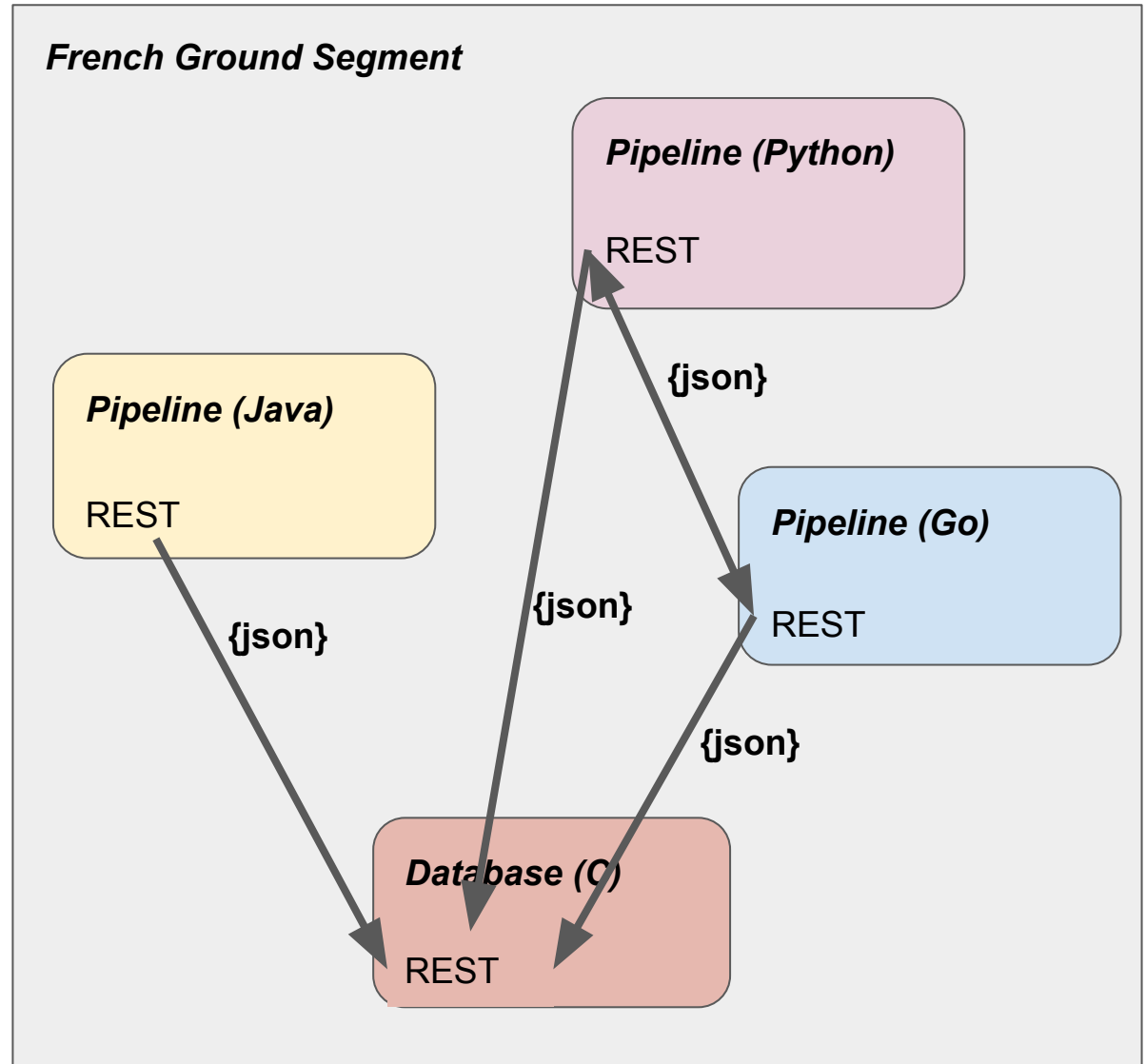

Validating JSON Messages with JSON-Schema

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 **Useful for the VO?**
- links (including [URI Templates](#) for target URIs)
- forms - specify a JSON Schema for the desired data

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 |
|-----------------------------------|------------------------|-------------------------------------|
| <code>{ }</code> | <i>Any JSON entity</i> | nothing |
| <code>{ "type": "string" }</code> | "This is a string" | 45 { "key", "this is a string" } |

