Network Working Group
Internet-Draft
Intended status: Informational
Expires: 7 May 2026

R. Polli
Digital Transformation Department, Italian Government

3 November 2025

REST API Linked Data Keywords
draft-polli-restapi-ld-keywords-07

Abstract

This document defines two keywords to provide semantic information in OpenAPI Specification and JSON Schema documents, and support contract-first semantic schema design.

About This Document

This note is to be removed before publishing as an RFC.

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-polli-restapi-ld-keywords/.

information can be found at https://github.com/ioggstream/draft-polli-restapi-ld-keywords.

Source for this draft and an issue tracker can be found at https://github.com/ioggstream/draft-polli-restapi-ld-keywords/issues.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 7 May 2026.

Copyright Notice

Copyright (c) 2025 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction

API providers usually specify semantic information in text or out-of-band documents; at best, this information is described in prose into specific sections of interface definition documents (see Section 1.2).

This is because API providers do not always value machine-readable semantics, or because they have no knowledge of semantic technologies - that are perceived as unnecessarily complex.

A full-semantic approach (e.g. writing RDF oriented APIs) has not become widespread because transferring and processing the semantics on every message significantly increases data transfer and computation requirements.

Moreover the semantic landscape do not provide easy ways of defining / constraining the syntax of an object: tools like [SHACL] and [OWL] restrictions are considered computationally intensive to process and complex to use from web and mobile developers.

This document provides a simple mechanism to attach semantic information to REST APIs that rely on different dialects of [JSONSCHEMA], thus supporting a contract-first schema design.

For example, the OpenAPI Specifications (see [OAS]) allow to describe REST APIs interactions and capabilities using a machine-readable format based on [JSON] or [YAML]. OAS 3.0 is based on JSON Schema draft-4 while OAS 3.1 relies on the latest JSON Schema draft.

1.1. Goals and Design Choices

This document has the following goals:

* describe in a single specification document backed by [JSONSCHEMA] (e.g. an OpenAPI document) both the syntax and semantics of JSON objects. This information can be either be provided editing the document by hand or via automated tools;

* easy for non-semantic experts and with reduced complexity;

* support for OAS 3.0 / JSON Schema Draft4;

while it is not intended to:

* integrate the syntax defined using [JSONSCHEMA];

* infer semantic information where it is not provided;

* convert [JSONSCHEMA] documents to RDF Schema (see [RDFS]) or XML Schema.

Thus, the following design choices have been made:

* the semantic context of a JSON object will be described using [JSON-LD-11] and its keywords;

* property names are limited to characters that can be used in variable names (e.g. excluding : and .) to avoid interoperability issues with code-generation tools;

* privilege a deterministic behavior over automation and composability;

* interoperable with the mechanisms described in Section 6.1 of [JSON-LD-11] for conveying semantic context in REST APIs.

1.2. Prosaic semantics

[JSONSCHEMA] allows to define the structure of the exchanged data using specific keywords. Properties' semantics can be expressed in prose via the description keyword.

```
Person:
  description: A Person.
  type: object
  properties:
    givenName:
      description: The given name of a Person.
      type: string
    familyName:
      description: The family name, or surname, of a Person.
      type: string
  example:
    givenName: John
    familyName: Doe
```

Figure 1: Example of JSON Schema model that provides semantic prose.

[JSON-LD-11] defines a way to interpret a JSON object as JSON-LD: the example schema instance (a JSON document conformant to a given schema) provided in the above "Person" schema can be integrated with semantic information adding the @type and @context properties.

```
{
  "@context": {
    "@vocab": "https://w3.org/ns/person#"
  },
  "@type": "Person",
  "givenName": "John",
  "familyName": "Doe"
}
```

Figure 2: Example of a schema instance transformed in a JSON-LD object.

This document shows how to integrate into a JSON Schema document information that can be used to add the @context and @type properties to the associated JSON Schema instances.

1.3. Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here. These words may also appear in this document in lower case as plain English words, absent their normative meanings.

The terms "content", "content negotiation", "resource", and "user agent" in this document are to be interpreted as in [HTTP].

