

# Specific RPP design challenges

~~IETF 122 Bangkok, Mar 2025~~

IETF 123 Madrid, Jul 2025

Pawel Kowalik, DENIC eG



# Warning

The following slides contain some of typical design problems with RESTful APIs,  
where people tend to have strong opinions...

# Create: POST vs. PUT dilemma

- PUT is appropriate to create a resource with a known URL
  - /domains/foo.com is a known URL
  - PUT gives some guarantees of idempotency, e.g. sending twice the same representation one would still get the same registered domain (unless something happens in between – same as with EPP)
- PUT also allows for full-replacement update
- PUT will also work for host objects
- For contacts it would depend if ID is client assigned or server assigned
  - If client assigns the ID, then PUT is still appropriate
  - If server assigns the ID, then POST on a collection would be appropriate
- **Challenge:** use POST everywhere for convenience, or PUT where appropriate?
- **Challenge 2:** is the ambiguity of PUT (create vs. update) good or bad actually?

# Modelling "check" command

Initial proposal from draft-wulink-rpp-core:

```
HEAD /domains/foo.example
```

- HEAD SHOULD be actually same as GET, just skipping the response body. Some frameworks make it just like that implicitly.
  - Even worse: some frameworks would actually execute the whole GET handler and just cut off the payload
- RFC5730: The EPP <check> command is used to **determine if an object can be provisioned within a repository**. It provides a hint that allows a client to anticipate the success or failure of provisioning an object using the <create> command
  - This is likely not really obeyed to in case of many EPP extensions just overloading check with other semantics
- The equivalent would be PUT/POST with **Expect HTTP Header**. Still not same, as EPP Check would only require domain name, not the whole representation. Close enough anyway?

# Modelling "check" command

- **Challenge 1:** is Expect header well known and used?
- **Challenge 2:** Expect seems not to be sent from browser (neither XMLHttpRequest nor Fetch API). **Against R14.3.**

**During hackathon we arrived to have:**

HEAD /domains/foo.example/availability -> just quick 200/404

GET /domains/foo.example/availability -> also 200/404 with more information in the payload

# Processing resources

In RESTful design there is only limited set of standard interface verbs. If it's not enough one can either go into RCP-style overloaded POST, or decompose the resource model to define "processing resources".

Example - Transfers:

PUT /domain/example.com/transfer (EPP op="request")

GET /domain/example.com/transfer (EPP op="query")

PUT /domain/example.com/transfer/approval (EPP op="approve")

PUT /domain/example.com/transfer/rejection (EPP op="reject")

DELETE /domain/example.com/transfer (EPP op="cancel")

# Processing resources (2)

Renewal:

PUT /domain/example.com/renewal

Going further... shall all “processes” be represented as resources, especially in “pending” state?

pendingRenew will allow for:

GET /domain/example.com/renewal

DELETE /domain/example.com/renewal

# Processing resources (3)

pendingCreate

GET /domain/example.com/creates

DELETE /domain/example.com/creates

... and so on for Update, Restore, Delete

If there is no process or no support, 404 is an appropriate answer.

A bonus: OPTIONS HTTP verb could then tell the client whether the process can be cancelled or not.

# Processing resources (4)

Grouping all processing resources - because they may be extensible so let's encapsulate the URL path namespace:

POST /domain/example.com/**processes**/renewals

POST /domain/example.com/**processes**/transfers

POST /domain/example.com/**processes**/transfers/rejection

DELETE /domain/example.com/**processes**/transfers/latest

...

