RPP – JSON Update

Maarten Wullink - SIDN Labs

Pawel Kowalik - DENIC

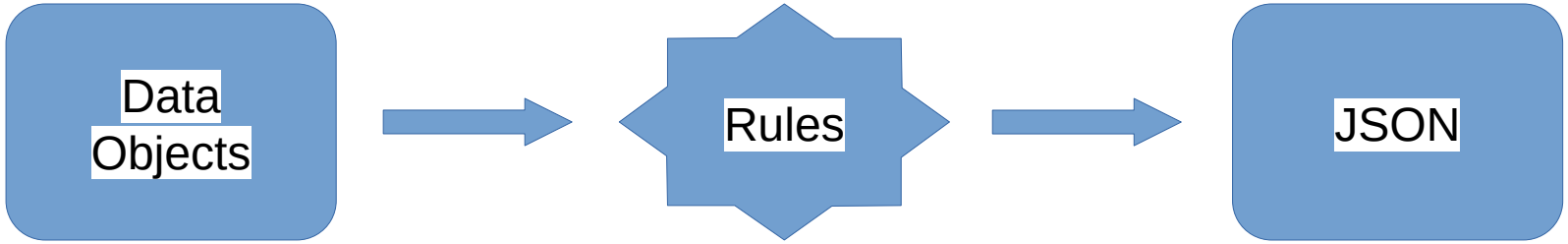
IETF 125, Shenzen, 2026

Updates - 01

- Changed mapping source from EPP XML to RPP Data Objects
- Created JSON Schemas for validation
- Added many examples

Mapping

- The previous version used XML schemas from RFC573[0-3] as source for mapping to JSON
- The new -01 draft now uses the RPP Data Objects[1]
- Introduces set of rules for creating JSON Schema and representations



[1] https://datatracker.ietf.org/doc/draft-kowalik-rpp-data-objects/

JSON Schema

- JSON schema used for validation of request and response messages
- Schemas for:
    - Shared common objects
    - High level resources
- Not all business rules can be captured in JSON schema
    - e.g. RGP date properties are dependent on current process state
- Strict validation of resources (unevaluatedProperties": false)

Mapping – Basic Types

- JSON typing is limited, JSON Schema has richer type system

RPP Primitive Type	JSON Type	Notes
String	`string`	Unicode character sequence
Integer	`number`	Whole number, positive or negative
Boolean	`boolean`	`true` or `false`
Decimal	`string`	Base-10 fractional value with exact representation within a defined precision. Number is encoded into string same as `"number"` in [@!RFC8259] without exponent part `"ext"`.
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]
Binary	`string`	Base64-encoded binary data

Mapping – Cardinality

- 1 (exactly one) → required property

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

- 0-1 (zero or one) → optional property

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

SIDN LABS

Mapping – Cardinality

- 0+ (zero or more) → optional array

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

- 1+ (one or more) → required array (minItems=1)

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

Associations

- Aggregation (relationship between independent objects)
  - Outer object uses unique identfier to reference inner objects

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

Associations

- Composition (relationship between parent and child)
  Child lifecycle is bound to the parent

```
{
        "@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
                    }
                ]
            }
        ]
}
```

Labelled - Associations

- Labelled aggregation and composition.
- Fixed “label” and “object” properties
- Can have duplicate “label” values (e.g. 2 techc)