The terms "fragment" and "fragment identifier" in this document are to be interpreted as in [URI].

The terms "node", "alias node", "anchor" and "named anchor" in this document are to be intepreded as in [YAML].

The terms "schema" and "schema instance" in this document are to be intepreded as in [JSONSCHEMA] draft-4 and higher.

The terms "JSON object", "JSON document", "member", "member name" in this document are to be intepreded as in [JSON]. The term "property" - when referred to a JSON document such as a schema instance - is a synonym of "member name", and the term "property value" is a synonym of "member value".

The terms "@context", "@type", "@id", "@value" and "@language" are to be interpreted as JSON-LD keywords in [JSON-LD-11], whereas the term "context" is to be interpreted as a JSON-LD Context defined in the same document.

Since JSON-LD is a serialization format for RDF, the document can use JSON-LD and RDF interchangeably when it refers to the semantic interpretation of a resource.

The JSON Schema keywords defined in Section 2 are collectively named "semantic keywords".

2. JSON Schema keywords

A schema (see [JSONSCHEMA]) MAY use the following JSON Schema keywords, collectively named "semantic keywords" to provide semantic information for all related schema instances.

x-jsonld-type: This keyword conveys an RDF type (see [RDF]) for the JSON schema instances described by the associate schema. It is defined in Section 2.1.

x-jsonld-context: This keyword conveys a JSON-LD context for the JSON schema instances described by the associate schema. It is defined in Section 2.2.

This specification MAY be used to:

* populate the @type property along the schema instance objects;

* compose an "instance context" to populate the @context property at the root of the schema instance.

The schema MUST be of type "object". This is because [JSON-LD-11] does not define a way to provide semantic information on JSON values that are not JSON objects.

The schema MUST NOT describe a JSON-LD (e.g. of application/ld+json media type) or conflicts will arise, such as which is the correct @context or @type (see Section 4.2).

Both JSON Schema keywords defined in this document might contain URI references. Those references MUST NOT be dereferenced automatically, since there is no guarantee that they point to actual locations. Moreover they could reference unsecured resources (e.g. using the "http://" URI scheme [HTTP]).

Appendix A provides various examples of integrating semantic information in schema instances.

2.1. The x-jsonld-type JSON Schema keyword

The x-jsonld-type value provides information on the RDF type of the associate schema instances.

This value MUST be valid according to the JSON-LD @type keyword as described in Section 3.5 of JSON-LD-11 (https://www.w3.org/TR/json-ld11/#specifying-the-type); it is thus related to the information provided via the x-jsonld-context keyword (see Section 2.2).

It SHOULD NOT reference an RDF Datatype (https://www.w3.org/TR/rdf11-concepts/#section-Datatypes) because it is not intended to provide syntax information, but only semantic information.

2.2. The x-jsonld-context JSON Schema keyword

The x-jsonld-context value provides the information required to interpret the associated schema instances as JSON-LD according to the specification in Section 6.1 of JSON-LD-11 (https://www.w3.org/TR/json-ld11/#interpreting-json-as-json-ld).

Its value MUST be a valid JSON-LD Context (see Section 9.15 of JSON-LD-11 (https://www.w3.org/TR/json-ld11/#context-definitions)).

When context composition (see Section 3.4) is needed, the context SHOULD be provided in the form of a JSON object; in fact, if the x-jsonld-context is a URL string, that URL needs to be dereferenced and processed to generate the instance context.

```
Place:
  type: object
  x-jsonld-context:
    "@vocab": "https://my.context/location.jsonld"
  properties:
    country: {type: string}

Person:
  x-jsonld-context: https://my.context/person.jsonld
  type: object
  properties:
    birthplace:
      $ref: "#/Place"
```

Figure 3: Composing URL contexts requires dereferencing them.

2.3. Interpreting schema instances

This section describes an OPTIONAL workflow to interpret a schema instance as JSON-LD.

