accept: application/rpp+json
Accept-Language: en
RPP-Cltrid: ABC-12345
RPP-Authorization: authinfo value=TXkgU2VjcmV0IFRva2Vu
```

Example Info response:

```
HTTP/2 200 OK
Date: Wed, 24 Jan 2024 12:00:00 UTC
Server: Example RPP server v1.0
Content-Length: 424
Content-Type: application/rpp+json
Content-Language: en
RPP-code: 01000
```

TODO: JSON message here

13.3. Poll for Messages

The messages endpoint is used for retrieving messages stored on the server for the client to process.

* Request: GET /messages
* Request message: None
* Response message: Poll response

The client MUST use the HTTP GET method on the messages resource collection to request the message at the head of the queue.

Example request:

```
GET messages HTTP/2
Host: rpp.example
Authorization: Bearer <token>
Accept: application/rpp+json
Accept-Language: en
RPP-Cltrid: ABC-12345
```

Example response:

```
HTTP/2 200 OK
Date: Wed, 24 Jan 2024 12:00:00 UTC
Server: Example RPP server v1.0
Content-Length: 312
Content-Type: application/rpp+json
Content-Language: en
RPP-code: 01301

TODO
```

13.4. Delete Message

* Request: DELETE /messages/{id}
* Request message: None
* Response message: Poll Ack response

The client MUST use the HTTP DELETE method to acknowledge receipt of a message from the queue. The "msgID" attribute of a received RPP Poll message MUST be included in the message resource URL, using the {id} path element. The server MUST use RPP headers to return the RPP result code and the number of messages left in the queue. The server MUST NOT add content to the HTTP message body of a successful response, the server may add content to the message body of an error response.

Example request:

```
DELETE messages/12345 HTTP/2
Host: rpp.example
Authorization: Bearer <token>
Accept: application/rpp+json
Accept-Language: en
RPP-Cltrid: ABC-12345
```

Example response:

```
HTTP/2 200 OK
Date: Wed, 24 Jan 2024 12:00:00 UTC
Server: Example RPP server v1.0
Content-Language: en
RPP-code: 01000
RPP-Queue-Size: 0
RPP-Svtrid: XYZ-12345
RPP-Cltrid: ABC-12345
Content-Length: 145

TODO
```

13.5. Create Resource

* Request: POST {collection}
* Request message: Object Create request
* Response message: Object Create response

The client MUST use the HTTP POST method to create a new object resource. If the RPP request results in a newly created object, then the server MUST return HTTP status code 200 (OK). The server MUST add the "Location" header to the response, the value of this header MUST be the URL for the newly created resource.

Example Domain Create request:

```
POST domains HTTP/2
Host: rpp.example
Authorization: Bearer <token>
Accept: application/rpp+json
Content-Type: application/rpp+json
Accept-Language: en
Content-Length: 220

TODO
```

Example Domain Create response:

```
HTTP/2 200
Date: Wed, 24 Jan 2024 12:00:00 UTC
Server: Example RPP server v1.0
Content-Language: en
Content-Length: 642
Content-Type: application/rpp+json
Location: https://rpp.example/domains/foo.example
RPP-code: 01000

TODO
```

13.6. Delete Resource

* Request: DELETE {collection}/{id}
* Request message: Optional
* Response message: Status

The client MUST the HTTP DELETE method and a resource identifying a unique object instance. The server MUST return HTTP status code 200 (OK) if the resource was deleted successfully.

Example Domain Delete request:

```
DELETE domains/foo.example HTTP/2
Host: rpp.example
Authorization: Bearer <token>
Accept: application/rpp+json
Accept-Language: en
RPP-Cltrid: ABC-12345
```

Example Domain Delete response:

```
HTTP/2 200 OK
Date: Wed, 24 Jan 2024 12:00:00 UTC
Server: Example RPP server v1.0
Content-Length: 80
RPP-Svtrid: XYZ-12345
RPP-Cltrid: ABC-12345
RPP-code: 01000
```

TODO

13.7. Processes Path Segment

Each provisioning object may be related to one or more running processes, such as a transfer or deletion. Each process can have its own data, which is distinct from the data of the provisioning object itself. The processes can be started, stopped, or interacted with using their own specific set of representations and operations.

All processes related to a provisioning object in RPP MUST exist under the /{collection}/{id}/processes/{process_name} path.

The server operator MAY support direct access to process resources using server generated identifier. Such resource MAY be accessible using following URL: /{collection}/{id}/processes/{process_name}/{process_id}, where process_id is the process identifier.

A process MAY also expose a resource at /{collection}/{id}/processes/{process_name}/latest to access and interact with the latest process instance. In case server offers any access to process information of given process name, the access to the last instance using /{collection}/{id}/processes/{process_name}/latest URL is MANDATORY.

The server operator MAY decide which processes such resources exist for, whether they only exist for the currently running processes or also for completed or cancelled processes. The period for which completed processes remain available for retrieval is defined by server policy.

13.7.1. Generic proces interface

A generic interface for interacting with the processes is defined as follows:

13.7.1.1. Starting:

POST /{collection}/{id}/processes/{process_name}

The payload of such a request contains process-specific input information. A started process MAY create a resource to access and interact with the process instance. In such case the response MUST be a 201 Created with a Location header pointing to the created resource together with the process state representation. The created resource can be made accessible both using the latest mnemonic under a URL /{collection}/{id}/processes/{process_name}/latest or using a process id under a URL /{collection}/{id}/processes/{process_name}/{process_id}.

When a process is created, executed and immediately completed by the server, a 201 Created response MAY still be provided together with the representation of the process result.

Server MAY decide not to expose any resource for interaction with the created process, in such case a 200 OK MUST be provided.

Example:

```
POST /rpp/v1/domains/foo.example/processes/renewals HTTP/2
... other headers removed for bravity ...

{
    "duration": "P2Y"
}
```

13.7.1.2. Cancelling:

A client MAY use the "latest" mnemonic to cancel the latest process instance, in such case the request MUST be:

DELETE /{collection}/{id}/processes/{process_name}/latest

If the client wants to cancel a specific process instance, the request MUST be:

DELETE /{collection}/{id}/processes/{process_name}/{process_id}

This request is intended to stop the running process. The server MUST return a 204 response if the process has been stopped and the resource is gone, or a 200 response if the process has been stopped but the resource remains.

13.7.1.3. Status

A client MAY use the "latest" mnemonic to request the latest process instance, in such case the request MUST be:

GET /{collection}/{id}/processes/{process_name}/latest

If the client wants to retrieve data of a specific process instance, the request MUST be:

GET /{collection}/{id}/processes/{process_name}/{process_id}

The request retrieves the representation of the task status. If no task is running, the server MAY return the status of the completed task or return a 404 response.

13.7.1.4. Other operations

Other operations on a process can be performed by adding path segments to the /{collection}/{id}/processes/{process_name}/latest or /{collection}/{id}/processes/{process_name}/{process_id} URL path.

13.7.1.5. Listing

A server MAY implement a listing facility for some or all, current or past processes.

The following URL structure and HTTP method MAY be exposed by the server and MUST be used by the client to retrieve process list filtered by process name:

GET /{collection}/{id}/processes/{process_name}/

The following URL structure and HTTP method MAY be exposed by the server and MUST be used by the client to retrieve full process list independent of the process name:

GET /{collection}/{id}/processes/

It is up to server policy to define the type of processes and state, running or completed, made available for the client. A server MAY also choose not implement this end point at all returning either the HTPP status code 404 Not Found or a 501 Not Implemented status code.

13.7.2. Relation to object representation

In certain situations a resource creation may require additional process data or implicitly start an asynchronous process with own inputs, lifecycle and state. In these cases, the representation sent to the server MAY contain a combination of object data and process-related data. For example a domain create request contains domain representation data which will be stored with domain object, and domain creation process data such as registration duration or price, which would be part as registration process data, but not directly stored with the domain object.

For the process data in the message body to be distinct and consistent with the URL path structure, it MUST be enclosed in the processes/{process_name} JSON path when transmitted with the object's representation.

Structure:

```
POST /.../{collection}/{id}
...
{
    ... object data ...
    "processes": {
        "{process_name}": {
            ... process data ...
        }
    }
    ...
}
```

Example: Domain Create request with 2-year registration:

```
POST /rpp/v1/domains HTTP/2
Host: rpp.example
Authorization: Bearer <token>
Accept: application/rpp+json
Content-Type: application/rpp+json
Accept-Language: en
Content-Length: 220

{
    "name": "foo.example",
    "processes": {
        "creation": {
            "periods": "P2Y"
        }
    }
    ... other domain data ...
}
```

13.8. Renew Resource

* Request: POST /{collection}/{id}/processes/renewals
* Request message: Renew request
* Response message: Renew response

Not every object resource includes support for the renew command. The response MUST include the Location header for the created renewal process resource.

Example Domain Renew request:

```
POST /rpp/v1/domains/foo.example/processes/renewals HTTP/2
Host: rpp.example
Authorization: Bearer <token>
Accept: application/rpp+json
Content-Type: application/rpp+json
RPP-Cltrid: ABC-12345
Accept-Language: en
Content-Length: 210

TODO: add renew request data here
```

Example Renew response:

```
HTTP/2 200 OK
Date: Wed, 24 Jan 2024 12:00:00 UTC
Server: Example RPP server v1.0
Content-Language: en
RPP-Svtrid: XYZ-12345
RPP-Cltrid: ABC-12345
Content-Length: 205
Location: https://rpp.example/rpp/v1/domains/foo.example/processes/renewals/XYZ-12345
Content-Type: application/rpp+json
RPP-code: 01000

TODO add renew response data here
```

13.9. Transfer Resource

The Transfer command is mapped to a nested resource, named "transfer". The semantics of the HTTP DELETE method are determined by the role of the client executing the DELETE method. The DELETE method is defined as "reject transfer" for the current sponsoring client of the object. For the new sponsoring client the DELETE method is defined as "cancel transfer".

13.9.1. Start

* Request: POST /{collection}/{id}/processes/transfers
* Request message: Optional
* Response message: Status

In order to initiate a new object transfer process, the client MUST use the HTTP POST method on a unique resource to create a new transfer resource object. Not all RPP objects support the Transfer command.

If the transfer request is successful, then the response MUST include the Location header for the object being transferred.

Example request not using object authorization:

```
POST /rpp/v1/domains/foo.example/processes/transfers HTTP/2
Host: rpp.example
Authorization: Bearer <token>
Accept: application/rpp+json
Accept-Language: en
RPP-Cltrid: ABC-12345
Content-Length: 0
```

Example request using object authorization:

```
POST /rpp/v1/domains/foo.example/processes/transfers HTTP/2
Host: rpp.example
Authorization: Bearer <token>
Accept: application/rpp+json
RPP-Cltrid: ABC-12345
RPP-Authorization: authinfo value=TXkgU2VjcmV0IFRva2Vu
Accept-Language: en
Content-Length: 0
```

Example request using 1 year renewal period, using the unit and value query parameters:

```
POST /rpp/v1/domains/foo.example/processes/transfers HTTP/2
Host: rpp.example
Authorization: Bearer <token>
Accept: application/rpp+json
Accept-Language: en
RPP-Cltrid: ABC-12345
Content-Length: 23

{
  "duration": "P2Y"
}
```

Example Transfer response:

```
HTTP/2 200 OK
Date: Wed, 24 Jan 2024 12:00:00 UTC
Server: Example RPP server v1.0
Content-Language: en
Content-Length: 182
Content-Type: application/rpp+json
Location: https://rpp.example/rpp/v1/domains/foo.example/processes/transfers/latest
RPP-code: 01001

{
  "trStatus": "pending",
  "reID": "ClientX",
  "acID": "ClientY",
  "reDate": "2000-06-06T22:00:00.0Z",
  "acDate": "2000-06-11T22:00:00.0Z",
  "exDate": "2002-09-08T22:00:00.0Z
}
```

13.9.2. Status

A transfer object may not exist, when no transfer has been initiated for the specified object. The client MUST use the HTTP GET method and MUST NOT add content to the HTTP message body.

* Request: GET {collection}/{id}/processes/transfers
* Request message: Optional
* Response message: Transfer Status response

Example domain name Transfer Status request without authorization information required:

```
GET /rpp/v1/domains/foo.example/processes/transfers HTTP/2
Host: rpp.example
Authorization: Bearer <token>
Accept: application/rpp+json
Accept-Language: en
RPP-Cltrid: ABC-12345
```

If the requested transfer object has associated authorization information that is not linked to another database object, then the HTTP GET method MUST be used and the authorization information MUST be included using the RPP-Authorization header.

Example domain name Transfer Query request using RPP-Authorization header:

```
GET /rpp/v1/domains/foo.example/processes/transfers HTTP/2
Host: rpp.example
Authorization: Bearer <token>
Accept: application/rpp+json
Accept-Language: en
RPP-Cltrid: ABC-12345
RPP-Authorization: authinfo value=TXkgU2VjcmV0IFRva2Vu
```

If the requested object has associated authorization information linked to another database object, then the HTTP GET method MUST be used and the RPP-Authorization header MUST be included.

Example domain name Transfer Query request and authorization using RPP-Authorization header:

```
GET /rpp/v1/domains/foo.example/processes/transfers HTTP/2
Host: rpp.example
Authorization: Bearer <token>
Accept: application/rpp+json
Accept-Language: en
RPP-Authorization: authinfo value=TXkgU2VjcmV0IFRva2Vu
Content-Length: 0
```

Example Transfer Query response:

```
HTTP/2 200 OK
Date: Wed, 24 Jan 2024 12:00:00 UTC
Server: Example RPP server v1.0
Content-Length: 230
Content-Type: application/rpp+json
Content-Language: en
RPP-code: 01000
```

TODO

13.9.3. Cancel

* Request: POST /{collection}/{id}/processes/transfers/cancelation
* Request message: Optional
* Response message: Status

The new sponsoring client MUST use the HTTP POST method to cancel a requested transfer.

Example request:

```
POST /rpp/v1/domains/foo.example/processes/transfers/cancelation HTTP/2
Host: rpp.example
Authorization: Bearer <token>
Accept: application/rpp+json
Accept-Language: en
RPP-Cltrid: ABC-12345
```

Example response:

```
HTTP/2 200 OK
Date: Wed, 24 Jan 2024 12:00:00 UTC
Server: Example RPP server v1.0
Content-Length: 80
RPP-Svtrid: XYZ-12345
RPP-Cltrid: ABC-12345
RPP-code: 01000

TODO
```

13.9.4. Reject

* Request: POST /{collection}/{id}/processes/transfers/rejection
* Request message: None
* Response message: Status

The currently sponsoring client of the object MUST use the HTTP POST method to reject a started transfer process.

Example request:

```
POST /rpp/v1/domains/foo.example/processes/transfers/rejection HTTP/2
Host: rpp.example
Authorization: Bearer <token>
Accept: application/rpp+json
Accept-Language: en
RPP-Cltrid: ABC-12345
```

Example Reject response:

```
HTTP/2 200 OK
Date: Wed, 24 Jan 2024 12:00:00 UTC
Server: Example RPP server v1.0
Content-Length: 80
RPP-Svtrid: XYZ-12345
RPP-Cltrid: ABC-12345
RPP-code: 01000

TODO
```

13.9.5. Approve

* Request: POST /{collection}/{id}/processes/transfers/approval
* Request message: Optional
* Response message: Status

The currently sponsoring client MUST use the HTTP POST method to approve a transfer requested by the new sponsoring client.

Example Approve request:

```
POST /rpp/v1/domains/foo.example/processes/transfers/approval HTTP/2
Host: rpp.example
Authorization: Bearer <token>
Accept: application/rpp+json
Accept-Language: en
RPP-Cltrid: ABC-12345
Content-Length: 0
```

Example Approve response:

```
HTTP/2 200 OK
Date: Wed, 24 Jan 2024 12:00:00 UTC
Server: Example RPP server v1.0
Content-Length: 80
RPP-Svtrid: XYZ-12345
RPP-Cltrid: ABC-12345
RPP-code: 01000
```

TODO

13.10. Update Resource

* Request: PATCH {collection}/{id}
* Request message: Object Update message
* Response message: Status

An object Update request MUST be performed using the HTTP PATCH method. The request message body MUST contain an Update message.

TODO: when using JSON, also allow for JSON patch so client can send partial update data only?

Example request:

```
PATCH domains/foo.example HTTP/2
Host: rpp.example
Authorization: Bearer <token>
Accept: application/rpp+json
Content-Type: application/rpp+json
Accept-Language: en
Content-Length: 252
```

TODO

Example response:

```
HTTP/2 200 OK
Date: Wed, 24 Jan 2024 12:00:00 UTC
Server: Example RPP server v1.0
Content-Length: 80
RPP-Svtrid: XYZ-12345
RPP-Cltrid: ABC-12345
RPP-code: 01000
```

TODO

14. Extension Framework

TODO

15. RPP Result Codes

RPP result codes are used to indicate the result of an RPP request. They are returned in the RPP-Code header of the HTTP response. The format of the RPP result code is a 5-digit string, where the first digit MUST always be "1", the second digit indicates the class of the result, and the remaining four digits indicate the specific result within that class, his allows implementers to define more specific result codes within each class. The classes of RPP result codes are designed to match the classes of HTTP status codes, to facilitate mapping between RPP result codes and HTTP status codes. The classes of RPP result codes are defined as follows:

* 11xxx: Informational
* 12xxx: Success
* 13xxx: Reserved for future use
* 14xxx: Client error
* 15xxx: Server error

The following RPP result codes are defined and used in this document:

RPP Result Code	HTTP Status Code	Description
12000	200 OK	Command completed successfully
12001	201 Created	Command completed successfully and a new resource was created

Table 2

16. Authentication and Authorization

Due to the stateless nature of RPP, the client MUST include the authentication credentials in each HTTP request. This MAY be done by using JSON Web Tokens (JWT) [RFC7519] or Basic authentication [RFC7617]. The server MUST validate the authentication credentials on each request and reject any request with invalid credentials with an appropriate HTTP status code.

When using JWTs for OAuth 2.0 [RFC6749] Access Tokens, the JWT profile described in [RFC9068] MUST be used. It is RECOMMENDED to use short-lived tokens and to implement token revocation mechanisms to mitigate the risk of token compromise. If sensitive information is included in the JWT payload, it MUST be encrypted to prevent unauthorized access when the token is persistent to a storage device. Furthermore, the best practices for JWT usage as outlined in [RFC8725] MUST be followed.

17. IANA Considerations

17.1. URN Sub-namespace for RPP (urn:ietf:params:rpp)

The IANA is requested to add the following value to the "IETF URN Sub-namespace for Registered Protocol Parameter Identifiers" registry, following the template in [RFC3553]: TODO: add filled in template, if we decide to use URN for profile identification, for example "urn:ietf:params:rpp:profile:example-profile"

Registry name: rpp
Specification: This Document
Repository: ?
Index value: ?

17.2. RPP registry group

The IANA is requested to create a new registry group for RPP, this will be used to group together all the RPP-related registries such as those for discovery URLs, extensions and profiles.

Name of the registry: RESTful Provisioning Protocol (RPP)
Reference: This Document
IANA Registry Reference: TODO

17.3. RPP Discovery registry

The IANA is requested to create a new registry for RPP discovery URLs, this registry will be used to register the well-known URLs for RPP discovery endpoints, used by RPP clients to discover the capabilities of a RPP server.

Name of the registry: RPP Discovery URLs
Registry group: RESTful Provisioning Protocol (RPP)
Registration procedure: Expert Review

Fields to be registered:

* tld: The top-level domain (TLD) for which the discovery URL is applicable, for example "example".
* url: The URL for the discovery endpoint, for example "https://rpp.example/.well-known/rpp" (https://rpp.example/.well-known/rpp").
* description: A human-readable description of the discovery URL and its intended use.

17.4. RPP Extension registry

The IANA is requested to create a new registry for RPP extensions, this registry will be used to register standardized extensions to the RPP protocol. Extensions are defined as additional features or capabilities that can be added to the base RPP protocol, for example support for additional resource types, additional operations or additional authentication methods.

Name of the registry: RPP Extensions
Registry group: RESTful Provisioning Protocol (RPP)
Registration procedure: Expert Review

Fields to be registered:

* name: The name of the extension, for example "RPP example extension".
* version: The version of the extension, for example "1.0".
* url: The URL for the extension specification, for example "https://www.iana.org/assignments/rpp-extensions/rpp-example-extension-1.0" (https://www.iana.org/assignments/rpp-extensions/rpp-example-extension-1.0").
* description: A human-readable description of the extension and its intended use.

17.5. RPP Profile registry

The IANA is requested to create a new registry for RPP profiles, this registry will be used to register standardized profiles for the RPP protocol. Profiles are defined as specific configurations of the RPP protocol that are designed to meet the needs of specific use cases or environments, for example an EPP compatibility profile that defines a set of RPP features and behaviors that are compatible with the Extensible Provisioning Protocol (EPP) [RFC5730].

Name of the registry: RPP Profiles
Registry group: RESTful Provisioning Protocol (RPP)
Registration procedure: Expert Review

Fields to be registered:

* name: The name of the profile, for example "EPP compatibility profile".
* id: A unique URN identifier for the profile, for example "urn:ietf:params:rpp:profile:epp-compatibility-1.0".
* version: The version of the profile, for example "1.0".
* url: The URL for the profile specification, for example "https://www.iana.org/assignments/rpp-profiles/epp-compatibility-provisioning-profile-1.0" (https://www.iana.org/assignments/rpp-profiles/epp-compatibility-provisioning-profile-1.0").
* description: A human-readable description of the profile and its intended use.

17.6. RPP Result Codes Registry

The IANA is requested to create a new registry "RPP Result codes", this registry will be used to register RPP result codes defined in this document and in future RPP specifications and extensions.

Name of the registry: RPP Result codes
Registry group: RESTful Provisioning Protocol (RPP)
Registration procedure: Expert Review

Fields to be registered:

* code: The RPP result code, for example "12000".
* description: A human-readable description of the result code and its intended use.

17.7. RPP Media Type (application/rpp+json)

The IANA is requested to add the following RPP media type to the "Media Types" registry, following the template in [RFC6838]:

Type name: application
Subtype name: rpp+json
Required parameters: version
Optional parameters: "N/A"
Encoding considerations: "N/A"
Security considerations: "N/A"
Interoperability considerations: "N/A"
Published specification: This document
Applications that use this media type: RPP protocol and extensions
Fragment identifier considerations: "N/A"
Additional information: "N/A"
Person & email address to contact for further information: Author’s email address
Intended usage: COMMON
Restrictions on usage: "N/A"
Author: Document authors
Change controller: Document authors
Provisional registration: No

18. Internationalization Considerations

TODO

19. Security Considerations

RPP relies on the security of the underlying HTTP transport, hence the best common practices for securing HTTP described in [RFC9325] also apply to RPP and MUST be followed by RPP implementations.

Data confidentiality and integrity MUST be enforced. Every client and server interaction MUST be encrypted using TLS version 1.3 [RFC8446]. Future versions of TLS MAY be used as they become available and are deemed secure.

20. Change History

20.1. Version 04 to 05

* Added Boostrap and Discovery sections to the document, describing how a client can discover the location and capabilities of an RPP server
* Added IANA Considerations section with a request for new RPP discovery URLs, extensions and profile URLs.

20.2. Version 03 to 04

* Added a new section on versioning, describing how versioning is applied to different RPP elements. (Issue #39)

* Added IANA request for RPP Result codes registry, and added a table with example RPP result codes. (Issue #39)
* Added IANA request for URN sub-namespace for RPP. (Issue #39)
* Added a new "Profiles" section to describe how to define and use profiles. (Issue #43)
* Added a new section "Authentication and Authorization". (Issue #37)
* Updated the "Security Considerations" section to include transport security. (Issue #37)
* Added IANA registration request for the new RPP media type. (Issue #40)

20.3. Version 02 to 03

* Added use of Problem Detail [RFC9457] for error responses

20.4. Version 01 to 02

