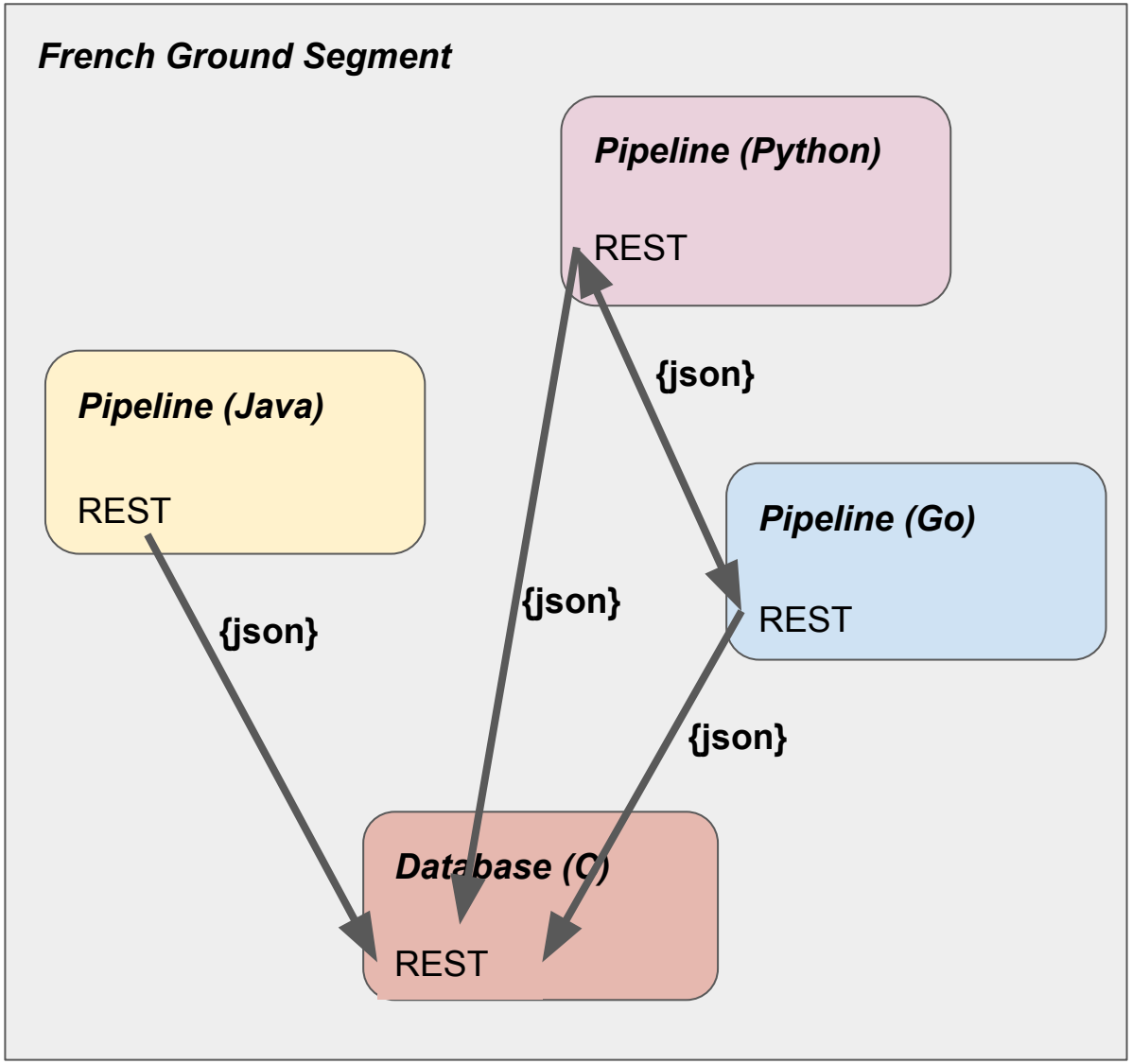
Validating JSON Messages with JSON-Schema

Use Case: SVOM: a GRB Monitor

- **Heterogeneous system**
  - Multiple Agents
  - Multiple institutes
  - Multiple languages
- **Continuous integration**
  - **JSON messages must be tested against specified schemes**



JSON Schema Home Page

JSON Schema

JSON Schema is a vocabulary that allows you to **annotate** and **validate** JSON documents.

