
Workgroup:	Network Working Group
Internet-Draft:	draft-wullink-rpp-requirements-00
Published:	28 April 2025
Intended Status:	Standards Track
Expires:	30 October 2025
Authors:	M. Wullink P. Kowalik
	<i>SIDN Labs DENIC</i>

RESTful Provisioning Protocol (RPP) - Requirements

Abstract

This document describes the requirement for the development of the RESTful Provisioning Protocol (RPP).

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 30 October 2025.

Copyright Notice

Copyright (c) 2025 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction	3
2. Terminology	3
3. Conventions Used in This Document	3
4. General	3
5. HTTP	4
6. REST	4
7. Data Model	5
8. Data Representation	5
9. Operations and responses	6
10. Discoverability	6
11. EPP compatibility	8
12. Security	8
13. Extensibility	9
14. Scalability	10
15. Performance	11
16. Internationalisation	11
17. Clients	11
18. Requirements for object types	12
18.1. Domain Object Type	12
18.2. Host Object Type	12
18.3. Contact Object Type	12
18.3.1. Data Representation	12
19. New features	12
20. IANA Considerations	12
21. Internationalization Considerations	12
22. Security Considerations	12
23. Appendix A. Extensions	12
24. Normative References	13

1. Introduction

This document describes the set of requirements for the RESTful Provisioning Protocol (RPP), an Application Programming Interface (API) for provisioning objects in a shared database. RPP is based on the HTTP [\[RFC9110\]](#) protocol and the architectural principles of [\[REST\]](#).

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.

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 [\[RFC2119\]](#).

4. General

R1.1. A well defined architecture MUST be defined for RPP, including a description of the responsibilities of the defined protocol layers.

R1.2. RPP MUST provide a clear, clean, easy to use and self-explanatory interface that can easily be integrated into existing software systems.

//TODO: [Issue #3](#)

R1.3. Wherever applicable RPP SHOULD leverage existing best practices and well adopted standards for building and documenting RESTful APIs. There MUST be a clear justification when deviating from this.

R1.4. RPP MUST include support for application level status codes, and MAY reuse the EPP status codes defined in [\[RFC5730\]](#).

R1.5. RPP MAY include support for providing detailed information about application status codes, for example as described in [\[RFC7807\]](#)

R1.6 RPP SHOULD support additional information about a successful operation (information or warning) to convey additional information to the client for example about deprecation or partial success.

5. HTTP

R2.1. The Hypertext Transfer Protocol (HTTP) [\[RFC9110\]](#) MUST be used as the transport mechanism for RPP.

R2.2. RPP SHOULD use the best common practices for designing HTTP based applications, described in [\[BCP56\]](#). There MUST be a clear justification when deviating from this.

R2.3. Consistent, predictable and meaningful URL structures MUST be used for for identifying, accessing object resources and enable request routing.

R2.4. RPP MUST use the existing HTTP status codes and MUST define application level status codes and map these to HTTP status codes. RPP MUST NOT redefine existing HTTP status code semantics and when overloading (generic) HTTP status codes with multiple RPP status codes, the provided RPP status code MUST be used by the client to determine the exact nature of the problem.

6. REST

R3.1. The RPP architecture MUST use the principles of the [\[REST\]](#) architectural style. A RPP server MUST conform to at least level 2 of the [\[RICHARDSON\]](#) Maturity Model (RMM).

R3.2 The RPP architecture MUST follow Resource-Oriented Architecture [\[ROI\]](#).

R3.3. The RPP specification MUST strive to minimise round trips between client and server. Approaches, where client would need to make multiple requests each time to discover resource URL or server capabilities in order to perform operation SHOULD be used sparingly and be always well justified.

R3.4. *Merged with R12.1*

R3.5. RPP specifications SHOULD incorporate a machine-readable and well-established API specification, such as [!@OpenAPI] or [RAML]. This will facilitate documentation, testing, code generation, and user-friendly extension descriptions. RPP MUST NOT require what API specification technology is to be used. The RPP core documents and extension documents may also choose different API specification solutions, this choice is left to the document authors.

7. Data Model

R4.1 The base data model structures MUST be data format agnostic. It MUST be possible to map the base data model to multiple data formats such as JSON, XML or YAML.

R4.2 Commonly used EPP extensions SHOULD be added to the RPP core data model. An example of this is the DNSSEC extension.