1. ensure that the initial schema instance does not contain any @context or @type property. For further information see Section 4.2;

2. add the @context property with the value of x-jsonld-context. This will be the initial "instance context": the only one that will be mangled;

3. add the @type property with the value of x-jsonld-type;

4. iterate on each instance property like the following:

    * identify the sub-schema associated to the property (e.g. resolving $refs) and check the presence of semantic keywords;

    * for the x-jsonld-type, add the @type property to the sub-instance;

    * for the x-jsonld-context, integrate its information in the instance context when they are not already present;

    * iterate this process in case of nested entries.

The specific algorithm for integrating the values of x-jsonld-context present in sub-schemas into the instance context (see Section 2) is an implementation detail.

3. Interoperability Considerations

See the interoperability considerations for the media types and specifications used, including [YAML-IANA], [JSON], [OAS], [JSONSCHEMA] and [JSON-LD-11].

Annotating a schema with semantic keywords containing JSON-LD keywords (e.g. @context, @type and @language) may hinder its ability to be interpreted as a JSON-LD document (e.g. using the JSON-LD 1.1 context for the JSON Schema vocabulary (https://www.w3.org/2019/wot/json-schema#json-ld11-ctx)); this can be mitigated extending that context and specifying that Linked Data keywords are JSON Literals.

```
{ "@context": {
    "x-jsonld-context: { "@type": "@json"},
    "x-jsonld-type: { "@type": "@json"}
  }
}
```

This is generally not a problem, since a generic [JSONSCHEMA] document cannot be reliably interpreted as JSON-LD using a single context: this is because the same JSON member keys can have different meanings depending on their JSON Schema position (see the notes in the Interpreting JSON Schema as JSON-LD 1.1 (https://www.w3.org/2019/wot/json-schema#interpreting-json-schema-as-json-ld-1-1) section of [JSON-SCHEMA-RDF]).

3.1. Syntax is out of scope

This specification is not designed to restrict the syntax of a JSON value nor to support a conversion between JSON Schema and XMLSchema (see Section 2.1).

3.2. Limited expressivity

Not all RDF resources can be expressed as JSON documents annotated with @context and @type: this specification is limited by the possibilities of Section 6.1 of JSON-LD-11 (https://www.w3.org/TR/json-ld11/#interpreting-json-as-json-ld). On the other hand, since this approach delegates almost all the processing to of JSON-LD, as long as JSON-LD evolves it will cover further use cases.

3.3. Disjoint with JSON-LD

This specification is not designed to pre-process or mangle JSON-LD documents (e.g. to add a missing @type to a JSON-LD document), but only to support schemas that do not describe JSON-LD documents.

Applications exchanging JSON-LD documents need to explicitly populate @type and @context, and use a proper media type since Linked Data processing and interpretation requires further checks.

If these applications describe messages using [JSONSCHEMA] or [OAS], they need to process them with a JSON-LD processor and declare all required properties in the schema - like in the example below.

```
PersonLD:
  type: object
  required: [ "@context", "@type", "givenName", "familyName" ]
  properties:
    "@context":
      type: object
      enum:
      - "@vocab": "https://w3.org/ns/person#"
    "@type":
      type: string
      enum:
      - Person
    givenName:
      type: string
    familyName:
      type: string
```

Figure 4: A JSON-Schema describing a JSON-LD document.

3.4. Composability

Limited composability can be achieved applying the process described in Section 2.3. Automatic composability is not an explicit goal of this specification because of its complexity. One of the issue is that the meaning of a JSON-LD keyword is affected by its position. For example, the @type keyword:

* in a node object, adds an rdf:type arc to the RDF graph (it also has a few other effects on processing, e.g. by enabling type-scoped contexts);

* in a value object, specifies the datatype of the produced literal;

* in the context, and more precisely in a term definition, specifies type coercion (https://www.w3.org/TR/json-ld11/#type-coercion). It only applies when the value of the term is a string.

These issues can be tackled in future versions of this specifications.

Moreover, well-designed schemas do not usually have more than 3 or 4 nested levels. This means that, when needed, it is possible to assemble and optimize an instance context (see Section 2) at design time and use it to valorize x-jsonld-context (see Figure 8).