Advantages

JSON Schema

- describes your existing data format
- clear, human- and machine-readable documentation
- complete structural validation, useful for
  - automated testing
  - validating client-submitted data

JSON Hyper-Schema

- describes your existing API - no new structures required
- links (including URI Templates for target URIs)
- forms - specify a JSON Schema for the desired data

Useful for the VO?

Screenshot http://json-schema.org/

What is a JSON Schema

- **A JSON schema is a JSON object**
  - JSON-schema components are also JSON-schemes
  - Controlled vocabulary
  - Compliant with a meta-schema

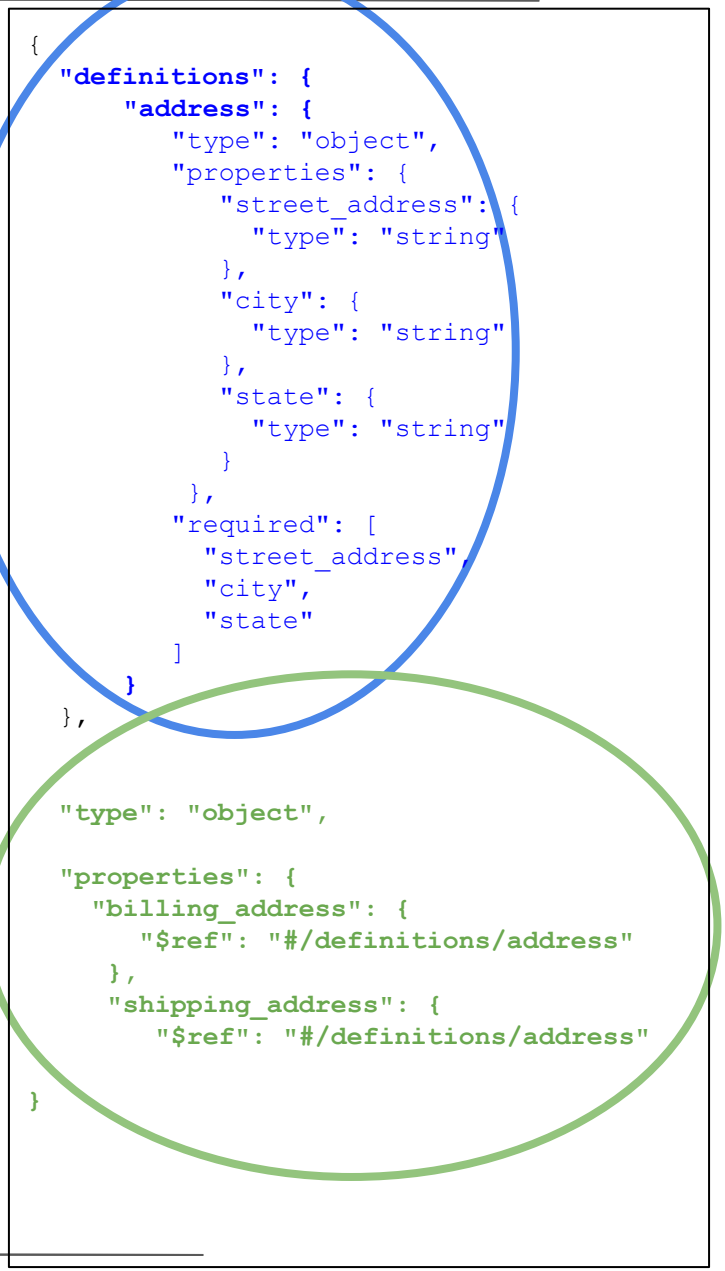
- **Can be interpreted by a processor to validate message contents**
  - Simple type validation
  - Complex type validation

Schema	Valid	Not valid
{ }	*Any JSON entity*	nothing
{ "type": "string" }	“This is a string”	45 { “key”, “this is a string”}

Typical Schema Structure

- **Dictionary of definitions**
  - Components

- **Root object structure**
  - Object or array



```
{
  "definitions": {
      "address": {
         "type": "object",
         "properties": {
            "street_address": {
              "type": "string"
            },
            "city": {
              "type": "string"
            },
            "state": {
              "type": "string"
            }
          },
         "required": [
           "street_address",
           "city",
           "state"
         ]
      }
  },

  "type": "object",

  "properties": {
    "billing_address": {
       "$ref": "#/definitions/address"
     },
     "shipping_address": {
        "$ref": "#/definitions/address"

}
```

