Workgroup:	Network Working Group
Internet-Draft:	draft-wullink-rpp-requirements-00
Published:	11 April 2025
Intended Status:	Standards Track
Expires:	13 October 2025
Authors:	M. Wullink *SIDN Labs* P. Kowalik *DENIC*

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 13 October 2025.

Copyright Notice

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 7
12. Security 8
13. Extensibility 9
14. Scalability 9
15. Performance 10
16. Internationalisation 10
17. Clients 10
18. New features 11
19. Other 11
20. IANA Considerations 12
21. Internationalization Considerations 12
22. Security Considerations 12
23. Normative References 12
Authors' Addresses 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 wel defined architecture MUST be defined for RPP, including a description of the responsibilities of the definded protocol layers.

R1.2. The API MUST provide a clear, clean, easy to use and self-explanatory interface that can easily be integrated into existing software systems and includes language bindings for the most popular programming languages. //PK: too vague //see: https://github.com/ietf-wg-rpp/rpp-requirements/issues/3

R1.3. RPP MUST leverage widely deployed web standards, tools, and infrastructure components such as HTTP, JSON, [!@OpenAPI] specification, API gateways, load balancing, caching and delegate functional responsibility to the HTTP layer when possible. For example, authentication is part of the HTTP layer and not part of the RPP application layer. //PK: too vague //see: https://github.com/ietf-wg-rpp/rpp-requirements/issues/4

R1.4. //MWU: having 2 layers of status codes, RPP and HTTP plus a mapping between the 2 adds complexity, need good reason for this. can we get away with only using HTTP status codes? if only using HTTP status codes, then R1.5 should be changed to a MUST? //Issue: https://github.com/ietf-wg-rpp/rpp-requirements/issues/5 RPP MUST include support for application level status codes, and MAY reuse the EPP status codes defined in [RFC5730]. RPP status codes MUST remain independent of HTTP status codes.

R1.5. RPP MAY include support for providing detailed information about application status codes, for example as descibed in [RFC7807]

5. HTTP

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

R2.2. RPP MUST 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 the existing HTTP status codes and MAY define application level status codes and map these to HTTP status codes. RPP MUST NOT redefine or overload existing HTTP status code semantics.

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].

//PK: this part is for me questionable. Especially with all the requirements R6.X. Full HAEOAS is likely not what we need (however we'd need to formulate a design requirement, which would support this call), but some level of hypermedia does not have to be necessarily wrong, like: links to other resources, signalling of available/permittable operations etc. //MWU: agree, HATEOS is prob not the way to go, but using link to other object in responses is something we probaly want. **R3.3.** The RPP specification MUST specify all resource URLs used and therefore the URLs are already known and do not need to be dynamically discoverable, designing for RMM level 3 is not recommended.

R3.4. RPP SHALL prefer simplicity. Wherever the whole operation can be defined by a resource URL and HTTP method and where HTTP method does not require a message body, RPP MUST NOT use a message body. This approach reduces complexity, improves performance, and aligns with the principles of RESTful design by leveraging the inherent semantics of HTTP methods.

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. //PK: I would keep it open in the requirements which technology it should be

7. Data Model

R.4.1 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.

R.4.2 Commonly used EPP extensions SHOULD be added to the RPP core data model, an example of this is the DNSSEC extension.

// PK: Extension? //MWU: not sure what this means? need better description? **R.4.3** 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.

//MWU: maybe better describe what a profile is and what it is not. **R.4.4** RPP MUST have mechanisms to define profiles to indicate:

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

R.4.5 The RPP architecture MUST include a loose coupling between the server and the client, allowing for adding non-breaking version changes on both sides.

R.4.6 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

R.5.1 RPP MUST use JSON as the default data format.

R.5.2 It MUST be able to extended RPP to include support other data formats (e.g. XML, YAML).

R.5.3 Validation of request and response message MUST be supported, in order to determine if the content is valid and no required attributes are missing.

R.5.4 RPP must define a default media type however the protocol shall be extensible to enable support for other media types.

R.5.5 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.

R.5.6 Allow for the use of server profiles, indicating required parts for the data model and/or mapping definitions.

R.5.7 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.

R.5.8 RPP MUST support partial update of data objects.

R.5.9 RPP MUST support full update of data objects.

R.5.10 Contact information MUST be provided using the JSContact [RFC9553] format. // PK: I think we should not have it as MUST requirement. R.4.1 is covering to support JSContact as additional format, but can we decide now whether this would be a default?

