Extending RPP

Maarten Wullink (SIDN Labs), Pawel Kowalik (DENIC)

IETF 127, San Francisco

Extensions

- **Additive**: an extension never removes, redefines, constrains or changes the definitions of a base object
- **Independent**: extensions can be combined per deployment, but must not depend on another
- **Identified**: standardised extensions use `urn:ietf:params:rpp:extension:<name>`, registered with IANA; private ones use a unique reverse-domain identifier
- **Discovered**: the server lists every supported extension in Discovery

Extension Points

Extension point	What an extension can add
Component objects	Shared components reused by objects and operations
Data objects	New objects, or extend existing ones
Process objects	New or extended; must extend the Base Process Object
Data elements	Properties and associations on existing objects
Operations	New operations; extra inputs, outputs,
External data types	A whole type from another specification (e.g. JSContact)
HTTP endpoints	Derived from data objects
Query parameters	Derived from operation parameters
HTTP headers	New headers for existing operations
Result codes	New codes
Problem details	New propertiess in the details object

Extension Guidelines

- New I-D draft-wullink-rpp-extension-guidelines (BCP) [1]
- Provides guidance on what extension points to use
- Contains a template for extension designers
    - Consistent format for extensions
- Extension Mapper key for JSON (“rpp:extends”)

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

Choose the simplest extension point

Step	Form	Impact
1	Reuse an open-ended element, for example new status labels	Lowest: list the values in IANA Considerations
2	Add data elements to an object, or input and output parameters to an operation	Contained in one object or operation
3	Define a new operation, or a process object	New behaviour on new or existing objects
4	Define a new data object	New resource and endpoint
5	Add protocol-level elements: headers, query parameters, result codes, problem fields, Discovery properties	Can have a larger impact on implementations

- Check the IANA RPP Extensions and Data Object registries first and reuse an existing extension
- An extension may combine extension points, but should use the simplest combination

JSON extension mechanism: rpp:extends

```
{
  "$id": "https://rpp.example/schemas/ext-domain-extproperty1.json",
  "rpp:extends": {
      "https://rpp.example/rpp/schema.json#/$defs/domainName.read": "#/$defs/ext.domainName.read"
  },
  "$defs": {
    "ext.domainName.read": {
      "type": "object",
      "properties": { "extProperty1": { "type": "string" } }
    },
    ...
  }
}
```

- ”rpp:extends" maps each base definition, by its full reference, to the local definition
- One schema document per extension, with its own top-level $id; local definitions use the prefix "ext"
- The base definition is always referenced, never copied

Base + extension(s) = effective schema

```
base schema                  extension.
rpp/schema.json              ext-domain-extproperty1.json
domainName.read              ext.domainName.read
      |                              |
      +----------  allOf ----------|
                     |
            effective.domainName.read
```

- 1. The deployment policy determines which extensions it supports, announced in Discovery
- 2. For each base definition, find the extension schemas to apply from "rpp:extends"
- 3. Compose with allOf: a $ref to the base definition and one to each extension definition
- 4. The result has its own $id, different from the base and extension schemas; prefix "effective" is recommended

An effective schema for two extensions

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "https://rpp.example/rpp/effective/domain.read.json",
  "$ref": "#/$defs/effective.domainName.read",
  "$defs": {
    "effective.domainName.read": {
      "allOf": [
        { "$ref": "https://rpp.example/rpp/schema.json#/$defs/domainName.read" },
        { "$ref": "https://rpp.example/schemas/ext-domain-extproperty1.json#/$defs/ext.domainName.read" },
        { "$ref": "https://rpp.example/schemas/ext-domain-extproperty2.json#/$defs/ext.domainName.read" }
      ]
    }
  }
}
```

- Adding or removing an extension changes the allOf list
- The result is one self-contained schema a client can validate against

Validation and consumer implementation

Validation

- Undeclared properties must be rejected
- Any supported combination of registered extensions validates
- A deployment without an extension rejects its properties, if used by client

Consumers

- Client builds JSON validator for extension, composed based on discovered set of extensions

Open points and feedback wanted

- Do we need additional extension points, not yet included?
- Should we allow a extension to extend other extensions?
    - This violates the extension independence rule

Questions and comments welcome, on the RPP list and on GitHub


SIDN.nl


@SIDN


SIDN

Thank You

www.sidnlabs.nl | stats.sidnlabs.nl

SIDN LABS