Workgroup:	Network Working Group
Internet-Draft:	draft-wullink-rpp-json-01
Published:	2 March 2026
Intended Status:	Standards Track
Expires:	3 September 2026
Authors:	M. Wullink *SIDN Labs* P. Kowalik *DENIC*

JSON for Restful Provisioning Protocol (RPP)

Abstract

This document defines the rules for representing the RESTful Provisioning Protocol (RPP) data objects, as defined in [I-D.kowalik-rpp-data-objects], using the JavaScript Object Notation (JSON) Data Interchange Format [RFC8259]. It specifies how RPP primitive types, common data types, component objects, resource objects, and associations are mapped to JSON and JSON Schema, and provides normative JSON Schema definitions and worked examples for domain name, contact, and host data objects.

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 3 September 2026.

Copyright Notice

Table of Contents

1. Introduction

The RESTful Provisioning Protocol (RPP) defines a set of data objects for managing foundational registry resources including domain names, contacts, and hosts. The data model is defined in [I-D.kowalik-rpp-data-objects] independently of any particular representation format. This document defines the JSON [RFC8259] representation of those data objects.

JSON has emerged as the de facto standard data format for modern RESTful APIs. Its widespread adoption across tools, libraries, and developer communities makes it well suited as the primary representation format for RPP. This document provides the normative rules and JSON Schema definitions required for implementations to produce and consume RPP messages in JSON.

The separation between the abstract data model and its concrete JSON representation ensures that the protocol's semantic foundation remains stable while enabling the adoption of JSON across diverse deployment environments.

1.1. Motivation

The RESTful Provisioning Protocol (RPP) introduces a new provisioning mechanism that aligns more closely with modern cloud infrastructure, enhancing the scalability of server deployments. While RESTful protocols do not mandate a specific media type for resource description, the widespread adoption of JSON in web services has established it as the de facto standard for modern APIs. The increasing availability of tools, software libraries, and a skilled workforce has led several registries to adopt JSON for data exchange within their API ecosystems. Registries supporting JSON can offer a unified API ecosystem that extends beyond domain name and IP address provisioning, maintaining a consistent technology stack, data formats, and developer experience.

JSON's syntax, known for its straightforwardness and minimal verbosity, significantly eases the tasks of writing, reading, and maintaining code. This simplicity is especially advantageous for the rapid comprehension and integration of provisioning APIs.

The lightweight nature of JSON can result in faster processing and data transfers, a critical aspect in high-volume transaction environments such as domain registration. Enhanced API response times can lead to more efficient domain lookups, registrations, and updates. JSON parsing is typically fast and well-supported by standard libraries, contributing to improved system performance amid frequent interactions between RPP clients and servers.

However, the absence of a standardised JSON format for domain provisioning has led to the emergence of TLD-specific implementations that lack interoperability, increasing the development effort required for integration. Similarly, at the registrar level, the absence of standards has resulted in numerous incompatible API implementations provided to clients and resellers. Standardising a JSON format for domain provisioning within the RPP framework could mitigate these challenges, reducing fragmentation and simplifying integration efforts across the domain registration industry.

2. Terminology

In this document the following terminology is used.

RPP Data Objects - The abstract data model definitions for domain name, contact, and host resources, as specified in [I-D.kowalik-rpp-data-objects].

RESTful Provisioning Protocol - A RESTful protocol for provisioning heterogeneous database objects.

JSON Schema - A vocabulary that allows annotation and validation of JSON documents, as described in [JSON-SCHEMA].

EPP Compatibility Profile - A set of additional constraints defined in [I-D.kowalik-rpp-data-objects] that a server MUST adhere to when supporting both RPP and EPP concurrently.

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].

JSON is case sensitive. Unless stated otherwise, JSON specifications and examples provided in this document MUST be interpreted in the character case presented. The examples in this document assume that request and response messages are properly formatted JSON documents. Indentation and white space in examples are provided only to illustrate element relationships and for improving readability, and are not REQUIRED features of the protocol.

All JSON Schema definitions in this document use JSON Schema draft 2020-12 [JSON-SCHEMA], and where not provided with a $schema keyword, the following default applies:

```
"$schema": "https://json-schema.org/draft/2020-12/schema"
```

4. JSON Representation Rules

This section defines the normative rules for representing the RPP data model in JSON. The data model is specified in [I-D.kowalik-rpp-data-objects], which defines all primitive types, common data types, component objects, resource objects, and associations independently of any concrete representation format. The rules in this section specify how those abstract definitions map to JSON and JSON Schema version 2020-12.

4.1. Primitive Type Mappings

RPP primitive types MUST be represented in JSON as follows:

RPP Primitive Type	JSON Type	Notes
String	string	Unicode character sequence
Integer	integer	Whole number, positive or negative
Boolean	boolean	true or false
Decimal	number	Base-10 fractional value
Date	string	Full-date as per [RFC3339], e.g. "2025-10-27"
Timestamp	string	Date-time in UTC as per [RFC3339], e.g. "2025-10-27T09:42:51Z"
URL	string	Uniform Resource Locator as per [RFC1738]

RPP Primitive Type	JSON Type	Notes
Binary	string	Base64-encoded binary data

Table 1

4.2. Cardinality Rules

The cardinality of each data element in the RPP data model MUST be represented as follows in JSON:

Rule 1: A data element with cardinality `1` (exactly one) MUST be represented as a JSON property and MUST be present in the containing JSON object. The element MUST be listed under `required` in the corresponding JSON Schema.

```
{
  "type": "object",
  "properties": {
    "name": { "type": "string" }
  },
  "required": ["name"]
}
```

Rule 2: A data element with cardinality `0-1` (zero or one) MUST be represented as an optional JSON property. The element MUST NOT be listed under `required` in the corresponding JSON Schema. When absent, the element MUST be omitted from the JSON object (not represented as `null`).

```
{
  "type": "object",
  "properties": {
    "expiryDate": { "type": "string", "format": "date-time" }
  }
}
```

Rule 3: A data element with cardinality `0+` (zero or more) MUST be represented as an optional JSON array. When no values are present, the property MUST be omitted or represented as an empty array.

```
{
 "type": "object",
 "properties": {
  "status": {
   "type": "array",
   "items": { "$ref": "#/$defs/status" }
  }
 }
}
```

Rule 4: A data element with cardinality `1+` (one or more) MUST be represented as a required JSON array with `"minItems": 1` and the element MUST be listed under `required` in the corresponding JSON Schema.

```
{
 "type": "object",
 "properties": {
  "postalInfo": {
   "type": "array",
   "items": { "$ref": "#/$defs/postalInfo" },
   "minItems": 1
  }
 },
 "required": ["postalInfo"]
}
```

4.3. Mutability Rules

Data elements in the RPP data model carry a mutability attribute: `create-only`, `read-only`, or `read-write`. These MUST be represented in JSON Schema as follows:

Rule 5: Data elements with mutability `read-only` MUST be annotated with `"readOnly": true` in the JSON Schema. Clients MUST NOT include read-only properties in create or update request bodies. Servers MUST ignore any read-only properties provided by a client in a request.

```
{
 "repositoryId": {
  "type": "string",
  "readOnly": true
 }
}
```

Rule 6: Data elements with mutability `create-only` MUST be annotated with `"writeOnly": true` in the JSON Schema for request schemas, and excluded from update request schemas. Servers MUST reject requests that attempt to modify a `create-only` element after object creation.