R4.3 RPP MUST allow an extension mechanism that allows clients to signal data omission or redaction, indicating data collected but not transmitted to the registry or redacted.

R4.4 RPP MUST have mechanisms to define profiles to indicate:

- Required parts of the data model
- Mapping definition
- Functional subsets for compatibility.

//TODO: [Issue #15](#)

R4.5 The RPP architecture MUST include loose coupling between the server and the client, allowing for non-coordinated introduction of non-breaking version changes on both sides.

R4.6 A RPP server and client MUST in default ignore unknown properties of representations however there MUST be a mechanism for a client to signal that a strict handling is wished where unknown fields are treated as error.

8. Data Representation

R5.1 RPP MUST use JSON as the default data format.

R5.2 It MUST be possible to extended RPP to include support other data formats (e.g. XML, YAML).

R5.3 Validation of request and response message MUST be supported for both clients and the servers, in order to determine if the content is valid and no required attributes are missing.

R5.4 RPP MUST define a default media type however the protocol SHALL be extensible to enable support for other media types.

R5.5 A client **MUST** be able to signal to the server what media type the server should expect for the request content and to use for the response content.

//removed: see <https://github.com/ietf-wg-rpp/rpp-requirements/issues/11> // **R5.6** RPP **MUST** support the use of server profiles to define required components of the data model and/or mapping definitions (see: R4.4).

R5.7 RPP **SHOULD** consider mechanisms for supporting data formats outside of core RPP domain. Especially formats, which lose their properties if transformed, like Verifiable Credentials for contacts which are digitally signed.

R5.8 RPP **MUST** support partial update of data objects.

R5.9 RPP **MUST** support full update of data objects.

R5.10 A generated RPP response representation that includes an object identifier (for example a contact handle) **MUST** also include a URL reference to the location of the object representation.

9. Operations and responses

R6.1 RPP **MUST** include support for a client requesting different depth of data representations, depending on the use case:

- Minimal representation (ID, or ID+name)
- Full representation (all data of the object)
- Full representation + dereferenced referrals (for example domain with contact and host details)

R6.2 RPP **MAY** return different representations of the same object in different contexts:

- GET request to the resource itself
- GET request to get a collection of objects
- Responses to PUT/POST/PATCH requests

R6.3 The data representation in a RPP response **MUST** only contain data related to the object, transactional information **MUST** be represented as one or more separate HTTP headers.

10. Discoverability

R7.1 RPP **MAY** include a bootstrap mechanism designed to allow clients to locate the network identifier for the RPP service of a registry operator, e.g. rpp.sidn.nl for the registry operator for the .nl ccTLD.

Solutions may include:

- IANA bootstrap Service Registry
- DNS TXT records

R7.2 An RPP server **MUST** publish a service discovery document in the well-known directory, described in [RFC5785]. This document contains structured machine readable information that required or useful for the client to be able to generate valid RPP requests. The information may contain, but is not limited to:

- Available services,
- Used Extensions
- Versions used for services and extensions
- Environment name (production, test etc.)
- Server datetime
- Maintenance notices
- Supported profiles
- ...

//TODO: [Issue #8](#)

R7.3 Server provided functionality, such as the set of supported profiles, languages or extensions, **MUST** discoverable using the discovery document.

R7.4 RPP **MUST** support versioning of:

- The protocol itself
- Data object types
- Representations
- Operations
- Profiles
- Extensions

R7.5 Versioning schema **MUST** carry information about breaking vs. non-breaking changes and allow clients to decide whether it is able to interact with the server. The versioning scheme **SHOULD** be like the scheme used for HTTP where minor version changes do not break compatibility.

R7.5 Versions used by the RPP protocol and used extensions **MUST** be discoverable by the client.

R7.6 Notices related to scheduled server maintenance timeslots **MAY** be included in the discovery document, this could be a human readable, non machine parsable character string.

//TODO: [Issue #9](#)

R7.7 RPP **MAY** only support a subset of EPP functionality, the supported functionality **MUST** be discoverable by the client

R7.8 Removed

//SEE: [Issue #21](#)

R7.9 An RPP response that includes unique object identifiers, MAY also include URL references for these objects.

11. EPP compatibility

R8.1 RPP MUST provide functional equivalents for core EPP functionalities related to domain name, host, and contact objects as defined in [\[RFC5731\]](#), [\[RFC5732\]](#) and [\[RFC5733\]](#).