Validation Scope

- **Validated properties**
  - Properties present in both message and schema are validated.
  - This rules can be overridden with the **`required`** operator

```
{
  "definitions": {
    "address": {
      "type": "object",
      "properties": {
        "street_address": { "type": "string" },
        "city":           { "type": "string" },
        "state":          { "type": "string" }
      },
      "required": ["street_address", "city”]
    }
  }
```

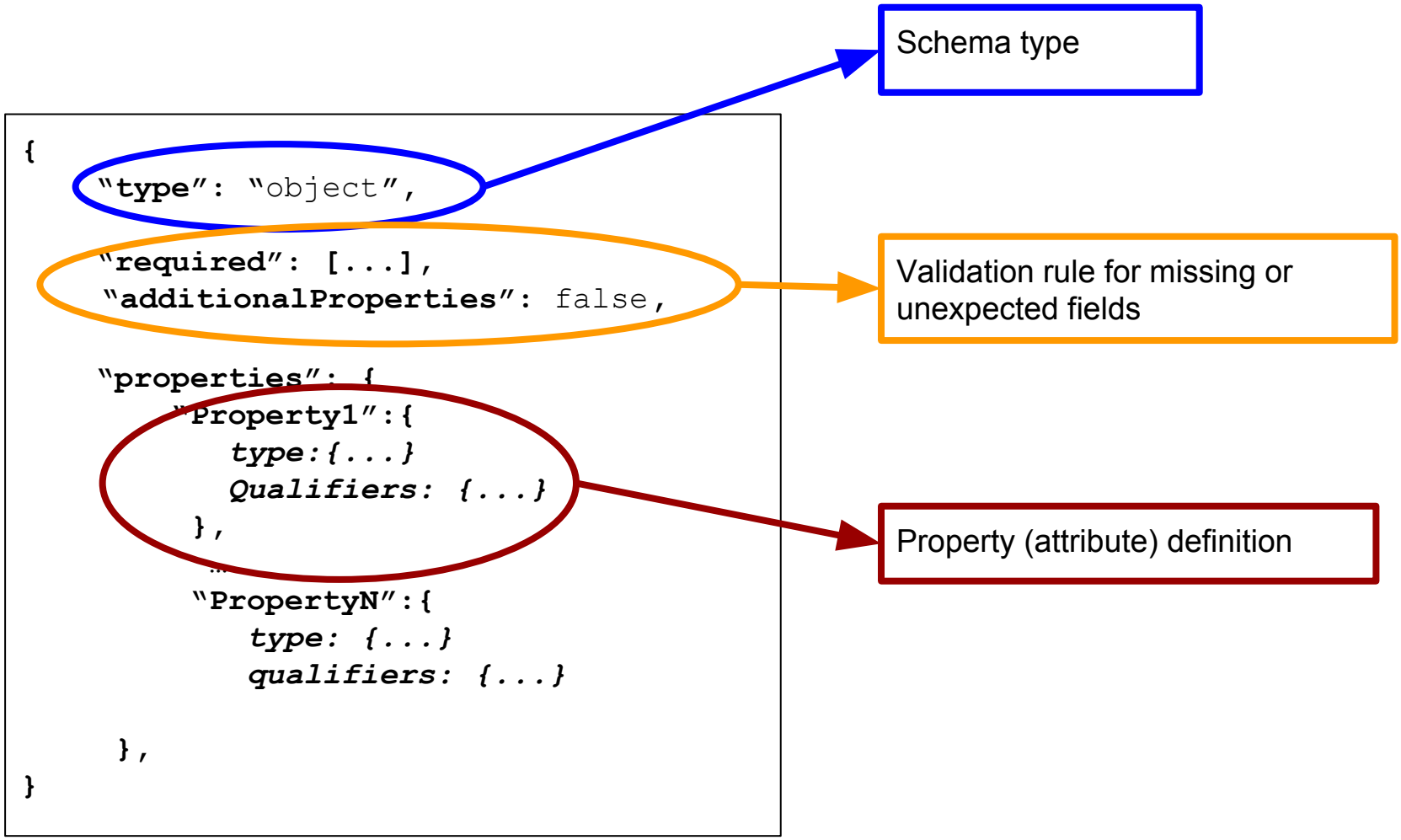
- **Ignored properties**
  - Properties not defined in the schema are ignored
  - This rules can be overridden with the **`additionalProperties: false/true`** operator.