9. Operations and responses

R.6.1 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)

R.6.2 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

R.6.3 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

R.7.1 RPP MAY include a bootstrap mechanism for helping clients to locate RPP available services, possible solutions include:

- IANA bootstrap Service Registry
- DNS TXT records

R.7.2 A discovery document MUST be made available in the well-known directory. //PK: This is somehow contadictory to R.5.1. If it's well known then you don't need other signalling //MWU: my thinking here is thats not nor client signalling but for the server to indicate used language, extensions etc. Similar as is done in the EPP hello command response

R.7.3 Server provided functionality, such as the set of supported profiles, languages or extensions, MUST discoverable using the discovery document.

R.7.4 RPP MUST support versioning of:

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

R.7.5 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.

R.7.5 Versions used by the RPP protocol and used extensions MUST be discoverable by the client.

//PK: extension? This might get quite complex. //MWU: maybe not, this could be as simple as a "notices" field in the discovery document **R.7.6** 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.

R.7.7 A RPP service MAY choose to only support a subset of EPP functionality, this MUST be discoverable by the client.

R.7.8 A client MUST be able to generate the URL for an object, based on the unique object identifier.

R.7.9 An RPP response that includes unique object identifiers, MAY also include URL references for these objects.

11. EPP compatibility

//MWU: this contradicts R.6.7 that says a subset of EPP may be supported only? **R.8.1** RPP MUST provide functional equivalents for core EPP functionalities related to domain names, hosts, and contacts as defined in [RFC5731], [RFC5732] and [RFC5733] mappings for core objects (domain, contact, host).

R.8.2 The automatic or mechanical mapping or conversion between EPP and RPP data model MUST be possible.

//MWU: better describe what a compatibility profiles is **R.8.3** Compatibility definitions for a RPP to EPP mapping MAY be defined in compatibility profiles.

R.8.4 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: R.4.2)

//MWU: only support string token, all other token alternatives should be based on web based auth schemes? **R.8.5** Only the string based EPP token defined in [RFC5730] MUST be supported, EPP token extensions SHOULD NOT be supported.

R.8.6 RPP SHOULD support client_id/password authentication to match EPP client authentication.

12. Security

R.9.1 RPP MUST support state-of-the-art authentication and authorization schemes allowing for easy integration in modern HTTP infrastructure.

R.9.2 RPP MUST support modern authentication and authorization standards (OAuth, OpenId Connect)

R.9.3 Support for an simplified and quicker object transfer process MAY be included, where approval from the losing registar is to be obtained interactively by the registrant during the transfer process.

R.9.4 The authorisation model MUST support authorizations, beyond current auth-code based authorisation for transfers, the following use cases MAY be supported:

- Object transfers without using an EPP style transfertoken
- DNS providers updating the DNSSEC key material
- Registrants using OpenID Connect to interactively allow DNS providers to update NS records, directly at theregistry or indirectly through a supporting registar.
- Object Renewals

R.9.5 RPP MUST employ strong authentication and utilize encrypted transport (HTTPS) to protect sensitive data.

R.9.6 Security mechanisms SHOULD be flexible to allow operators to choose appropriate methods and support federated authentication scenarios.

R.9.7 RPP MAY include a mechanism for cryptographic verification of request and response messages as an additional security layer.

R.9.8 RPP MUST allowing for multiple user accounts linked to a single registrar, registar user management MAY be delegated to an administrator account linked to a registrar, allowing for self service account management by the registar.

R.9.9 RPP MUST support a granalar authorization matrix, where one or more permissions are coupled to a user account. Allowing for the creation of different types of user accounts, such a readonly users only allowed to fetch data about existing objects, and power users allowed to create and modify objects.

R.9.10 RPP MUST allow users to update their credentials and enforce strong passwords and limited lifetime for passwords and other tokens.

13. Extensibility

R.10.1 The protocol MUST be extensible to accommodate new functionalities, data elements, and operations beyond the initial scope.

R.10.2 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.

R.10.3 RPP SHOULD promote standardisation of commonly used extension attributes.

R.10.4 Extensions for new operations on existing resources MUST be supported.

//MWU: IANA registry for status/error codes? **R.10.5** RPP MUST support extensions adding new status codes.

R.10.6 RPP MUST support extensions adding new HTTP headers.

//PK: I would rather see a similar model as rfc6838 with simplified namespacing. Standard/ registered tree (no namespace) and vendor tree maybe with reversed domain name notation. //MWU using a domain name notation will create issues when creating programming language binding, for example using dots in a JSON key name causes problems when using Python. **R.10.7** 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 discresion as well as a coordinated process for core, generic or shared elements.

R.10.8 When a registry for RPP extensions is required, the IANA MUST be used for this function.

//MWU: only allow server to use 1 version of extension at the same time? otherwise client always needs to inlude ext version to use in the request **R.10.9** RPP extensions MUST include support for versioning, the version of the extention supported by the server MUST be included in the discovery document.