Once a context is assembled, the RDF data can be generated using the algorithms described in [JSONLD-11-API] for example through a library.

```
from pyld import jsonld
...
jsonld_text = jsonld.expand(schema_instance, context)
```

4. Security Considerations

See the interoperability considerations for the media types and specifications used, including [YAML-IANA], [JSON], [OAS], [JSONSCHEMA] and [JSON-LD-11].

4.1. Integrity and Authenticity

Adding a semantic context to a JSON document alters its value and, in an implementation-dependent way, can lead to reordering of fields. This process can thus affect the processing of digitally signed content.

4.2. Conflicts

If an OAS document includes the keywords defined in Section 2 the provider explicitly states that the semantic of the schema instance:

* is defined at contract level;

* is the same for every message;

* and is not conveyed nor specific for each message.

In this case, processing the semantic conveyed in a message might have security implications.

An application that relies on this specification might want to define separate processing streams for JSON documents and RDF graphs, even when RDF graphs are serialized as JSON-LD documents. For example, it might want to raise an error when an application/json resource contains unexpected properties impacting on the application logic like @type and @context.

5. IANA Considerations

None

6. References

6.1. Normative References

[HTTP] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/rfc/rfc9110>.

[JSON] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/rfc/rfc8259>.

[JSON-LD-11] "JSON-LD 1.1", n.d., <https://www.w3.org/TR/json-ld11/>.

[JSONSCHEMA] "JSON Schema", n.d., <https://json-schema.org/specification.html>.

[OAS] Darrel Miller, Jeremy Whitlock, Marsh Gardiner, Mike Ralphson, Ron Ratovsky, and Uri Sarid, "OpenAPI Specification 3.0.0", 26 July 2017.

[RDF] "RDF Concepts and Abstract Syntax", n.d., <https://www.w3.org/TR/rdf11-concepts/>.

[RDFS] "RDF Schema 1.1", n.d., <https://www.w3.org/TR/rdf-schema/>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[URI] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/rfc/rfc3986>.

[YAML] Oren Ben-Kiki, Clark Evans, Ingy dot Net, Tina M端ller, Pantelis Antoniou, Eemeli Aro, and Thomas Smith, "YAML Ain’t Markup Language Version 1.2", 1 October 2021, <https://yaml.org/spec/1.2.2/>.

[YAML-IANA]
"The application/yaml Media Type", n.d.,
<https://www.iana.org/assignments/media-types/application/yaml>.

6.2. Informative References

[I-D.ietf-jsonpath-base]
G旦ssner, S., Normington, G., and C. Bormann, "JSONPath: Query expressions for JSON", Work in Progress, Internet-Draft, draft-ietf-jsonpath-base-21, 24 September 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-jsonpath-base-21>.

[JSON-POINTER]
Bryan, P., Ed., Zyp, K., and M. Nottingham, Ed., "JavaScript Object Notation (JSON) Pointer", RFC 6901, DOI 10.17487/RFC6901, April 2013, <https://www.rfc-editor.org/rfc/rfc6901>.

[JSON-SCHEMA-RDF]
"JSON Schema in RDF", n.d., <https://www.w3.org/2019/wot/json-schema/>.

[JSONLD-11-API]
"JSON-LD 1.1 Processing Algorithms and API", n.d., <https://www.w3.org/TR/json-ld11-api/>.

[OWL] "OWL 2 Web Ontology Language Document Overview", n.d., <https://www.w3.org/TR/owl2-overview/>.

[SHACL] "Shapes Constraint Language (SHACL)", 20 July 2017, <https://www.w3.org/TR/shacl/>.

[XS] "XML Schema", n.d., <https://www.w3.org/2001/XMLSchema>.