References

`"properties": {` `  "votable": {` `  "$ref": "#/definitions/typevotable"` `  }` `}`	The ***votable*** property is described by the content of the ***typevotable*** key of the ***definitions*** object located at root of the current document
`"properties": {` `  "votable": {` `  "$ref":` `"http://other/schema/#/definitions/typevotable"` `  }` `}`	The ***votable*** property is described by the content of the ***typevotable*** key of the ***definitions*** object located at root of the document located by the URL.
`"properties": {` `  "votable": {` `  "$ref": "#typevotable"` `  }` `}`	The ***votable*** property is described by the content of the schema element having ***typevotable*** as identifier: ***"$id": "typevotable"***
`"properties": {` `  "votable": {` `  "$ref":` `"/var/schemas/other.schema/#/definitions/typevotable"` `  }` `}`	The ***votable*** property is described by the content of the ***typevotable*** key of the ***definitions*** object located at root of the document located at ***/var/schema s/autre.schema***

- **Allows to build schemas with reusable components**
  - No name-space issue

Object Schema



Array Description

```
{
      "type": "array",
      "items": {
          "oneOf":  [
          {
              "type": "string"
          },
          {
              "type": "integer"
          }
         ]
      }
}
```

- **Array example**
  - This array must contain either strings or integers
  - `Items` can also contain complex types (objects)

Supported Types

- **Entities declared in a schema can have specified types**
  - Keyword: `"type": "one of the predefined types"`

`null`	Objet null
`array`	Array of entities
`boolean`	
`number`	Base 10 numerical
`integer`	Integer value
`string`	String (ASCII or unicode)
`object`	JSON object
`enum`	Array of possible values

Types Qualifiers

Qualifier	Type	Scope	Role
`min/maxItems`	int	array	Limits for an array size
`uniqueItems`	bool	array	Require the element unicity
`pattern`	regexp	string	Regular expression validating a string
`min/maxLength`	int	string	String length limits
`format`	vocab	string	• "date-time": Date representation, as defined by RFC 3339, section 5.6. • "email": Internet email address, see RFC 5322, section 3.4.1. • "hostname": Internet host name, see RFC 1034, section 3.1. • "ipv4": IPv4 address, according to dotted-quad ABNF syntax as defined in RFC 2673, section 3.2. • "ipv6": IPv6 address, as defined in RFC 2373, section 2.2. • "uri": A universal resource identifier (URI), according to RFC3986. • Lots of suggestions
`min/maximum`	num	numerical	Limits for a numerical value
`multipleOf`	int	numerical	Value must be multiple of (>0)

Choice Selectors

- **`oneOf/anyOf/allOf` operators**
  - Operator combinations are not necessarily consistent
  - Can be reverted with the **`"not"`** operator

Operator	Key/Value context	Array context
`oneOf`: One and only one constraint must be valid `anyOf`: At least one constraint must be valid `allOf`: All constraints must be valid	`{` `  "oneOf": [` `  { "type": "number",` `  "multipleOf": 5 },` `  { "type": "number",` `  "multipleOf": 3 }` `  ]` `}` `{` `  "allOf": [` `  { "type": "number",` `  "multipleOf": 5 },` `  { "type": "number",` `  "maximum": 3 }` `  ]` `}`	`{` `  "oneOf": [` `  "type": [` `  "string", "boolean"` `  ],` `  { schéma1 },` `  { schema2 }` `  ]` `}`

Referring to a Schema in HTTP Responses

- **The schema validating a message is not referenced within the message.**
  - No header of META element in JSON

- **HTTP context:**
  - must be referenced in the header
    - Link key in the header

      ```
      Link: <http://example.com/my-hyper-schema#>; rel="describedby"
      ```
    - Profile attached to the MIME type

      ```
      Content-Type: application/json;
                    profile="http://example.com/myschema.json"
      ```

- **No HTTP context:**
  - Ad Hoc solution
  - Mentioning the schema in the service documentation
    http://json-schema.org/latest/json-schema-core.html#rfc.section.10