Typical Schema Structure

- **Dictionary of definitions**
 - Components

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

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

```
  "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**

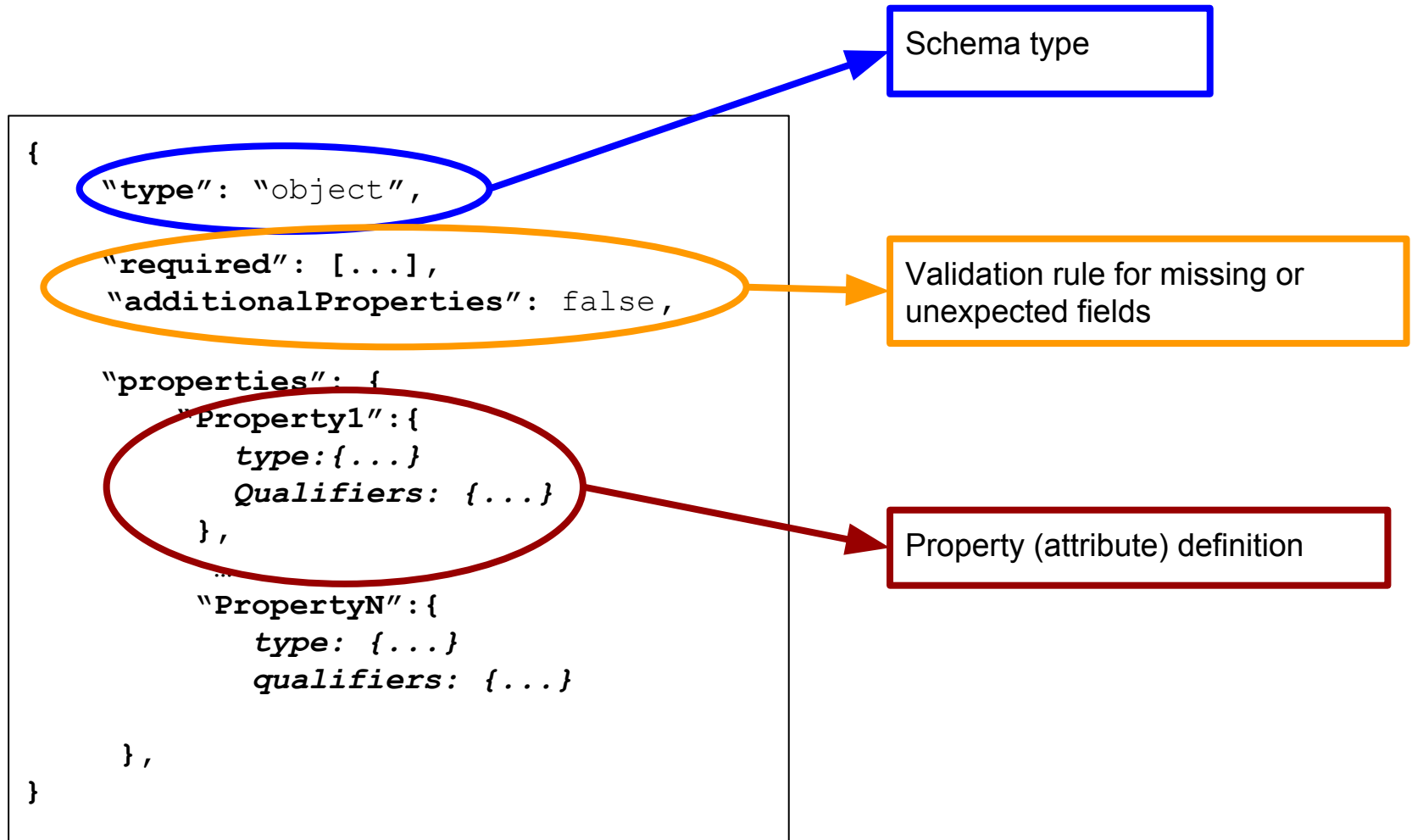
- Properties not defined in the schema are ignored
- This rules can be overridden with the **additionalProperties: false/true** operator.

References

| | |
|--|--|
| <pre>"properties": { "votable": { "\$ref": "#/definitions/typevotable" } }</pre> | The votable property is described by the content of the typevotable key of the definitions object located at root of the current document |
| <pre>"properties": { "votable": { "\$ref": "http://other/schema/#/definitions/typevotable" } }</pre> | 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. |
| <pre>"properties": { "votable": { "\$ref": "#typevotable" } }</pre> | The votable property is described by the content of the schema element having typevotable as identifier: "\$id": "typevotable" |
| <pre>"properties": { "votable": { "\$ref": "/var/schemas/other.schema/#/definitions/typevotable" } }</pre> | 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 |
|----------------------------|---------------------|------------------------|--|
| <code>min/maxItems</code> | <code>int</code> | <code>array</code> | Limits for an array size |
| <code>uniqueItems</code> | <code>bool</code> | <code>array</code> | Require the element unicity |
| <code>pattern</code> | <code>regexp</code> | <code>string</code> | Regular expression validating a string |
| <code>min/maxLength</code> | <code>int</code> | <code>string</code> | String length limits |
| <code>format</code> | <code>vocab</code> | <code>string</code> | <ul style="list-style-type: none">• "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 |