Rule 7: Data elements with mutability `read-write` have no additional annotation. They MAY appear in both request and response bodies.

4.4. Association Rules

The RPP data model defines several association types between objects, the following rules specify their JSON representations. A Aggregation represents a relationship between two independent objects, where one object references another. A Composition represents a parent-child relationship where the child object is embedded within the parent object and cannot exist independently.

4.5. Labelled associations

Some associations between objects carry a string label that provides additional context for the relationship. The label is not an identifier of the target object, but rather a descriptor of the association itself. Labelled associations can occur in both aggregations and compositions. When representing labelled associations in JSON, the property `label` MUST be included alongside the reference to the target object. A property with the name `object` MUST be used to contain the reference to the target object, which can be either limited representation containing at minimum the primary object identifier for aggregations or an embedded object for compositions.

4.5.1. Aggregation

An `Aggregation[Type]` represents a relationship between two independent objects. When the cardinality allows more than one target, it MUST be represented as a JSON array. Each element of the array MUST be the identifier of the referenced object.

Rule 8: `Aggregation[Type]` with cardinality `0+` or `1+` MUST be represented as a JSON array of embedded objects. Each object in the array MUST include the data elements of the referenced object type that are relevant to the context (at minimum the primary identifier field). Other data elements of the referenced object type MAY be included as needed to provide additional context for the client, but are not required. The JSON Schema MUST allow for the presence of these additional fields.

Example: domain nameservers (Aggregation[Host Data Object]) in a read response, returning a limited object representation, only cvontaining the primary identifier field hostName:

```
{
   "@type": "domainName",
   "name": "name.example",
   "nameservers": [
      { "@type": "host", "hostName": "ns1.name.example" },
      { "@type": "host", "hostName": "ns2.name.example" }
   ]
}
```

4.5.2. Composition

A Composition[Type] represents a parent-child relationship where the child's lifecycle is bound to the parent and the child cannot exist independently of the parent. In JSON, the child object MUST be fully embedded within the parent object. The JSON representation of a composition is the same as that of an aggregation. The distinction between the two is semantic and does not affect the JSON structure.

```
{
     "@type": "domainName",
     "name": "name.example",
     "nameservers": [
        {
           "@type": "host",
           "hostName": "ns1.name.example",
           "provisioningMetadata": {
              "@type": "provisioningMetadata",
              "repositoryId": "NS1EXAMPLE-REP",
              "sponsoringClientId": "ClientX"
           },
           "status": [ { "@type": "status", "label": "ok" } ],
           "dns": [
              {
                 "@type": "dnsResourceRecord",
                 "hostNamelabel": "ns1.name.example",
                 "type": "A",
                 "data": "192.0.2.1",
                 "ttl": 3600
              }
           ]
        }
     ]
}
```

4.5.3. Labelled Aggregation

A LabelledAggregation[Type] is a relationship between two independent objects where each association carries a string label. Multiple associations with the same label are allowed.

Rule 9: LabelledAggregation[Type] with cardinality 0+ MUST be represented as a JSON array of objects. Each object in the array MUST contain a label property (string) alongside the identifier of the referenced object. The object MUST include at least the primary identifier field of the referenced object type. Other data elements of the referenced object type MAY be included as needed to provide additional context for the client, but are not required. The JSON Schema MUST allow for the presence of these additional fields.

Example: domain contacts (LabelledAggregation[Contact Object]):

```
"contacts": [
   {
      "label": "admin",
      "object": {
         "@type": "contact",
         "id": "ABC-8013"
      }
   },
   {
      "label": "tech",
      "object": {
         "@type": "contact",
         "id": "ABC-8014"
      }
    }
]
```

4.5.4. Dictionary Aggregation

A DictionaryAggregation[Type] is a relationship between two independent objects where each association carries a unique string label that serves as a dictionary key.

Rule 10: DictionaryAggregation[Type] MUST be represented as a JSON object where each key is the unique label and the corresponding value is the referenced object, the object MUST include at least the primary identifier field of the referenced object type. Other data elements of the referenced object type MAY be included as needed to provide additional context for the client, but are not required. The JSON Schema MUST allow for the presence of these additional fields.

Example: domain contacts keyed by unique role (DictionaryAggregation[Contact Object]):

```
"contacts": {
   "admin": {
      "@type": "contact",
      "id": "ABC-8013"
   },
   "tech": {
      "@type": "contact",
      "id": "ABC-8014"
   }
}
```

4.5.5. Labelled Composition

A LabelledComposition[Type] is a parent-child relationship where each embedded child carries a string label. Multiple instances with the same label are allowed.

Rule 11: LabelledComposition[Type] with cardinality 0+ MUST be represented as a JSON array of embedded objects. Each object in the array MUST contain a label property alongside the data elements of the composed type.

Example: contact postal info (LabelledComposition[Postal Info Object]):

```
"addresses": [
   {
      "label": "int",
      "object": {
         "@type": "postalInfo",
         "type": "PERSON",
         "name": "John Doe",
         "addr": {
            "@type": "postalAddress",
            "street": ["123 Example Dr."],
            "city": "Dulles",
            "sp": "VA",
            "pc": "20166-6503",
            "cc": "US"
         }
      }
   }
]
```

4.5.6. Dictionary Composition

A DictionaryComposition[Type] is a parent-child relationship where each embedded child carries a unique string label used as a dictionary key.

Rule 12: DictionaryComposition[Type] MUST be represented as a JSON object where each key is the unique label and the corresponding value is the fully embedded child object.

Example: contact postal info (DictionaryComposition[Postal Info Object]):

```
"addresses": {
   "int": {
      "@type": "postalInfo",
      "type": "PERSON",
      "name": "John Doe",
      "addr": {
         "@type": "postalAddress",
         "street": ["123 Example Dr."],
         "city": "Dulles",
         "sp": "VA",
         "pc": "20166-6503",
         "cc": "US"
      }
   }
}
```

4.6. Object Identifier Rules

Rule 13: When a resource or component object is referenced by identifier (for example in an aggregation), the identifier MUST be represented as a JSON string using the value of the object's primary identifier data element.

Rule 14: When a resource or component object is embedded (as in a composition), all data elements of the object MUST be represented as properties of a JSON object according to the rules of this section.

4.7. JSON Schema Definition Rules

Rule 15: Each RPP component object and resource object MUST have a corresponding JSON Schema definition. Object definitions MUST be placed in the `$defs` keyword of the JSON Schema document.

Rule 16: Identifier fields MUST use `"type": "string"` in JSON Schema.

Rule 17: Enumeration constraints on string fields MUST be expressed using the `"enum"` keyword in JSON Schema.

Example (Transfer Status enum):

```
"transferStatus": {
   "type": "string",
   "enum": ["pending", "clientApproved", "clientCancelled",
          "clientRejected", "serverApproved", "serverCancelled"]
}
```

Rule 18: Each JSON Schema definition for an RPP object MUST include a `"required"` array listing all data elements with cardinality `1` or `1+`.