The JSON-Schema: an Active Project

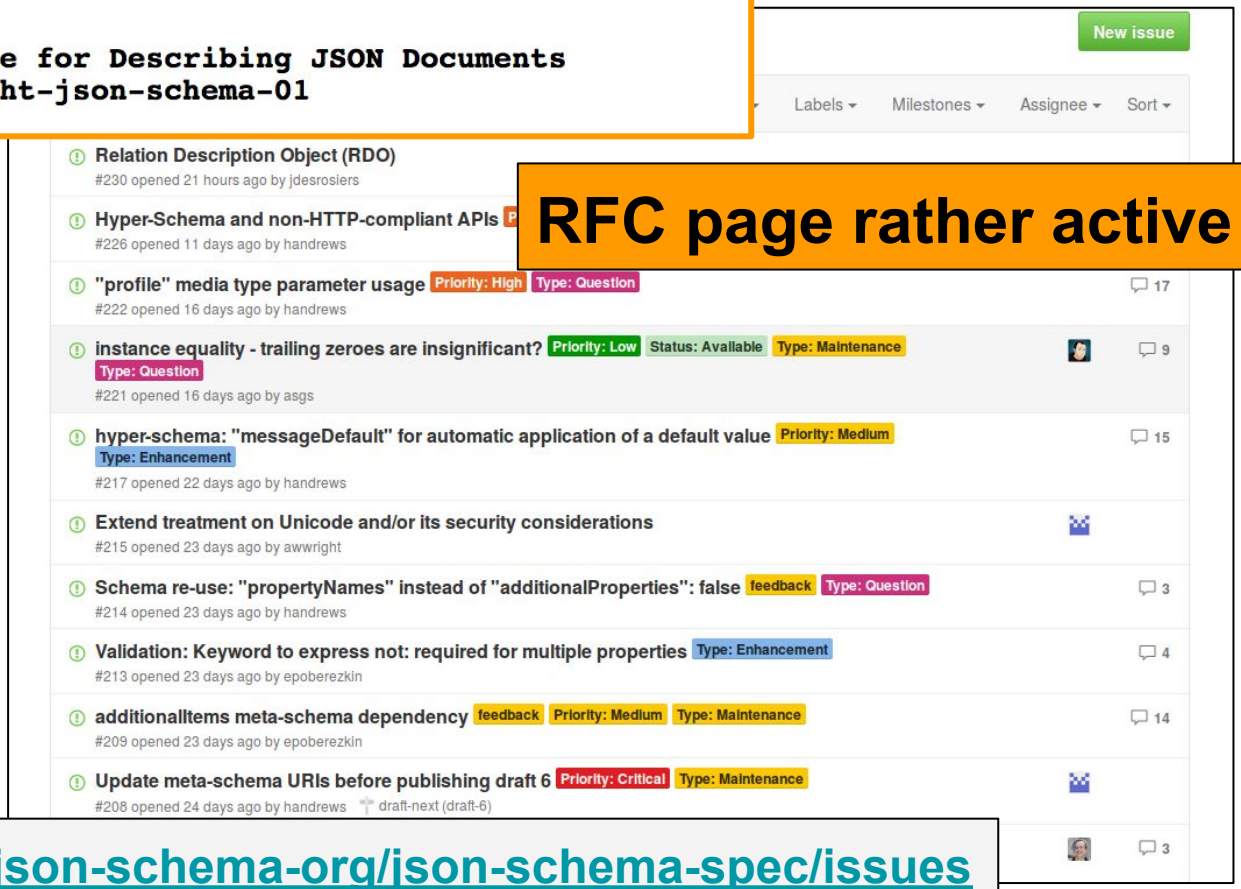
Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: October 17, 2017

A. Wright, Ed.

H. Andrews, Ed.
Cloudflare, Inc.
April 15, 2017

IETF draft V6

JSON Schema: A Media Type for Describing JSON Documents
draft-wright-json-schema-01