* Updated the examples, changed from "_.example.org" to "_.example"
* Merged the RPP-EPP-Code and RPP-Code headers into a single RPP-Code header
* Update the RPP-Authorization header to match the HTTP Authorization header format
* Added new process path segment and process representations
* Updated the Check request to now use an "availability" path segment and support both GET and HEAD methods

20.5. Version 00 to 01

* Updated "Request Headers" and "Response Headers" section
* Changed transfer resource URL and HTTP method for reject, approve and cancel, in order to make the API easier to use

20.6. Version 00 (draft-rpp-core) to 00 (draft-wullink-rpp-core)

* Renamed the document name to "draft-wullink-rpp-core"
* Removed sections: Design Considerations, Resource Naming Convention, Session Management, HTTP Layer, Content Negotiation, Object Filtering, Error Handling
* Renamed Commands section to Endpoints
* Removed text about extensions
* Changed naming to be less EPP like and more RDAP like

21. Acknowledgements

The authors would like to thank the following people for their helpful text contributions, comments and suggestions.

* Q Misell, AS207960 Cyfyngedig

22. References

22.1. Normative References

[REST] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", 2000, <http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm>.

[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/info/rfc2119>.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, DOI 10.17487/RFC2616, June 1999, <https://www.rfc-editor.org/info/rfc2616>.

[RFC2782] Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, DOI 10.17487/RFC2782, February 2000, <https://www.rfc-editor.org/info/rfc2782>.

[RFC3553] Mealling, M., Masinter, L., Hardie, T., and G. Klyne, "An IETF URN Sub-namespace for Registered Protocol Parameters", BCP 73, RFC 3553, DOI 10.17487/RFC3553, June 2003, <https://www.rfc-editor.org/info/rfc3553>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

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

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

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

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

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

[RFC6648] Saint-Andre, P., Crocker, D., and M. Nottingham, "Deprecating the "X-" Prefix and Similar Constructs in Application Protocols", BCP 178, RFC 6648, DOI 10.17487/RFC6648, June 2012, <https://www.rfc-editor.org/info/rfc6648>.

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

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <https://www.rfc-editor.org/info/rfc6838>.

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

[RFC7617] Reschke, J., "The 'Basic' HTTP Authentication Scheme", RFC 7617, DOI 10.17487/RFC7617, September 2015, <https://www.rfc-editor.org/info/rfc7617>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[RFC8725] Sheffer, Y., Hardt, D., and M. Jones, "JSON Web Token Best Current Practices", BCP 225, RFC 8725, DOI 10.17487/RFC8725, February 2020, <https://www.rfc-editor.org/info/rfc8725>.

[RFC8941] Nottingham, M. and P. Kamp, "Structured Field Values for HTTP", RFC 8941, DOI 10.17487/RFC8941, February 2021, <https://www.rfc-editor.org/info/rfc8941>.

[RFC9068] Bertocci, V., "JSON Web Token (JWT) Profile for OAuth 2.0 Access Tokens", RFC 9068, DOI 10.17487/RFC9068, October 2021, <https://www.rfc-editor.org/info/rfc9068>.

[RFC9110] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/info/rfc9110>.

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

[RFC9325] Sheffer, Y., Saint-Andre, P., and T. Fossati, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 9325, DOI 10.17487/RFC9325, November 2022, <https://www.rfc-editor.org/info/rfc9325>.

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

[RFC9535] G旦ssner, S., Ed., Normington, G., Ed., and C. Bormann, Ed., "JSONPath: Query Expressions for JSON", RFC 9535, DOI 10.17487/RFC9535, February 2024, <https://www.rfc-editor.org/info/rfc9535>.

[SemVer] Semantic Versioning, "Semantic Versioning 2.0.0", <https://semver.org/>.

22.2. Informative References

[RFC8748] Carney, R., Brown, G., and J. Frakes, "Registry Fee Extension for the Extensible Provisioning Protocol (EPP)", RFC 8748, DOI 10.17487/RFC8748, March 2020, <https://www.rfc-editor.org/info/rfc8748>.

Authors' Addresses

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

Pawel Kowalik
DENIC

Email: pawel.kowalik@denic.de
URI: https://denic.de/