| <code>min/maximum</code> | <code>num</code> | <code>numerical</code> | Limits for a numerical value |
| <code>multipleOf</code> | <code>int</code> | <code>numerical</code> | 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 | <pre>{ "oneOf": [{ "type": "number", "multipleOf": 5 }, { "type": "number", "multipleOf": 3 }] }</pre> | <pre>{ "oneOf": ["type": ["string", "boolean"], { schéma1 }, { schema2 }] }</pre> |
| anyOf: At least one constraint must be valid | | |
| allOf: All constraints must be valid | <pre>{ "allOf": [{ "type": "number", "multipleOf": 5 }, { "type": "number", "maximum": 3 }] }</pre> | |

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

IETF draft V6

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

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

RFC page rather active

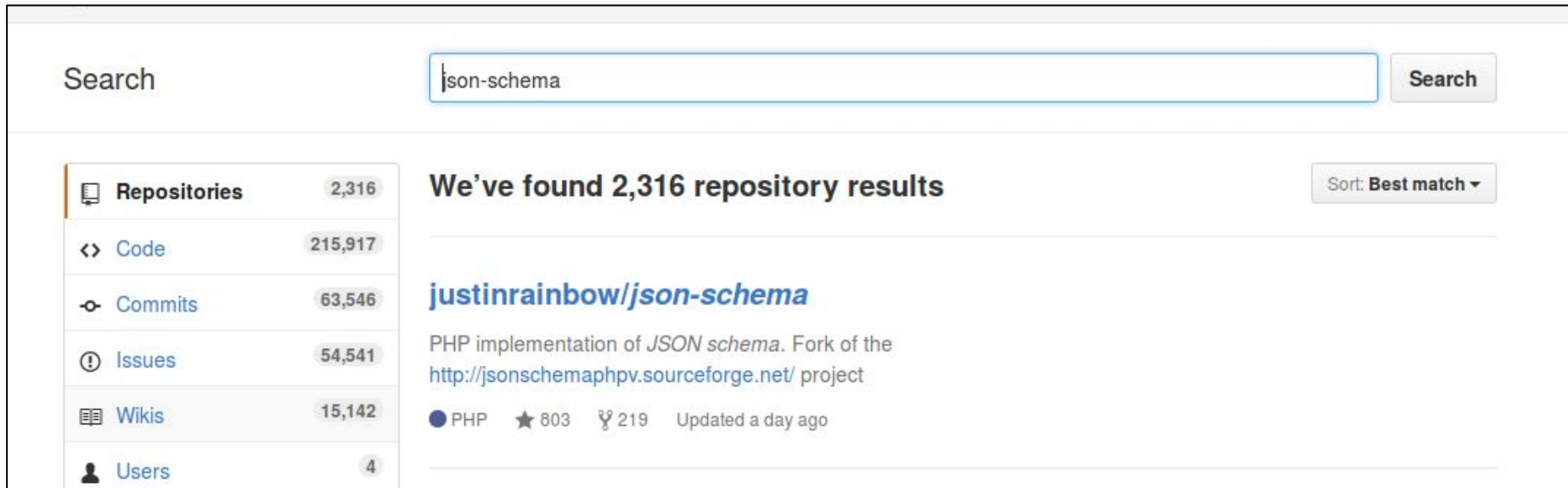
- 1 Relation Description Object (RDO) #230 opened 21 hours ago by jdesrosiers
- 1 Hyper-Schema and non-HTTP-compliant APIs #226 opened 11 days ago by handrews
- 1 "profile" media type parameter usage Priority: High Type: Question #222 opened 16 days ago by handrews
- 1 instance equality - trailing zeroes are insignificant? Priority: Low Status: Available Type: Maintenance Type: Question #221 opened 16 days ago by asgs
- 1 hyper-schema: "messageDefault" for automatic application of a default value Priority: Medium Type: Enhancement #217 opened 22 days ago by handrews
- 1 Extend treatment on Unicode and/or its security considerations #215 opened 23 days ago by awwright
- 1 Schema re-use: "propertyNames" instead of "additionalProperties": false feedback Type: Question #214 opened 23 days ago by handrews
- 1 Validation: Keyword to express not: required for multiple properties Type: Enhancement #213 opened 23 days ago by epoberezkin