Appendix A. Examples

A.1. Schema with semantic information

The following example shows a Person JSON Schema with semantic information provided by the x-jsonld-type and x-jsonld-context. Type information is provided as a URI reference.

```
Person:
  "x-jsonld-type": "https://schema.org/Person"
  "x-jsonld-context":
     "@vocab": "https://schema.org/"
     custom_id: null  # detach this property from the @vocab
     country:
       "@id": addressCountry
       "@language": en
  type: object
  required:
  - given_name
  - family_name
  properties:
    familyName: { type: string, maxLength: 255  }
    givenName:  { type: string, maxLength: 255  }
    country:    { type: string, maxLength: 3, minLength: 3 }
    custom_id:  { type: string, maxLength: 255  }
  example:
    familyName: "Doe"
    givenName: "John"
    country: "FRA"
    custom_id: "12345"
```

Figure 5: A JSON Schema data model with semantic context and type.

The example object is assembled as a JSON-LD object as follows.

```
{
  "@context": {
    "@vocab": "https://schema.org/",
    "custom_id": null
  },
  "@type": "https://schema.org/Person",
  "familyName": "Doe",
  "givenName": "John",
  "country": "FRA",
  "custom_id": "12345"
}
```

The above JSON-LD can be represented as text/turtle as follows.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
@prefix schema: <https://schema.org/>

_:b0 rdf:type schema:Person    ;
     schema:country     "FRA"  ;
     schema:familyName  "Doe"  ;
     schema:givenName   "John" .
```

A.2. Schema with semantic and vocabulary information

The following example shows a "Person" schema with semantic information provided by the x-jsonld-type and x-jsonld-context.

```
Person:
  "x-jsonld-type": "https://schema.org/Person"
  "x-jsonld-context":
     "@vocab": "https://schema.org/"
     email: "@id"
     custom_id: null  # detach this property from the @vocab
     country:
       "@id": addressCountry
       "@type": "@id"
       "@context":
          "@base": "http://publications.europa.eu/resource/authority/country/"

  type: object
  required:
  - email
  - given_name
  - family_name
  properties:
    email: { type: string, maxLength: 255  }
    familyName: { type: string, maxLength: 255  }
    givenName:  { type: string, maxLength: 255  }
    country:    { type: string, maxLength: 3, minLength: 3 }
    custom_id:  { type: string, maxLength: 255  }
  example:
    familyName: "Doe"
    givenName: "John"
    email: "jon@doe.example"
    country: "FRA"
    custom_id: "12345"
```

Figure 6: A JSON Schema data model with semantic context and type.

The resulting RDF graph is

```
@prefix schema: <https://schema.org/> .
@prefix country: <http://publications.europa.eu/resource/authority/country/> .

<mailto:jon@doe.example>
  schema:familyName "Doe"          ;
  schema:givenName "John"          ;
  schema:addressCountry country:FRA .
```

Figure 7: An RDF graph with semantic context and type.

A.3. Cyclic schema

The following schema contains a cyclic reference. Type information is resolved using the @vocab keyword specified in the x-jsonld-context.

```
Person:
  description: Simple cyclic example.
  x-jsonld-type: Person
  x-jsonld-context:
    "email": "@id"
    "@vocab": "https://w3.org/ns/person#"
    children:
      "@container": "@set"
  type: object
  properties:
    email: { type: string }
    children:
      type: array
      items:
        $ref: '#/Person'
  example:
    email: "mailto:a@example"
    children:
    - email: "mailto:dough@example"
    - email: "mailto:son@example"
```

The example schema instance contained in the above schema results in the following JSON-LD document.

```
{
  "email": "mailto:a@example",
  "children": [
    {
      "email": "mailto:dough@example",
      "@type": "Person"
    },
    {
      "email": "mailto:son@example",
      "@type": "Person"
    }
  ],
  "@type": "Person",
  "@context": {
    "email": "@id",
    "@vocab": "https://w3.org/ns/person#",
    "children": {
      "@container": "@set"
    }
  }
}
```

Applying the workflow described in Section 2.3 just recursively copying the x-jsonld-context, the instance context could have been more complex.

```
{
  ...
  "@context": {
    "email": "@id",
    "@vocab": "https://w3.org/ns/person#",
    "children": {
      "@container": "@set",
      "@context": {
        "email": "@id",
        "@vocab": "https://w3.org/ns/person#",
        "children": {
          "@container": "@set"
        }
      }
    }
  }
}
```

Figure 8: An instance context containing redundant information

A.4. Composite instance context

In the following schema document, the "Citizen" schema references the "BirthPlace" schema.