Rule 19: JSON Schema definitions for shared RPP objects MUST NOT use `"additionalProperties": false` if the schema is intended to be extended, However, root schemas MUST use `"unevaluatedProperties"`: false to prevent the presence of undeclared properties in JSON subschemas.

Rule 20: Every RPP object representation MUST include a `"@type"` property whose value is the object's identifier as registered in the IANA RPP Data Object Registry. This property enables identification and allows clients and servers to unambiguously determine the type of an object. The `"@type"` property MUST be included in the JSON Schema `"properties"` object for each RPP object definition with a `"const"` constraint fixing the value to the object's registered identifier. The `"@type"` property MUST be listed in the `"required"` array of the corresponding JSON Schema definition.

Example (Domain Name Data Object):

```
{
  "@type": "domainName",
  "name": "example.example"
}
```

Rule 21: When a transfer request or other operation requires authorization information (e.g., EPP-style authinfo), the client MUST NOT include the `authorisationInformation` object in the JSON request body. Instead, the client MUST convey the authorization information using the `RPP-Authorization` HTTP request header as defined in [I-D.wullink-rpp-core]. Servers MUST reject any request that includes an `authorisationInformation` object in the JSON body with an appropriate error response.

4.7.1. RPP Profiles and Validation

RPP profiles, such as the EPP Compatibility Profile defined in [I-D.kowalik-rpp-data-objects], may impose additional constraints on top of the base RPP data model. These additional constraints MUST be enforced by implementations through validation rules that go beyond what can be expressed in JSON Schema. Such validation rules MUST be clearly documented in the profile specification and implemented by both clients and servers when operating under that profile. For example, the EPP Compatibility Profile requires that certain fields be present in specific object types, and that certain identifier fields conform to EPP syntax rules. These constraints cannot be fully captured in JSON Schema and therefore require additional validation logic in implementations.

5. JSON Schema Definitions

This section provides normative JSON Schema definitions for RPP component objects and resource objects. All schemas use JSON Schema draft 2020-12 [JSON-SCHEMA].

5.1. Common Component Schemas

This section defines shared data types that are based on the primitive data types above and are re-used across multiple data object definitions.

5.1.1. Identifier

Identifiers are character strings with a specified minimum length, a specified maximum length, and a specified format outlined in Section 2.8 of [RFC5730]. Identifiers for certain object types MAY have additional constraints imposed either by server policy, object-specific specifications, or both.

5.1.2. Client Identifier

Client identifiers are character strings with a specified minimum length, a specified maximum length, and a specified format. Client identifiers use the `clIDType` syntax described in [RFC5730].

In JSON, a Client Identifier MUST be represented as a `string` value.

```
{
  "$defs": {
    "clientIdentifier": {
      "type": "string",
      "minLength": 3,
      "maxLength": 16,
      "pattern": "^[a-zA-Z0-9]([-a-zA-Z0-9]*[a-zA-Z0-9])?$"
    }
  }
}
```

5.1.3. Phone Number

Telephone number syntax is derived from structures defined in [ITU.E164.2005]. Telephone numbers described in this specification are character strings that MUST begin with a plus sign ("+", ASCII value 0x002B), followed by a country code defined in [ITU.E164.2005], followed by a dot (".", ASCII value 0x002E), followed by a sequence of digits representing the telephone number. An optional "x" (ASCII value 0x0078) separator with additional digits representing extension information can be appended to the end of the value.

In JSON, a Phone Number MUST be represented as a `string` value conforming to the pattern described above.

```
{
  "$defs": {
    "phoneNumber": {
      "type": "string",
      "pattern": "^\\+[0-9]{1,3}\\.[0-9]+( x[0-9]+)?$"
    }
  }
}
```

5.1.4. Period Object

```
{
 "$defs": {
  "period": {
   "type": "object",
   "properties": {
    "@type": { "type": "string", "const": "period" },
    "value": {
     "type": "integer",
     "minimum": 1,
     "maximum": 99
    },
    "unit": {
     "type": "string",
     "enum": ["y", "m"]
    }
   },
   "required": ["@type", "value", "unit"]
  }
 }
}
```

5.1.5. Provisioning Metadata Object

The following constraints cannot be expressed in JSON Schema and MUST be enforced by implementations:

- `updatingClientId` and `updateDate` MUST NOT be present if the object has never been modified.
- `transferDate` MUST NOT be present if the object has never been transferred.
- In EPP Compatibility Profile, `repositoryId` MUST be provided.

```
{
  "$defs": {
    "provisioningMetadata": {
      "type": "object",
      "properties": {
        "@type":            { "type": "string", "const": "provisioningMetadata", "readOnly":
true },
        "repositoryId":      { "type": "string", "readOnly": true },
        "sponsoringClientId": { "$ref": "#/$defs/clientIdentifier", "readOnly": true },
        "creatingClientId":   { "$ref": "#/$defs/clientIdentifier", "readOnly": true },
        "creationDate":      { "type": "string", "format": "date-time", "readOnly": true },
        "updatingClientId":   { "$ref": "#/$defs/clientIdentifier", "readOnly": true },
        "updateDate":        { "type": "string", "format": "date-time", "readOnly": true },
        "transferDate":      { "type": "string", "format": "date-time", "readOnly": true }
      },
      "required": ["@type", "sponsoringClientId"]
    }
  }
}
```

5.1.6. Status Object

The following constraints cannot be expressed in JSON Schema and MUST be enforced by implementations:

- label MUST use camelCase notation using only ASCII alphabetic characters. Labels set explicitly by the server MUST use the prefix "server"; labels set explicitly by a client MUST use the prefix "client"; all other labels MUST NOT use either prefix. The allowed set of label values depends on the provisioning object type and MAY be extended by extensions.
- due: Servers MAY restrict the ability of clients to set or update this value.
- When the RGP feature is supported, the following additional status labels MAY appear on objects that support RGP: addPeriod, autoRenewPeriod, renewPeriod, transferPeriod, redemptionPeriod, pendingRestore, rgpPendingDelete. The labels redemptionPeriod, pendingRestore, and rgpPendingDelete MUST only appear alongside the standard pendingDelete status.

```
{
 "$defs": {
  "status": {
   "type": "object",
   "properties": {
    "@type":  { "type": "string", "const": "status" },
    "label":  { "type": "string", "pattern": "^[a-zA-Z]+$" },
    "reason": { "type": "string" },
    "due":    { "type": "string", "format": "date-time" }
   },
   "required": ["@type", "label"]
  }
 }
}
```

5.1.7. DNS Resource Record

The following constraints cannot be expressed in JSON Schema and MUST be enforced by implementations:

- `hostNamelabel` MUST be a syntactically valid DNS host name in zone file string representation. Both absolute FQDNs and relative host names are allowed.
- `type` MUST be a valid string representation of a DNS resource record type as defined in [RFC1035]. Allowed values MAY be further constrained by server policy.
- `data` MUST be a syntactically valid resource record data value for the given `type` in zone file string representation.
- `ttl` value range MAY be constrained by server policy.

```
{
 "$defs": {
  "dnsResourceRecord": {
   "type": "object",
   "properties": {
    "@type":        { "type": "string", "const": "dnsResourceRecord" },
    "hostNamelabel": { "type": "string", "format": "hostname" },
    "type":         { "type": "string" },
    "data":         { "type": "string" },
    "ttl":          { "type": "integer" }
   },
   "required": ["@type", "hostNamelabel", "type", "data", "ttl"]
  }
 }
}
```

5.1.8. Authorisation Information Object

The following constraints cannot be expressed in JSON Schema and MUST be enforced by implementations:

- `method` MUST be one of the values registered in the IANA RPP Authorisation Method Registry as defined in [I-D.wullink-rpp-core]. In EPP Compatibility Profile, this value MUST be "authinfo" for standard password-based authorisation.
- The Authorisation Information Object is immutable. When authorisation information changes, a new instance MUST be created rather than modifying the existing one. The value of `authdata` MAY not be returned in read responses, depending on the method and server policy.

```
{
 "$defs": {
  "authorisationInformation": {
   "type": "object",
   "properties": {
    "@type":    { "type": "string", "const": "authorisationInformation" },
    "method":   { "type": "string" },
    "authdata": { "type": "string" }
   },
   "required": ["@type", "method", "authdata"]
  }
 }
}
```

5.1.9. Postal Address Object

The following constraints cannot be expressed in JSON Schema and MUST be enforced by implementations:

- `cc` MUST be a valid two-character country code from [ISO3166-1]. The JSON Schema pattern enforces uppercase alpha-2 format.
- In EPP Compatibility Profile, `city` and `cc` MUST be provided.

```
{
 "$defs": {
  "postalAddress": {
   "type": "object",
   "properties": {
    "@type": { "type": "string", "const": "postalAddress" },
    "street": {
     "type": "array",
     "items": { "type": "string" }
    },
    "city":  { "type": "string" },
    "sp":    { "type": "string" },
    "pc":    { "type": "string" },
    "cc":    { "type": "string", "pattern": "^[A-Z]{2}$" }
   },
   "required": ["@type"]
  }
 }
}
```

5.1.10. Postal Info Object

The following constraints cannot be expressed in JSON Schema and MUST be enforced by implementations:

- `name` MAY be required by implementations when `type` is "PERSON". In EPP Compatibility Profile, `name` MUST be provided.
- `org` MAY be required by implementations when `type` is "ORG".
- In EPP Compatibility Profile, `addr` MUST be provided.

```
{
 "$defs": {
  "postalInfo": {
   "type": "object",
   "properties": {
    "@type": { "type": "string", "const": "postalInfo" },
    "type": {
     "type": "string",
     "enum": ["PERSON", "ORG"]
    },
    "name": { "type": "string" },
    "org":  { "type": "string" },
    "addr": { "$ref": "#/$defs/postalAddress" }
   },
   "required": ["@type"]
  }
 }
}
```

5.1.11. Transfer Data Object

```
{
 "$defs": {
  "transferData": {
   "type": "object",
   "properties": {
    "@type": { "type": "string", "const": "transferData", "readOnly": true },
    "transferStatus": {
     "type": "string",
     "enum": ["pending", "clientApproved", "clientCancelled",
            "clientRejected", "serverApproved", "serverCancelled"],
     "readOnly": true
    },
    "transferDirection": {
     "type": "string",
     "enum": ["pull", "push"],
     "readOnly": true
    },
    "requestingClientId": { "$ref": "#/$defs/clientIdentifier", "readOnly": true },
    "requestDate":        { "type": "string", "format": "date-time", "readOnly": true },
    "actingClientId":     { "$ref": "#/$defs/clientIdentifier", "readOnly": true },
    "actionDate":         { "type": "string", "format": "date-time", "readOnly": true }
   },
   "required": [
    "@type", "transferStatus", "transferDirection", "requestingClientId",
    "requestDate", "actingClientId", "actionDate"
   ]
  }
 }
}
```

5.1.12. Restore Data Object

The Restore Data Object represents the current state of a restore request for an object that has entered the Redemption Grace Period (RGP). It is returned as the output of all restore operations.

The following constraints cannot be expressed in JSON Schema and MUST be enforced by implementations:

- `requestDate` MUST NOT be present if no restore request has been submitted yet.
- `reportDate` MUST NOT be present if no restore report has been accepted yet.
- `reportDueDate` MUST NOT be present when `restoreStatus` is not `"pendingRestore"`.

```
{
  "$defs": {
    "restoreData": {
      "type": "object",
      "properties": {
        "@type":         { "type": "string", "const": "restoreData", "readOnly": true },
        "restoreStatus": {
          "type": "string",
          "enum": ["pendingRestore", "restored", "rgpPendingDelete"],
          "readOnly": true
        },
        "requestDate":   { "type": "string", "format": "date-time", "readOnly": true },
        "reportDate":    { "type": "string", "format": "date-time", "readOnly": true },
        "reportDueDate": { "type": "string", "format": "date-time", "readOnly": true }
      },
      "required": ["@type", "restoreStatus"]
    }
  }
}
```

5.1.13. Restore Report Object

The Restore Report Object contains the redemption grace period restore report submitted by the sponsoring client as required by the RGP process ([RFC3915]).

The following constraints cannot be expressed in JSON Schema and MUST be enforced by implementations:

- At least one and at most two `statements` MUST be provided.
- `restoreTime` MAY be omitted when the restore report is submitted inline within the restore request in a single-step process.
- In EPP Compatibility Profile, `restoreTime` MUST be present as defined in [RFC3915].
- In EPP Compatibility Profile, exactly two `statements` MUST be present as defined in [RFC3915].

```
{
  "$defs": {
    "restoreReport": {
      "type": "object",
      "properties": {
        "@type":        { "type": "string", "const": "restoreReport", "readOnly": true },
        "preData":      { "type": "string" },
        "postData":     { "type": "string" },
        "deleteTime":   { "type": "string", "format": "date-time" },
        "restoreTime":  { "type": "string", "format": "date-time" },
        "restoreReason": { "type": "string" },
        "statements": {
          "type": "array",
          "items": { "type": "string" },
          "minItems": 1,
          "maxItems": 2
        },
        "other": { "type": "string" }
      },
      "required": ["@type", "statements"]
    }
  }
}
```

5.2. Resource Object Schemas

Resource objects represent the main entities managed by RPP: domain names, contacts, and hosts. Each resource object has a corresponding root JSON Schema definition that specifies its properties, required fields, and constraints.

5.2.1. Domain Name Data Object

The Domain Name Data Object represents a domain name and its associated provisioning data.

The following constraints cannot be expressed in JSON Schema and MUST be enforced by implementations:

- `name` MUST be a fully qualified domain name conforming to the syntax described in [RFC1035]. Servers MAY restrict allowable domain names to a specific namespace for which they are authoritative. The implicit trailing dot MUST NOT be included.