- 1 additionalItems meta-schema dependency feedback Priority: Medium Type: Maintenance #209 opened 23 days ago by epoberezkin
- 1 Update meta-schema URIs before publishing draft 6 Priority: Critical Type: Maintenance #208 opened 24 days ago by handrews draft-next (draft-6)

<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



The screenshot shows a GitHub search interface. At the top, there is a search bar containing the text 'json-schema' and a 'Search' button. Below the search bar, a sidebar on the left lists various categories with their respective counts: Repositories (2,316), Code (215,917), Commits (63,546), Issues (54,541), Wikis (15,142), and Users (4). The main content area displays the search results, starting with the heading 'We've found 2,316 repository results' and a 'Sort: Best match' dropdown. The first result is for the repository 'justinrainbow/json-schema', which is a PHP implementation of JSON schema. It is a fork of the project at 'http://jsonschemaphpv.sourceforge.net/'. The repository has 803 stars, 219 forks, and was updated a day ago.

| Category | Count |
|--------------|---------|
| Repositories | 2,316 |
| Code | 215,917 |
| Commits | 63,546 |
| Issues | 54,541 |
| Wikis | 15,142 |
| Users | 4 |

We've found 2,316 repository results Sort: Best match ▾

justinrainbow/json-schema
PHP implementation of *JSON schema*. Fork of the <http://jsonschemaphpv.sourceforge.net/> project
● PHP ★ 803 🍴 219 Updated a day ago

- **Why 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?**

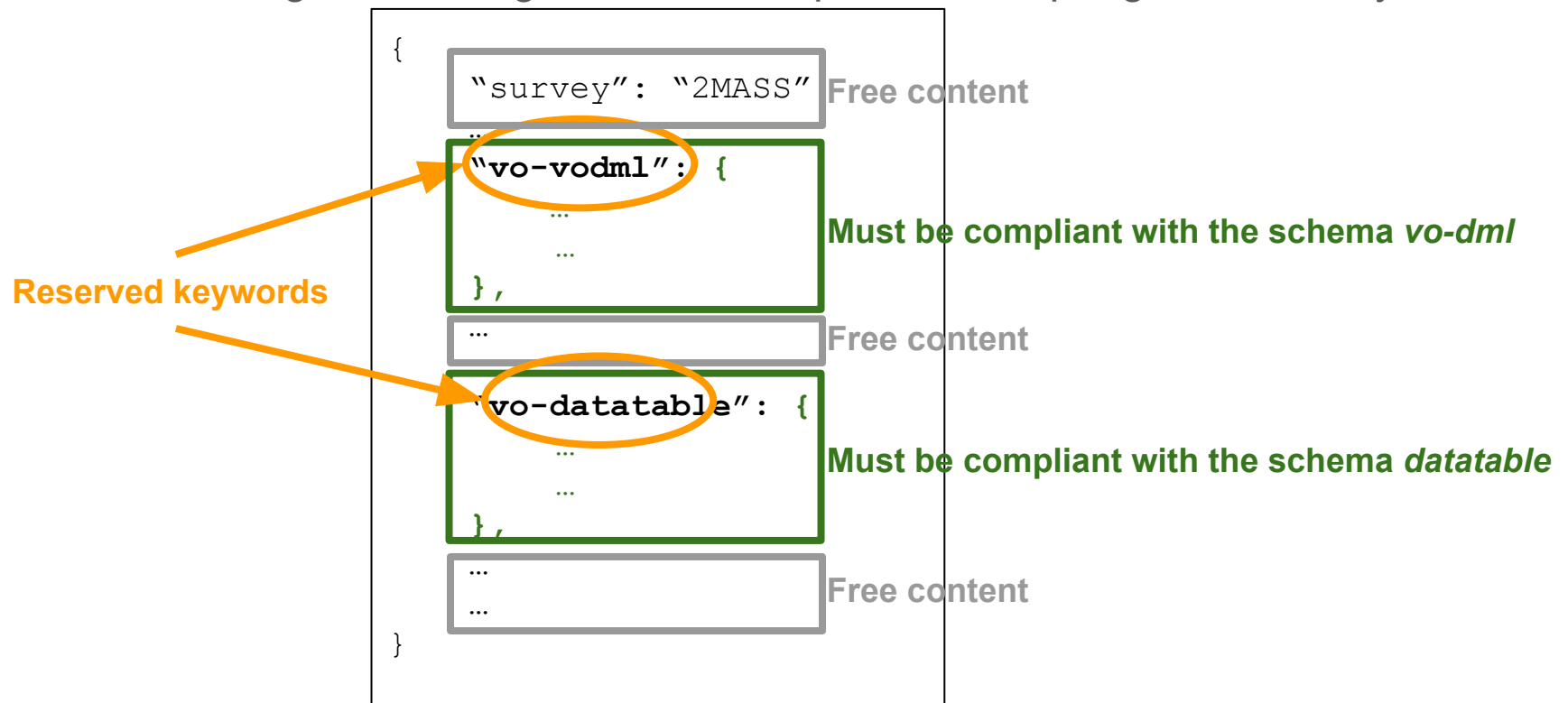
- (+) Simpler schemas for different scopes
 - display, storage...
- (-) Need to publish and support 2 different schemas

- **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 formatted 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/>