R8.2 The automatic or mechanical mapping or conversion between EPP and RPP data model MUST be possible.

R8.3 Compatibility definitions for a RPP to EPP mapping MAY be defined in compatibility profiles (see: R4.4).

//TODO: [Issue #15](#)

R8.4 RPP MUST include an extension framework able to define equivalents of most commonly used EPP extensions, which are not a part of core protocol (see: R4.2)

R8.5 EPP password based Authorization Information defined in [\[RFC5731\]](#) and [\[RFC5733\]](#) MUST be supported in RPP.

R8.6 RPP SHOULD support client_id/password authentication to match EPP client authentication.

12. Security

R9.1 RPP MUST support state-of-the-art authentication and authorization schemes allowing for easy integration in modern HTTP infrastructure.

R9.2 RPP MUST support modern authentication and authorization standards (OAuth, OpenId Connect)

R9.3 Support for an simplified and quicker object transfer process MAY be included, where approval from the losing registrar is to be obtained interactively by the registrant during the transfer process.

R9.4 RPP MUST include an authorisation model/framework that goes beyond the current EPP password based Authorization Information (AuthInfo) used for object transfers. The following use cases MAY be supported:

- Object transfers without using an EPP password based Authorization Information
- Registrants using OpenID Connect can interactively allow DNS operator to update their NS records, directly in the registry database or indirectly using a registrar.

R9.5 RPP MUST employ strong authentication and utilize encrypted transport (HTTPS) to protect sensitive data.

R9.6 Security mechanisms SHOULD be flexible to allow operators to choose appropriate methods and support federated authentication scenarios.

R9.7 RPP MAY include a mechanism for cryptographic verification of request and response messages as an additional security layer.

R9.8 RPP MUST allowing for multiple user accounts linked to a single registrar, registrar user management MAY be delegated to an administrator account linked to a registrar, allowing for self service account management by the registrar.

R9.9 RPP MUST support a granular authorization matrix, where one or more permissions are coupled to a user account. Allowing for the creation of different types of user accounts, such as readonly users only allowed to fetch data about existing objects, and power users allowed to create and modify objects.

R9.10 RPP MUST allow users to update their credentials and enforce strong passwords and limited lifetime for passwords and other tokens.

13. Extensibility

R10.1 The protocol MUST be extensible to accommodate new functionalities, data elements, and operations beyond the initial scope.

R10.2 RPP MUST allow for flexibility in extending the data model e.g. adding new objects or a new attribute to an existing object MUST be possible.

R10.3 RPP SHOULD promote standardisation of commonly used extension attributes.

R10.4 Extensions for new operations on existing resources MUST be supported.

R10.5 RPP MUST support extensions that define new status codes not already defined in the core RPP RFCs.

//TODO: [Issue #20](#)

R10.6 RPP MUST support extensions adding new HTTP headers.

R10.7 RPP SHALL have mechanisms to assure conflict avoidance when extending the protocol, including but not limited to data model, representations, operations, parameters, error codes and signalling. There MUST be a mechanism of conflict-free, non-coordinated extending in private/vendor discretion as well as a coordinated process for core, generic or shared elements.

//TODO: [Issue #10](#)

R10.8 When a public registry for RPP extensions is required, then IANA MUST be used for this function.

R10.9 RPP extensions MUST include support for versioning, the version of the extension supported by the server MUST be included in the discovery document.

//TODO: [Issue #11](#)

//dropped R10.10: see <https://github.com/ietf-wg-rpp/rpp-requirements/issues/20> // **R10.10** RPP status codes are maintained in an IANA registry for RPP status codes, these include all status codes defined in the core RPP RFCs and by any extension that is also a registered IANA RPP extension.

R10.11 Extension designers or RPP implementers MAY add new status codes, if a newly created status code is generic enough to be useful for the wider RPP community, then the extension designer SHOULD register the new status code in the RPP IANA registry.

14. Scalability

R11.1 RPP MUST be stateless and MUST NOT maintain application state on the server required for processing future RPP requests. Every client request needs to provide all the information required for the server to be able to successfully process the request. The client MAY maintain application session state, for example by using a JWT token.

R11.2 RPP MUST support cacheability of responses, if applicable to the operation semantics and MUST not include transaction related identifiers and values.

R11.3 RPP MUST support load balancing at the level of request messages (URL) and load balancing MUST be possible without processing HTTP body.