Create request schema (create-only and read-write properties):

```
{
 "$schema": "https://json-schema.org/draft/2020-12/schema",
 "type": "object",
 "properties": {
  "@type": { "type": "string", "const": "domainName" },
  "name": { "type": "string" },
  "registrant": { "type": "string" },
  "contacts": {
   "type": "array",
   "items": { "$ref": "#/$defs/contact" }
  },
  "nameservers": {
   "type": "array",
   "items": { "$ref": "#/$defs/host" }
  },
  "dns": {
   "type": "array",
   "items": { "$ref": "#/$defs/dnsResourceRecord" }
  },
  "authorisationInformation": { "$ref": "#/$defs/authInfo" },
  "period":   { "$ref": "#/$defs/period" }
 },
 "required": ["@type", "name"],
 "unevaluatedProperties": false
}
```

Read response schema (read-write and read-only properties):

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "type": "object",
  "properties": {
    "@type":              { "type": "string", "const": "domainName", "readOnly": true },
    "name":               { "type": "string", "readOnly": true },
    "provisioningMetadata":  { "$ref": "#/$defs/provisioningMetadata" },
    "status": {
      "type": "array",
      "items": { "$ref": "#/$defs/status" },
      "readOnly": true
    },
    "registrant":  { "type": "string" },
    "contacts": {
      "type": "array",
      "items": { "$ref": "#/$defs/contact" }
    },
    "nameservers": {
      "type": "array",
      "items": { "$ref": "#/$defs/host" }
    },
    "dns": {
      "type": "array",
      "items": { "$ref": "#/$defs/dnsResourceRecord" }
    },
    "subordinateHosts": {
      "type": "array",
      "items": { "$ref": "#/$defs/host" },
      "readOnly": true
    },
    "expiryDate": { "type": "string", "format": "date-time", "readOnly": true },
    "authorisationInformation":   { "$ref": "#/$defs/authInfo" }
  },
  "required": ["@type", "name", "provisioningMetadata"],
  "unevaluatedProperties": false
}
```

5.2.2. Contact Data Object

The following constraints cannot be expressed in JSON Schema and MUST be enforced by implementations:

- `postalInfo` keys MUST be either "int" (internationalised, all-ASCII) or "loc" (localised, MAY use non-ASCII characters). At most one entry of each key is allowed.

Create request schema (create-only and read-write properties):

```
{
 "$schema": "https://json-schema.org/draft/2020-12/schema",
 "type": "object",
 "properties": {
  "@type": { "type": "string", "const": "contact" },
  "id": { "type": "string" },
  "postalInfo": {
   "type": "object",
   "additionalProperties": { "$ref": "#/$defs/postalInfo" },
   "minProperties": 1,
   "maxProperties": 2
  },
  "voice": {
   "type": "array",
   "items": { "$ref": "#/$defs/phoneNumber" }
  },
  "fax": {
   "type": "array",
   "items": { "$ref": "#/$defs/phoneNumber" }
  },
  "email": {
   "type": "array",
   "items": { "type": "string", "format": "email" }
  },
  "authorisationInformation": { "$ref": "#/$defs/authInfo" },
  "disclose":  { "type": "object" }
 },
 "required": ["@type", "id", "postalInfo"],
 "unevaluatedProperties": false
}
```

Read response schema (read-write and read-only properties):

```
{
 "$schema": "https://json-schema.org/draft/2020-12/schema",
 "type": "object",
 "properties": {
  "@type": { "type": "string", "const": "contact", "readOnly": true },
  "id": { "type": "string", "readOnly": true },
  "provisioningMetadata": { "$ref": "#/$defs/provisioningMetadata" },
  "status": {
   "type": "array",
   "items": { "$ref": "#/$defs/status" },
   "readOnly": true
  },
  "postalInfo": {
   "type": "object",
   "additionalProperties": { "$ref": "#/$defs/postalInfo" },
   "minProperties": 1,
   "maxProperties": 2
  },
  "voice": {
   "type": "array",
   "items": { "$ref": "#/$defs/phoneNumber" }
  },
  "fax": {
   "type": "array",
   "items": { "$ref": "#/$defs/phoneNumber" }
  },
  "email": {
   "type": "array",
   "items": { "type": "string", "format": "email" }
  },
  "authorisationInformation": { "$ref": "#/$defs/authInfo" },
  "disclose":  { "type": "object" }
 },
 "required": ["@type", "id", "provisioningMetadata", "postalInfo"],
 "unevaluatedProperties": false
}
```

5.2.3. Host Data Object

The following constraints cannot be expressed in JSON Schema and MUST be enforced by implementations:

- `hostName` MUST be a syntactically valid fully qualified host name.
- If the host name is subordinate to a domain for which the server is authoritative, the superordinate domain MUST already exist in the server.

Create request schema (create-only and read-write properties):

```
{
 "$schema": "https://json-schema.org/draft/2020-12/schema",
 "type": "object",
 "properties": {
  "@type":    { "type": "string", "const": "host" },
  "hostName": { "type": "string", "format": "hostname" },
  "dns": {
   "type": "array",
   "items": { "$ref": "#/$defs/dnsResourceRecord" }
  }
 },
 "required": ["@type", "hostName"],
 "unevaluatedProperties": false
}
```

Read response schema (read-write and read-only properties):

```
{
 "$schema": "https://json-schema.org/draft/2020-12/schema",
 "type": "object",
 "properties": {
  "@type":    { "type": "string", "const": "host", "readOnly": true },
  "hostName": { "type": "string", "format": "hostname" },
  "provisioningMetadata": { "$ref": "#/$defs/provisioningMetadata" },
  "status": {
   "type": "array",
   "items": { "$ref": "#/$defs/status" },
   "readOnly": true
  },
  "dns": {
   "type": "array",
   "items": { "$ref": "#/$defs/dnsResourceRecord" }
  }
 },
 "required": ["@type", "hostName", "provisioningMetadata"],
 "unevaluatedProperties": false
}
```

6. Examples

This section provides examples that follow the JSON representation rules and JSON Schema definitions specified in the previous sections. The examples illustrate typical request and response messages for domain name, contact, and host resources.

6.1. Domain Name

6.1.1. Create

Example domain create request:

```
{
   "@type": "domainName",
   "name": "example.example",
   "period": {
      "@type": "period",
      "value": 2,
      "unit": "y"
   },
   "nameservers": [
      { "@type": "host", "hostName": "ns1.example.example" },
      { "@type": "host", "hostName": "ns2.example.example" }
   ],
   "registrant": "jd1234",
   "contacts": [
      { "label": "admin", "id": "sh8013" },
      { "label": "tech",  "id": "sh8013" }
   ],
   "authorisationInformation": {
      "@type": "authorisationInformation",
      "method": "authinfo",
      "authdata": "2fooBAR"
   }
}
```

Example domain create response from a server with RGP support:

```
{
   "@type": "domainName",
   "name": "example.example",
   "provisioningMetadata": {
      "@type": "provisioningMetadata",
      "repositoryId": "EXAMPLE1-REP",
      "sponsoringClientId": "ClientX",
      "creatingClientId": "ClientX",
      "creationDate": "1999-04-03T22:00:00.0Z"
   },
   "status": [
      { "@type": "status", "label": "ok" },
      { "@type": "status", "label": "addPeriod" }
   ],
   "expiryDate": "2001-04-03T22:00:00.0Z"
}
```

6.1.2. Read

Example domain read response:

```
{
   "@type": "domainName",
   "name": "example.example",
   "provisioningMetadata": {
      "@type": "provisioningMetadata",
```

```
      "repositoryId": "EXAMPLE1-REP",
      "sponsoringClientId": "ClientX",
      "creatingClientId": "ClientY",
      "creationDate": "1999-04-03T22:00:00.0Z",
      "updatingClientId": "ClientX",
      "updateDate": "1999-12-03T09:00:00.0Z",
      "transferDate": "2000-04-08T09:00:00.0Z"
   },
   "status": [
      { "@type": "status", "label": "ok" }
   ],
   "registrant": "jd1234",
   "contacts": [
      { "label": "admin", "id": "sh8013" },
      { "label": "tech",  "id": "sh8013" }
   ],
   "nameservers": [
      {
         "@type": "host",
         "hostName": "ns1.example.example",
         "provisioningMetadata": {
            "@type": "provisioningMetadata",
            "repositoryId": "NS1EXAMPLE-REP",
            "sponsoringClientId": "ClientX"
         },
         "status": [ { "@type": "status", "label": "ok" } ],
         "dns": [
            {
               "@type": "dnsResourceRecord",
               "hostNamelabel": "ns1.example.example.",
               "type": "A",
               "data": "192.0.2.1",
               "ttl": 3600
            }
         ]
      },
      {
         "@type": "host",
         "hostName": "ns1.example.example",
         "provisioningMetadata": {
            "@type": "provisioningMetadata",
            "repositoryId": "NS1EXAMPLENET-REP",
            "sponsoringClientId": "ClientZ"
         },
         "status": [ { "@type": "status", "label": "ok" } ]
      }
   ],
   "subordinateHosts": [
      {
         "@type": "host",
         "hostName": "ns1.example.example",
         "provisioningMetadata": {
            "@type": "provisioningMetadata",
            "repositoryId": "NS1EXAMPLE-REP",
            "sponsoringClientId": "ClientX"
         },
         "status": [ { "@type": "status", "label": "ok" } ]
      },
```

```
      {
         "@type": "host",
         "hostName": "ns2.example.example",
         "provisioningMetadata": {
            "@type": "provisioningMetadata",
            "repositoryId": "NS2EXAMPLE-REP",
            "sponsoringClientId": "ClientX"
         },
         "status": [ { "@type": "status", "label": "ok" } ]
      }
   ],
   "expiryDate": "2005-04-03T22:00:00.0Z",
   "authorisationInformation": {
      "@type": "authorisationInformation",
      "method": "authinfo",
      "authdata": "2fooBAR"
   }
}
```

6.1.3. Update

Example domain update request (read-write properties):

```
{
   "@type": "domainName",
   "registrant": "sh8013",
   "authorisationInformation": {
      "@type": "authorisationInformation",
      "method": "authinfo",
      "authdata": "2BARfoo"
   }
}
```

Example domain update response:

```
{
   "@type": "domainName",
   "name": "example.example",
   "provisioningMetadata": {
      "@type": "provisioningMetadata",
      "repositoryId": "EXAMPLE1-REP",
      "sponsoringClientId": "ClientX",
      "creatingClientId": "ClientY",
      "creationDate": "1999-04-03T22:00:00.0Z",
      "updatingClientId": "ClientX",
      "updateDate": "2000-01-15T09:00:00.0Z"
   },
   "status": [
      { "@type": "status", "label": "ok" }
   ],
   "registrant": "sh8013"
}
```

6.1.4. Delete

The domain delete operation takes the domain name as the resource identifier in the request. No request body is required.

Example domain delete response (minimal, server may return full representation):

```
{
   "@type": "domainName",
   "name": "example.example",
   "provisioningMetadata": {
      "@type": "provisioningMetadata",
      "repositoryId": "EXAMPLE1-REP",
      "sponsoringClientId": "ClientX"
   }
}
```

6.1.5. Renew

The renew operation accepts a transient `currentExpiryDate` parameter for validation and an optional `renewalPeriod`.

Example domain renew request:

```
{
  "currentExpiryDate": "2005-04-03T22:00:00.0Z",
  "renewalPeriod": {
     "@type": "period",
     "value": 5,
     "unit": "y"
  }
}
```

Example domain renew response:

```
{
  "@type": "domainName",
  "name": "example.example",
  "expiryDate": "2010-04-03T22:00:00.0Z"
}
```

6.1.6. Transfer Request

Authorization information for the transfer MUST be conveyed using the `RPP-Authorization` HTTP header (see Rule 21), not in the JSON request body.

Example domain transfer request (pull transfer)

```
{
  "transferDirection": "pull",
  "transferPeriod": {
     "@type": "period",
     "value": 1,
     "unit": "y"
  }
}
```

Example domain transfer response (Transfer Data Object):

```
{
  "@type": "transferData",
  "transferStatus": "pending",
  "transferDirection": "pull",
  "requestingClientId": "ClientX",
  "requestDate": "2000-06-08T22:00:00.0Z",
  "actingClientId": "ClientY",
  "actionDate": "2000-06-13T22:00:00.0Z",
  "expiryDate": "2002-09-08T22:00:00.0Z"
}
```

6.1.7. Transfer Query

Example domain transfer query response (Transfer Data Object):

```
{
   "@type": "transferData",
   "transferStatus": "pending",
   "transferDirection": "pull",
   "requestingClientId": "ClientX",
   "requestDate": "2000-06-06T22:00:00.0Z",
   "actingClientId": "ClientY",
   "actionDate": "2000-06-11T22:00:00.0Z",
   "expiryDate": "2002-09-08T22:00:00.0Z"
}
```

6.1.8. Transfer Cancel / Reject / Approve

Transfer cancel, reject, and approve responses return the Transfer Data Object. The response structure is the same as the Transfer Query response above. The `transferStatus` value reflects the outcome of the operation (e.g. `"clientCancelled"`, `"clientRejected"`, or `"clientApproved"`).

6.1.9. Restore Request

Example domain restore request (without inline report; object transitions to `pendingRestore` state):

```
{}
```

Example domain restore response (Restore Data Object, server requires a report):

```
{
   "@type": "restoreData",
   "restoreStatus": "pendingRestore",
   "requestDate": "2025-01-20T15:30:00.0Z",
   "reportDueDate": "2025-01-27T15:30:00.0Z"
}
```

Example domain restore request with inline restore report (single-step; object restored immediately):

```
{
   "@type": "domainName",
   "restoreReport": {
      "@type": "restoreReport",
      "preData": "Domain example.example was registered on 2024-01-15 with registrant
jd1234.",
      "postData": "Domain example.example is being restored with the same registration
data.",
      "deleteTime": "2025-01-10T12:00:00.0Z",
      "restoreTime": "2025-01-20T15:30:00.0Z",
      "restoreReason": "Domain deleted in error by client operator.",
      "statements": [
         "The information in this report is true to the best of my knowledge.",
         "I have a valid reason for restoring this domain name."
      ]
   }
}
```

Example domain restore response with inline report (Restore Data Object, immediately restored):