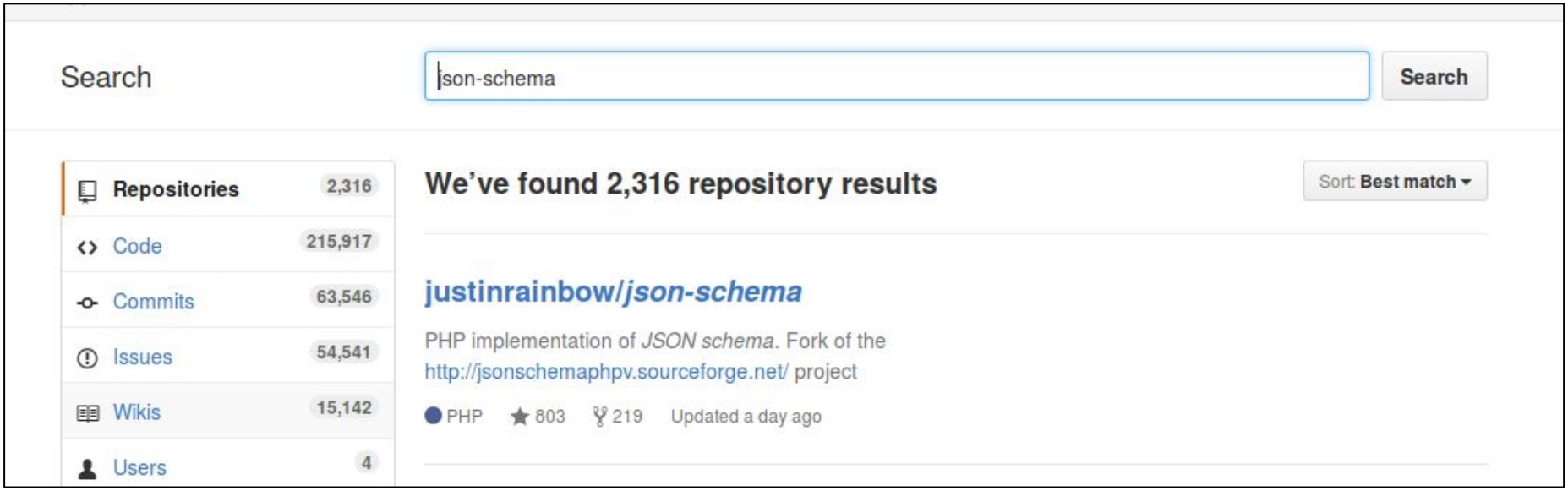
RFC page rather active

https://github.com/json-schema-org/json-schema-spec/issues
http://json-schema.org/latest/json-schema-core.html

The JSON-Schema an Active Project

Github: Large number of tools in different languages

- Editors, validators, translators,tutorials...
- Language barycenter somewhere in the JS world



JSON-SCHEMA and the VO

- **Whay defining JSON message formats**
  - **Easier parsing**
    - In most of the languages
  - **Facilitating the client job**
    - Data transport
      - Complementary to the VOTable schema
    - Registry responses
  - **Validation tools**
    - Just a feeling, no investigation right now
- **Different possible approaches**
  - As an XML schema facet
    - JSON-schema = translation(XML schema)
  - Developing specific JSON-Schemes
  - Using schema components

Translating XML Schema

- **Translating XSD into JSON-schema**
  - (+) No need of schema duplication in standards
  - (-) No evidence that the XSD -> JSON translation is always possible
  - (-) No real interest if the processing of JSON message is as complex as the VOTable parsing.
- **Some translation issues**
  - The key order is not preserved in JSON
    - Only arrays keep the order
  - Duplicate keys are not supported (`<INFO>`)
  - There is not attribute attached to keys (`@ucd, @unit,...`)
- **Tested translators**
  - Basic solution (Web tutorial)
    - **JAXb** -> **POJO objects** -> **Jackson** -> **JSON schema**
      - Render data classes but not the semantic rules
  - XSD2JSON: The better I found (https://github.com/fnogatz/xsd2json)
    - Written in Prolog
      - using an AI language denotes the need of inferences
    - `<choice>` not supported
      - Discussion with the developer stalled
  - Lots of tools in the JS/NodeJS world