POST /domain/example.com/**processes**/**consolidate**

# Processing resources (5)

In resource representation - write:

PUT /domains/example.com

```
{
  "@type": "Domain",
  "name": "example.com",
  "processes": {
    "creation": {
      "period": "P2Y"
      ...
    }
  }
  ...
}
```

# Processing resources (6)

In resource representation – read :

GET /domains/example.com

```
{
  "@type": "Domain",
  "name": "example.com",
  "processes": {
    "transfer": {
      "status": "... "
    }
  }
  ...
}
```

# Authcode

Authcode is an authorization information for certain use cases:

- Reading objects of different owner
- Transferring objects
- EPP keeps it in the payload... but for HTTP GET there is no payload
  - Query string is not appropriate for any confidential information
- Header
  - Own header, like RPP-Authcode ?
  - Register Authentication Scheme Name “rpp” and have potentially 2 “Authorization” headers? -> **this is forbidden rfc9110 5.2**

# Authcode (2)

Outcome of hackathon discussion:

- RPP-Authentication header:

```
RPP-Authentication: <method> <credential>
```

- <method> is an extension point
  - “authInfo” for EPP compatible pw credential
- we define <credential> as RFC8941 dict

```
RPP-Authentication: authInfo authInfo=TXkgU2VjcmV0IFRva2Vu,  
roid: REG-XYZ-12345
```

# Optional properties vs. JSON null

OpenAPI / JSON Schema definition of an optional field:

```
{
  "$schema": "https://json-schema.org/draft-07/schema",
  "type": "object",
  "properties": {
    "required": {
      "type": "string"
    },
    "not required": {
      "type": "string"
    }
  },
  "required": [
    "required"
  ]
}
```

```
{
  "required": "foo"
}
```

```
{
  "required": "foo",
  "not required": null
}
```

not required	must be string
--------------	----------------

# Optional properties vs. JSON null (2)

What would a framework do?

```
class test:  
    required: str  
    not_required: Optional[str] = None
```



```
{  
    "required": "foo",  
    "not_required": null  
}
```

Would optional field value null be same as field missing?

**If time allows...**



# Rich queries

- EPP Command-Response Extension type allows for large payloads in the commands, also for Info Command, which would be mapped to HTTP GET
  - In practice large requests are with CHECK command of multi-domains
- Challenge: HTTP GET would rather not contain body
- Possible approaches:
  - KISS - for single item use query string and assume it's long enough for real life use-cases
  - for multi-target (bulk) queries define dedicated end-points
  - HTTP QUERY
    - [draft-ietf-httpbis-safe-method-w-body](#)
  - Special resource like \_query and POST
  - overloaded POST on collection (with different media type)

# Hypermedia or not hypermedia

POST /domains/ HTTP/1.1

Host: rpp.example.com

Content-Type: application/rpp+json

```
{  
  "name": "example.example.com",  
  ...  
  "registrant": [  
    "CTC42977"  
  ]  
  ...  
}
```

# Hypermedia or not hypermedia (2)

POST /domains/ HTTP/1.1

Host: rpp.example.com

Content-Type: application/rpp+json

```
{  
  "name": "example.example.com",  
  ...  
  "registrant": [  
    "contacts/CTC42977"  
  ]  
  ...  
}
```

# Hypermedia or not hypermedia (3)

POST /domains/ HTTP/1.1

Host: rpp.example.com

Content-Type: application/rpp+json

```
{
  "name": "example.example.com",
  ...
  "links": [
    {
      "href": "contacts/CTC42977",
      "rel": "registrant"
    }
  ]
  ...
}
```

# Hypermedia or not hypermedia (4)

POST /domains/ HTTP/1.1

Host: rpp.example.com

Content-Type: application/rpp+json

```
{
  "name": "example.example.com",
  ...
  "links": [
    {
      "href": "https://rpp.foo.example/contacts/CTC42977",
      "rel": "registrant"
    }
  ]
  ...
}
```

# Hypermedia or not hypermedia (5)

POST /domains/ HTTP/1.1

Host: rpp.example.com

Content-Type: application/rpp+json

```
{
  "name": "example.example.com",
  ...
  "links": [
    {
      "href": "https://rpp.foo.example/contacts/CTC42977",
      "id": "CTC42977",
      "rel": "registrant"
    }
  ]
  ...
}
```

**Questions?**



# Thank you

- Contact: [pawel.kowalik@denic.de](mailto:pawel.kowalik@denic.de)
- LinkedIn: <https://www.linkedin.com/in/pawelk/>