R11.4 Every request message MUST at most contain a single object for the server to operate on, with the exception of operations that are explicitly defined as a bulk operation.

R11.5 RPP MUST support asynchronous processing for operations on multiple objects, otherwise resource intensive or involving manual steps. The client request results in a confirmation of receipt and a means for retrieving the final completed processing result at a later time.

15. Performance

R12.1 In order to minimise message sizes and needed processing RPP SHOULD be designed not to include a HTTP message body in the request or response when this is not necessary, for example when the required data can be transmitted using the URL and/or HTTP headers.

R12.2 RPP MAY allow for common bulk operations, resource listing, and filtering capabilities. RPP MUST NOT mandate such functionalities where this may impact scalability or performance negatively.

R12.3 *Removed*

16. Internationalisation

R13.1 RPP MUST support internationalization, for object types and messages defined in the core protocol and extensions

//TODO: [Issue #23](#)

R13.2 RPP MUST support human-readable localized response messages.

17. Clients

R14.1 RPP MUST support server applications as clients. This will be a primary use-case of registry/registrar integration.

R14.2 RPP MUST support interaction from command-line tools or desktop applications capable of sending HTTP requests. These can be generic clients such as curl or Postman but also specialized RPP command line tools or scripts.

R14.3 RPP SHOULD support web browsers as clients, such as SPA (single page applications) without any proxy backend between webbrowser and the RPP server.

R14.4 RPP SHOULD support mobile applications as clients, also here through direct integration without any proxy backend.

18. Requirements for object types

18.1. Domain Object Type

18.2. Host Object Type

18.3. Contact Object Type

18.3.1. Data Representation

C5.1 RPP SHOULD consider using JSContact [RFC9553] format for contact representation.

19. New features

//MWU: is there a difference between optional features and an extension? i would think so, we can define an optional feature in the core protocol but make it optional. an extension is defined after the completion of the core protocol and is also optional.

//PK: would you see it as a part of generic protocol requirements or an extension

//MWU: using a "MAY" right now it is optional feature but not an extension The server MAY support generating a representation of a historical overview for an object, e.g. show all events linked to the object (create, update ...). The historical time window is determined by server policy and MUST be included in the discovery service document.

20. IANA Considerations

TODO

21. Internationalization Considerations

TODO

22. Security Considerations

TODO

23. Appendix A. Extensions

// List of required extions here // see: <https://github.com/ietf-wg-rpp/rpp-requirements/issues/19>

A.1 An extension for a Search API to allow for searching for objects in the registry database. Includes advanced search capabilities for object info request.

A.2 An extension that allows a DNS operator to update the DNSSEC key material for a domain object. This extension MAY be used by the DNS operator to update the DNSSEC key material for a domain object, without the need for the registrar to be involved in this process.

TODO

24. Normative References

- [BCP56]** Best Current Practice 56, <<https://www.rfc-editor.org/info/bcp56>>. At the time of writing, this BCP comprises the following:
- Nottingham, M., "Building Protocols with HTTP", BCP 56, RFC 9205, DOI 10.17487/RFC9205, June 2022, <<https://www.rfc-editor.org/info/rfc9205>>.
- [RAML]** raml.org, "RESTful API Modeling Language", 2025, <<https://raml.org/>>.
- [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>>.
- [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>>.
- [RFC5785]** Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, DOI 10.17487/RFC5785, April 2010, <<https://www.rfc-editor.org/info/rfc5785>>.

- [RFC7807] Nottingham, M. and E. Wilde, "Problem Details for HTTP APIs", RFC 7807, DOI 10.17487/RFC7807, March 2016, <<https://www.rfc-editor.org/info/rfc7807>>.
- [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>>.
- [RFC9553] Stepanek, R. and M. Loffredo, "JSContact: A JSON Representation of Contact Data", RFC 9553, DOI 10.17487/RFC9553, May 2024, <<https://www.rfc-editor.org/info/rfc9553>>.
- [RICHARDSON] Fowler, M., "Richardson Maturity Model", 2010, <<https://martinfowler.com/articles/richardsonMaturityModel.html>>.
- [ROI] Richardson, L. and S. Ruby, "RESTful Web Services, Chapter 4", 2007, <<https://www.oreilly.com/library/view/restful-web-services/9780596529260/ch04.html>>.

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/>