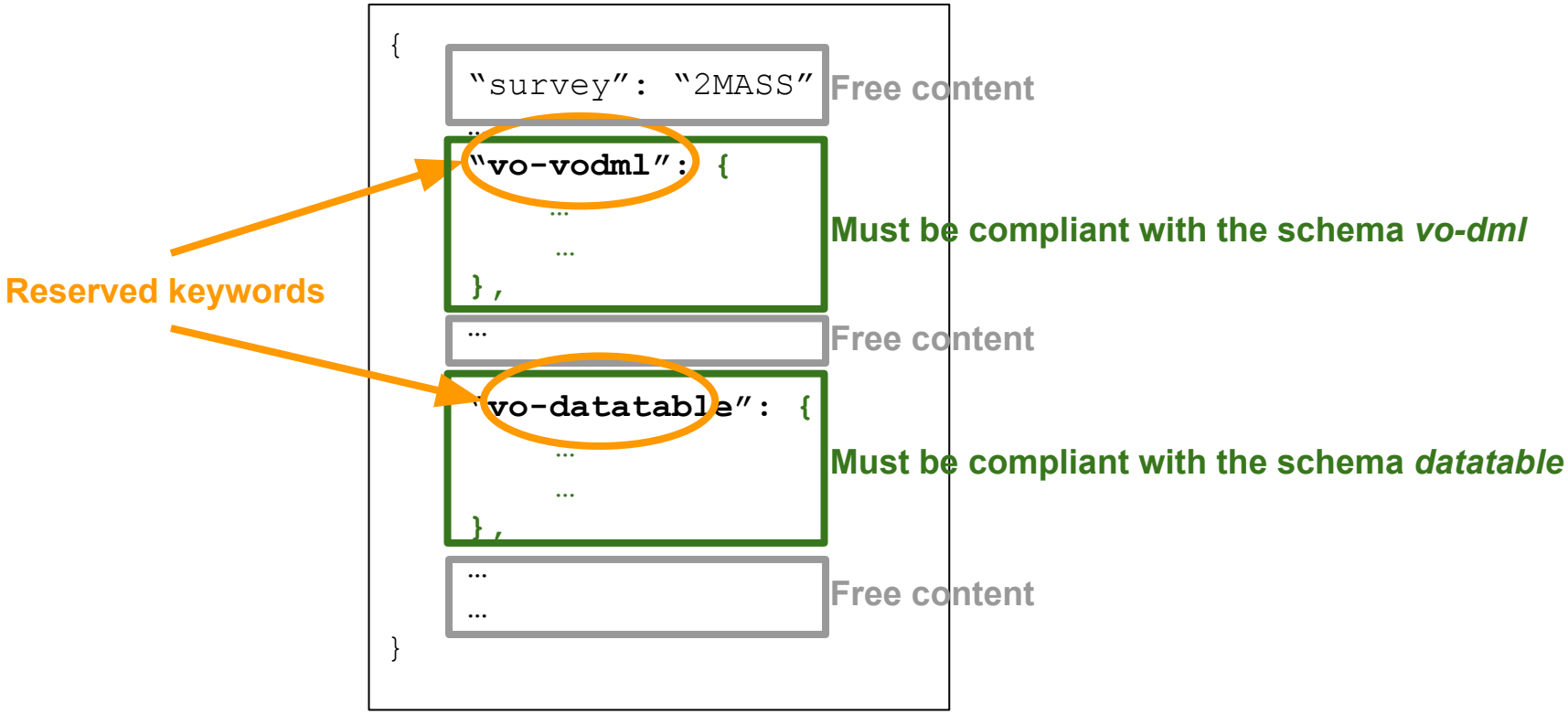
Writing specific JSON-SCHEMA

- **Developing specific schemes?**
  - (+) Simpler schemas for different scopes
    - display, storage…
  - (-) Need to publish and support 2 different schemas

- **Large number of tools**
  - From a data set
    - http://jsonschema.net/#/
  - Online
    - http://www.jsoneditoronline.org/
  - Eclipse/EMacs/vim…
  - DSL?
  - From *TypeScript* classes

The Schema Component Approach

- A set of sub-schemas defined by a IVOA standard
- Each sub-schema is identified by a specific key
  - Prefixed by `vo-` e.g.
- A client reading one of that keys knows which schema component it has to refer to.
- A client looking for a specific complex type has just to search the relevant key
- JSON messages including formated components keep a great flexibility



JSON-SCHEMA and the VO: Validators

- **Excellent online validator**
  - http://www.jsonschemavalidator.net/
  - Standalone version available under .NET :=(
- **Others Python and Java tools tested**
  - GitHub:
    - json-schema-validator
    - python-json-schema-validator
  - Seems working
    - Need further test
- **Lots of projects in the JS/NodeJS world**
    - **To be tested**

Tutorials

- **Large number of tutorials among which are:**
  - API REST validation
    http://techblog.constantcontact.com/api/using-json-schema-to-validate-web-service-requests/
  - JSON-SCHEMA page
    http://json-schema.org/documentation.html
  - STSCI !! (Michael Droettboom)
    https://spacetelescope.github.io/understanding-json-schema/