```
{
   "@type": "restoreData",
   "restoreStatus": "restored",
   "requestDate": "2025-01-20T15:30:00.0Z",
   "reportDate": "2025-01-20T15:30:00.0Z"
}
```

6.1.10. Restore Report

Example domain restore report request:

```
{
   "@type": "domainName",
   "restoreReport": {
      "@type": "restoreReport",
      "preData": "Domain example.example was registered on 2024-01-15 with registrant
jd1234.",
      "postData": "Domain example.example is being restored with the same registration
data.",
      "deleteTime": "2025-01-10T12:00:00.0Z",
      "restoreTime": "2025-01-20T15:30:00.0Z",
      "restoreReason": "Domain deleted in error by client operator.",
      "statements": [
         "The information in this report is true to the best of my knowledge.",
         "I have a valid reason for restoring this domain name."
      ]
   }
}
```

Example domain restore report response (Restore Data Object):

```
{
  "@type": "restoreData",
  "restoreStatus": "restored",
  "requestDate": "2025-01-20T15:30:00.0Z",
  "reportDate": "2025-01-22T09:15:00.0Z"
}
```

6.1.11. Restore Query

The Restore Query operation takes no request body (Parameters: None).

```
{}
```

Example domain restore query response (Restore Data Object, object in `pendingRestore` state):

```
{
  "@type": "restoreData",
  "restoreStatus": "pendingRestore",
  "requestDate": "2025-01-20T15:30:00.0Z",
  "reportDueDate": "2025-01-27T15:30:00.0Z"
}
```

Example domain restore query response (Restore Data Object, object restored):

```
{
  "@type": "restoreData",
  "restoreStatus": "restored",
  "requestDate": "2025-01-20T15:30:00.0Z",
  "reportDate": "2025-01-22T09:15:00.0Z"
}
```

6.2. Contact

6.2.1. Create

Example contact create request:

```
{
   "@type": "contact",
   "id": "jd1234",
   "postalInfo": {
      "int": {
         "@type": "postalInfo",
         "type": "PERSON",
         "name": "John Doe",
         "org": "Example Inc.",
         "addr": {
            "@type": "postalAddress",
            "street": [
               "123 Example Dr.",
               "Suite 100"
            ],
            "city": "Dulles",
            "sp": "VA",
            "pc": "20166-6503",
            "cc": "US"
         }
      }
   },
   "voice": ["+1.7035555555"],
   "fax": ["+1.7035555556"],
   "email": ["jdoe@example.example"],
   "authorisationInformation": {
      "@type": "authorisationInformation",
      "method": "authinfo",
      "authdata": "2fooBAR"
   }
}
```

Example contact create response:

```
{
   "@type": "contact",
   "id": "jd1234",
   "provisioningMetadata": {
      "@type": "provisioningMetadata",
      "repositoryId": "JD1234-REP",
      "sponsoringClientId": "ClientX",
      "creatingClientId": "ClientX",
      "creationDate": "1999-04-03T22:00:00.0Z"
   },
   "status": [
      { "@type": "status", "label": "ok" }
   ],
   "postalInfo": {
      "int": {
         "@type": "postalInfo",
         "type": "PERSON",
         "name": "John Doe",
         "org": "Example Inc.",
         "addr": {
            "@type": "postalAddress",
            "street": [
               "123 Example Dr.",
               "Suite 100"
            ],
            "city": "Dulles",
            "sp": "VA",
            "pc": "20166-6503",
            "cc": "US"
         }
      }
   },
   "voice": ["+1.7035555555"],
   "fax": ["+1.7035555556"],
   "email": ["jdoe@example.example"]
}
```

6.2.2. Read

Example contact read response:

```
{
   "@type": "contact",
   "id": "jd1234",
   "provisioningMetadata": {
      "@type": "provisioningMetadata",
      "repositoryId": "JD1234-REP",
      "sponsoringClientId": "ClientX",
      "creatingClientId": "ClientX",
      "creationDate": "1999-04-03T22:00:00.0Z",
      "updatingClientId": "ClientX",
      "updateDate": "2000-01-15T09:00:00.0Z"
   },
   "status": [
      { "@type": "status", "label": "ok" }
   ],
   "postalInfo": {
      "int": {
         "@type": "postalInfo",
         "type": "PERSON",
         "name": "John Doe",
         "org": "Example Inc.",
         "addr": {
            "@type": "postalAddress",
            "street": ["123 Example Dr.", "Suite 100"],
            "city": "Dulles",
            "sp": "VA",
            "pc": "20166-6503",
            "cc": "US"
         }
      }
   },
   "voice": ["+1.7035555555"],
   "email": ["jdoe@example.example"]
}
```

6.2.3. Update

TBD

6.2.4. Delete

The contact delete operation takes the contact identifier as the resource identifier. No request body is required.

6.2.5. Transfer Request

Authorization information for the transfer MUST be conveyed using the `RPP-Authorization` HTTP header (see Rule 21), not in the JSON request body.

Example contact transfer request (pull transfer)

```
{
   "transferDirection": "pull"
}
```

Example contact transfer response (Transfer Data Object):

```
{
   "@type": "transferData",
   "transferStatus": "pending",
   "transferDirection": "pull",
   "requestingClientId": "ClientX",
   "requestDate": "2000-06-08T22:00:00.0Z",
   "actingClientId": "ClientY",
   "actionDate": "2000-06-13T22:00:00.0Z"
}
```

6.2.6. Transfer Query

Example contact transfer query response (Transfer Data Object):

```
{
   "@type": "transferData",
   "transferStatus": "pending",
   "transferDirection": "pull",
   "requestingClientId": "ClientX",
   "requestDate": "2000-06-06T22:00:00.0Z",
   "actingClientId": "ClientY",
   "actionDate": "2000-06-11T22:00:00.0Z"
}
```

6.2.7. Transfer Cancel / Reject / Approve

Transfer cancel, reject, and approve responses return the Transfer Data Object. The response structure is the same as the Transfer Query response above. The `transferStatus` value reflects the outcome of the operation (e.g. `"clientCancelled"`, `"clientRejected"`, or `"clientApproved"`).

Note: Unlike domain transfers, contact transfers do not include an `expiryDate` field in the Transfer Data Object, as contacts do not have registration periods.

6.3. Host

6.3.1. Create

Example host create request:

```
{
  "@type": "host",
  "hostName": "ns1.example.example",
  "dns": [
    {
      "@type": "dnsResourceRecord",
      "hostNamelabel": "ns1.example.example.",
      "type": "A",
      "data": "192.0.2.1",
      "ttl": 3600
    },
    {
      "@type": "dnsResourceRecord",
      "hostNamelabel": "ns1.example.example.",
      "type": "AAAA",
      "data": "2001:db8::1",
      "ttl": 3600
    }
  ]
}
```

Example host create response:

```
{
   "@type": "host",
   "hostName": "ns1.example.example",
   "provisioningMetadata": {
      "@type": "provisioningMetadata",
      "repositoryId": "NS1EXAMPLE-REP",
      "sponsoringClientId": "ClientX",
      "creatingClientId": "ClientX",
      "creationDate": "1999-04-03T22:00:00.0Z"
   },
   "status": [
      { "@type": "status", "label": "ok" }
   ],
   "dns": [
      {
         "@type": "dnsResourceRecord",
         "hostNamelabel": "ns1.example.example.",
         "type": "A",
         "data": "192.0.2.1",
         "ttl": 3600
      },
      {
         "@type": "dnsResourceRecord",
         "hostNamelabel": "ns1.example.example.",
         "type": "AAAA",
         "data": "2001:db8::1",
         "ttl": 3600
      }
   ]
}
```


6.3.2. Read

Example host read response:

```
{
   "@type": "host",
   "hostName": "ns1.example.example",
   "provisioningMetadata": {
      "@type": "provisioningMetadata",
      "repositoryId": "NS1EXAMPLE-REP",
      "sponsoringClientId": "ClientX",
      "creatingClientId": "ClientY",
      "creationDate": "1999-04-03T22:00:00.0Z"
   },
   "status": [
      { "@type": "status", "label": "ok" }
   ],
   "dns": [
      {
         "@type": "dnsResourceRecord",
         "hostNamelabel": "ns1.example.example.",
         "type": "A",
         "data": "192.0.2.1",
         "ttl": 3600
      }
   ]
}
```

6.3.3. Update

Example host update request:

```
{
   "@type": "host",
   "hostName": "ns1.example.example",
   "dns": [
      {
         "@type": "dnsResourceRecord",
         "hostNamelabel": "ns1.example.example.",
         "type": "A",
         "data": "198.51.100.1",
         "ttl": 3600
      }
   ]
}
```

6.3.4. Delete

The host delete operation takes the host name as the resource identifier. No request body is required. The server SHOULD reject the request if the host object is associated with any domain name objects.

6.3.5. Restore Request

Example host restore request (without inline report; object transitions to `pendingRestore` state):

```
{}
```

Example host restore request response (Restore Data Object, server requires a report):

```
{
   "@type": "restoreData",
   "restoreStatus": "pendingRestore",
   "requestDate": "2025-01-20T15:30:00.0Z",
   "reportDueDate": "2025-01-27T15:30:00.0Z"
}
```

Example host restore request with inline restore report (single-step; object restored immediately):

```
{
   "@type": "host",
   "restoreReport": {
      "@type": "restoreReport",
      "preData": "Host ns1.example.example was registered on 2024-01-15 by ClientX.",
      "postData": "Host ns1.example.example is being restored with the same registration
data.",
      "deleteTime": "2025-01-10T12:00:00.0Z",
      "restoreTime": "2025-01-20T15:30:00.0Z",
      "restoreReason": "Host deleted in error by client operator.",
      "statements": [
         "The information in this report is true to the best of my knowledge.",
         "I have a valid reason for restoring this host object."
      ]
   }
}
```

Example host restore response with inline report (Restore Data Object, immediately restored):

```
{
   "@type": "restoreData",
   "restoreStatus": "restored",
   "requestDate": "2025-01-20T15:30:00.0Z",
   "reportDate": "2025-01-20T15:30:00.0Z"
}
```

6.3.6. Restore Report

Example host restore report request:

```
{
   "@type": "host",
   "restoreReport": {
      "@type": "restoreReport",
      "preData": "Host ns1.example.example was registered on 2024-01-15 by ClientX.",
      "postData": "Host ns1.example.example is being restored with the same registration
data.",
      "deleteTime": "2025-01-10T12:00:00.0Z",
      "restoreTime": "2025-01-20T15:30:00.0Z",
      "restoreReason": "Host deleted in error by client operator.",
      "statements": [
         "The information in this report is true to the best of my knowledge.",
         "I have a valid reason for restoring this host object."
      ]
   }
}
```

Example host restore report response (Restore Data Object):

```
{
   "@type": "restoreData",
   "restoreStatus": "restored",
   "requestDate": "2025-01-20T15:30:00.0Z",
   "reportDate": "2025-01-22T09:15:00.0Z"
}
```

6.3.7. Restore Query

The Restore Query operation takes no request body (Parameters: None).

Example host restore query response (Restore Data Object, object in pendingRestore state):

```
{
   "@type": "restoreData",
   "restoreStatus": "pendingRestore",
   "requestDate": "2025-01-20T15:30:00.0Z",
   "reportDueDate": "2025-01-27T15:30:00.0Z"
}
```

Example host restore query response (Restore Data Object, object restored):

```
{
   "@type": "restoreData",
   "restoreStatus": "restored",
   "requestDate": "2025-01-20T15:30:00.0Z",
   "reportDate": "2025-01-22T09:15:00.0Z"
}
```

7. IANA Considerations

TODO

8. Internationalization Considerations

TODO

9. Security Considerations

TODO

10. Acknowledgments

TODO

11. Change History

11.1. Version 00 to 01

- Updated all examples and schemas to be based on RPP Data Object and no longer on EPP XML schemas. (Issue #15)
- Updated labelled and dictionary aggregation rules (Issue #17)
- Added required "@type" property to all JSON Schema definitions. (Issue #20)
- Updated all example domain names to use the .example TLD. (Issue #26)

12. References

12.1. Normative References

[I-D.kowalik-rpp-data-objects] Kowalik, P. and M. Wullink, "RESTful Provisioning Protocol (RPP) Data Objects", Work in Progress, Internet-Draft, draft-kowalik-rpp-data-objects-02, 14 November 2025, <https://datatracker.ietf.org/doc/html/draft-kowalik-rpp-data-objects-02>.

[I-D.wullink-rpp-core] Wullink, M. and P. Kowalik, "RESTful Provisioning Protocol (RPP)", Work in Progress, Internet-Draft, draft-wullink-rpp-core-03, 2 November 2025, <https://datatracker.ietf.org/doc/html/draft-wullink-rpp-core-03>.

[ISO3166-1] International Organization for Standardization, "Codes for the representation of names of countries and their subdivisions - Part 1: Country code", ISO 3166-1:2020, 2020, <https://www.iso.org/standard/72482.html>.

[ITU.E164.2005] International Telecommunication Union, "The international public telecommunication numbering plan", ITU-T Recommendation E.164, February 2005.

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <https://www.rfc-editor.org/info/rfc1035>.

[RFC1738] Berners-Lee, T., Masinter, L., and M. McCahill, "Uniform Resource Locators (URL)", RFC 1738, DOI 10.17487/RFC1738, December 1994, <https://www.rfc-editor.org/info/rfc1738>.

[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>.

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

[RFC3915] Hollenbeck, S., "Domain Registry Grace Period Mapping for the Extensible Provisioning Protocol (EPP)", RFC 3915, DOI 10.17487/RFC3915, September 2004, <https://www.rfc-editor.org/rfc/rfc3915>.

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

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

12.2. Informative References

[JSON-SCHEMA] JSON Schema, "JSON Schema: A Media Type for Describing JSON Documents", 2020, <https://json-schema.org/draft/2020-12/json-schema-core>.

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/