14. Scalability

R.11.1 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.

R.11.2 RPP MUST support cacheability of responses, if applicable to the operation semantics and MUST not include transaction related identifiers and values. //PK: what would that mean? I guess any representation MUST NOT include transaction related identifiers and values, not only the cachable ones. //MWU: agree

R.11.3 RPP MUST support load balancing at the level of request messages (URL) and load balancing MUST be possible without processing any HTTP payload.

R.11.4 Every request message MUST at most contain a single object for the server to operate on, with the exception of operations that are explicitely defined as a bulk operation. Bulk operations MAY be processed asynchronously. // PK Would asynchronous and synchronous processing be a separate requirement? // MWU: maybe the async processing could be optional?

//MWU: better descibe where asyc processing would be useful **R.11.5** RPP MAY support asynchronous processing for appropriate bulk or otherwise resource intensive operations. 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

R.12.1 RPP MUST NOT include a HTTP message body in the request or response when this is not necessary, for example when the required data can be tranmitted using the URL and/or HTTP headers.

R.12.2 RPP MAY allow for common bulk operations, resource listing, and filtering capabilities where this does not impact scalability negatively.

R.12.3 RPP MAY support compound object create request having embedded contact/host vs. request serialization (client waiting for contact/host creation to succeed before sending a domain request). Return complete representation (similar to object info in EPP) after compound request completed or return redirect to newly created object location. //PK: from the feedback I got so far, I would drop this requirement //MWU: going for more EPP style, creating required NS and contact objects in seperate request first, would be easier adding compound request would be lot more work. //MWU: maybe make it somehow easier to find and reuse existing objects? now we see lot of registrars creating new contacts each time even though its for the same registrant. Maybe include a contact search API?

16. Internationalisation

R.13.1 RPP MUST support internationalization, including but not limited to:

- Contact objects
- Email addresses
- Internationalized Domain Names (IDNs)

R.13.2 RPP MUST support Human-readable localized response mesages.

17. Clients

R.14.1 RPP MUST support server applications as clients. This will be a primary use-case of registry/registrar integration.

R.14.2 RPP MUST support interaction from command-line tools or desktop applications capable of sending HTTP requests. Whese can be generic clients such as curl or Postman but also specialized RPP command line tools or scripts.

R.14.3 RPP SHOULD support web browsers as clients, such as SPA (single page applications) without any proxy backend between webbrowser and the RPP server.

R.14.4 RPP SHOULD support mobile applications as clients, also here through direct integration without any proxy backend.

18. 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.

19. Other

The items below have been mentioned on the mailinglist and may need to be added as an requirement.

- Data Omission - what requirements will there be around a registrar's ability to signal that it has collected some data but has not transmitted it to the registry? //PK: covered in R.4.3 ?
- Registration attribution - will there be requirements for attribution of registration actions (who did what), and will cryptography be used? //PK: extension ?
- Registrant verification - will there be requirements to support registrant verification (NIS2)? //PK: this is a typical extension case (new attributes and ops)
- Linking - will the protocol support linking to RDAP objects, other RPP objects, etc...
- Include identification of legal vs. natural contacts. //PK: do we collect also here requirements for certain object types, or only core protocol?
- Expanded common models (when compared to EPP), maybe there should be much more attributes then it is in EPP (Vatnumber / Company Number/ properties for Identity papers) Trademark information and more There are a lot of epp extensions in the wild that try to handle this. //PK: same here
- include also DNS provisioning as potential use-case //PK: this one is beyond current charter
- possibility to seamlessly compose the API with other registry use-cases to have uniform API layer from the client perspective
- investigate possible multi-party authorisation schemas. Use case: DNS operator would get authorisation to update DS or NS record through RPP. //PK: covered in R.8.4 ?
  - This may refer to roles not yet considered in EPP processes, e.g. a third-party DNSSEC signing authority. For a potential use case, see the discussion on the DD mailing list on that topic, which discusses potential approaches for DNS, some of which would require modelling relationships between parent/child/signer zone authorities in a provisioning protocol.

- possibility of mobile app or direct browser integration (use case for registries which directly authenticate their domain holders and allow operations on a domain and/or if RPP would be exposed by a registrar) //PK: I would add a chapter Clients. Covered in R.31.x

20. IANA Considerations

TODO

21. Internationalization Considerations

TODO

22. Security Considerations

TODO

23. Normative References

[BCP56] Nottingham, M., "Building Protocols with HTTP", BCP 56, RFC 9205, June 2022.

<https://www.rfc-editor.org/info/bcp56>

[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>.

[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/