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

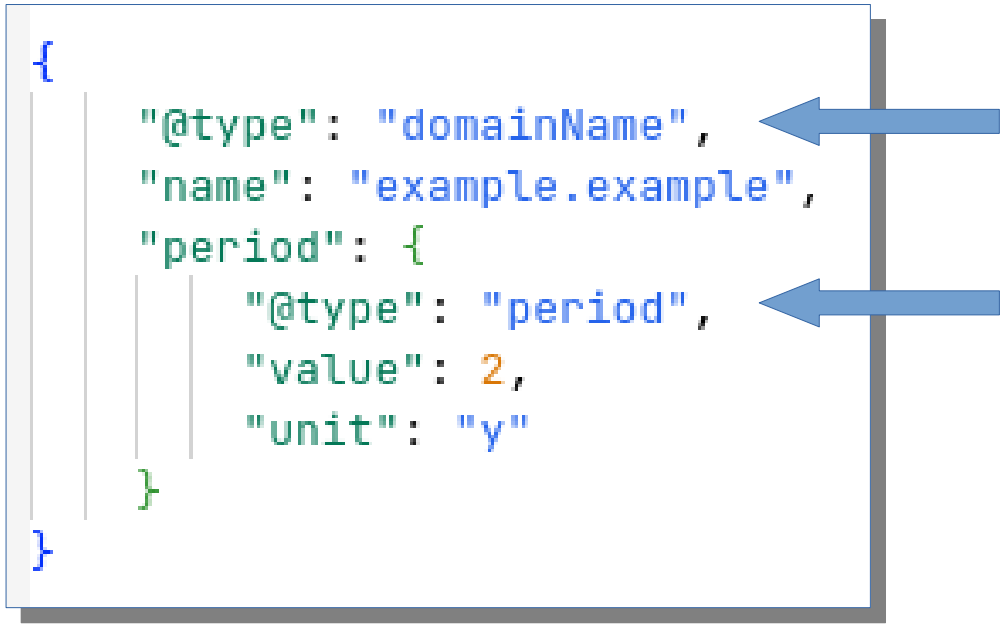
Dictionary - Associations

- Dictionary aggregation and composition.
- Unique name for relation

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

Objects - Type

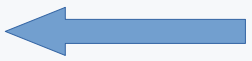
- JSON-LD style semantic typing using @type
  - Helpful when JSON Schema cannot validate everything?
  - Do we need it?



Objects – Required data

- All properties are optional by default
  - Strict validation using “required” property

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



Shared Objects

- Designed to be reused by other objects
- Examples objects include:
    - ClientIdentifier
    - Period
    - Status
    - PostalInfo

Shared Objects

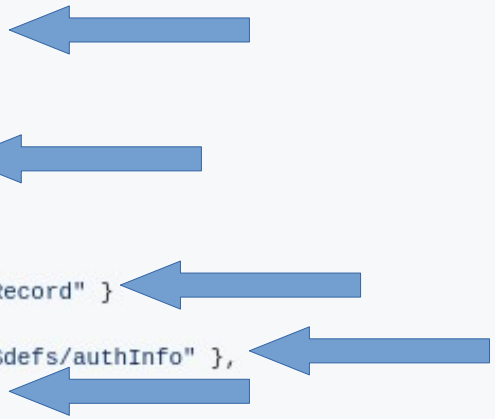
- Can be reused by other objects
- Declared using “$defs” for lookup and “@type” for object type

```
{
  "$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"]
    }
  }
}
```

Resource Objects

- Provisionable objects that include common shared objects
- Domain create schema:

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "type": "object",
  "properties": {
    "@type": { "type": "string", "const": "domainName" },
    "name": { "type": "string" },
    "registrant": { "type": "string" },
    "contacts": {
      "type": "array",
      "items": { "$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
}
```



Resource Objects

- Domain create JSON example:

```
{
    "@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"
    }
}
```

Questions - Profiles

- Profiles [1] simplify communicating capabilities and requirements
- But, make validation more challenging
- Introduces additional validation requirements per implementation

For example, a profile may change optional property to required property?

- Custom change of RPP provided schema?
- Do additional checks in implementation logic?

[1] https://datatracker.ietf.org/doc/draft-wullink-rpp-core/

SIDN LABS

Questions – Contact Data

- How do we represent contact data?
  - More EPP-like as now?
  - Try to fit in JSContact?

Next Steps

- Sync with latest version of Data Objects Draft (already being done)
  - Update process interface (Transfer/Restore)
- Ask to adopt as wg document


SIDN.nl


@SIDN


SIDN

Thank You

www.sidnlabs.nl | stats.sidnlabs.nl