```
   BirthPlace:
     x-jsonld-type: https://w3id.org/italia/onto/CLV/Feature
     x-jsonld-context:
       "@vocab": "https://w3id.org/italia/onto/CLV/"
       country:
         "@id": "hasCountry"
         "@type": "@id"
         "@context":
           "@base": "http://publications.europa.eu/resource/authority/country/"
       province:
         "@id": "hasProvince"
         "@type": "@id"
         "@context":
           "@base": "https://w3id.org/italia/data/identifiers/provinces-identifiers/vehic
le-code/"
     type: object
     required:
       - province
       - country
     properties:
       province:
         description: The province where the person was born.
         type: string
       country:
         description: The iso alpha-3 code of the country where the person was born.
         type: string
     example:
       province: RM
       country: ITA
   Citizen:
     x-jsonld-type: Person
     x-jsonld-context:
       "email": "@id"
       "@vocab": "https://w3.org/ns/person#"
     type: object
     properties:
       email: { type: string }
       birthplace:
         $ref: "#/BirthPlace"
     example:
       email: "mailto:a@example"
       givenName: Roberto
       familyName: Polli
       birthplace:
         province: LT
         country: ITA
```

Figure 9: A schema with object contexts.

The example schema instance contained in the above schema results in the following JSON-LD document. The instance context contains information from both "Citizen" and "BirthPlace" semantic keywords.

```
{
  "email": "mailto:a@example",
  "givenName": "Roberto",
  "familyName": "Polli",
  "birthplace": {
    "province": "RM",
    "country": "ITA",
    "@type": "https://w3id.org/italia/onto/CLV/Feature"
  },
  "@type": "Person",
  "@context": {
    "email": "@id",
    "@vocab": "https://w3.org/ns/person#",
    "birthplace": {
      "@context": {
        "@vocab": "https://w3id.org/italia/onto/CLV/",
        "city": "hasCity",
        "country": {
          "@id": "hasCountry",
          "@type": "@id",
          "@context": {
            "@base": "http://publications.europa.eu/resource/authority/country/"
          }
        },
        "province": {
          "@id": "hasProvince",
          "@type": "@id",
          "@context": {
            "@base": "https://w3id.org/italia/data/identifiers/provinces-identifiers/v
ehicle-code/"
          }
        }
      }
    }
  }
}
```

Figure 10: A @context that includes information from different schemas.

That can be serialized as text/turtle as

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix eu: <https://w3.org/ns/person#> .
@prefix itl: <https://w3id.org/italia/onto/CLV/> .

<mailto:a@example>
  rdf:type eu:Person ;
  eu:birthplace _:b0 ;
  eu:familyName "Polli" ;
  eu:givenName  "Roberto"
.
_:b0 rdf:type itl:Feature ;
  itl:hasCountry <http://publications.europa.eu/resource/authority/country/ITA> .
  itl:hasProvince <https://w3id.org/italia/data/identifiers/provinces-identifiers/vehi
cle-code/RM>
.
```

Figure 11: The above entry in text/turtle

A.5. Identifiers and IRI Expansion

IRI expansion expects string identifiers, so an @id that should be expanded in conjunction with a @base can only be assigned to string properties.

```
Person:
  type: object
  x-jsonld-type: "Person"
  x-jsonld-context:
    "@vocab": "https://w3id.org/italia/onto/CPV/"
    "@base": "https://example.org/people/"
    taxCode: "@id"  # taxCode is a string property.
  required:
  - taxCode
  properties:
    # Since taxCode is an identifier to be expanded
    #   with @base, it must be a string.
    taxCode:
      type: string
  example:
    taxCode: "RSSMRA85M01H501U"
```

Figure 12: A schema that uses IRI expansion with a string property.

Appendix B. Acknowledgements

Thanks to Giorgia Lodi, Matteo Fortini and Saverio Pulizzi for being the initial contributors of this work.

In addition to the people above, this document owes a lot to the extensive discussion inside and outside the workgroup. The following contributors have helped improve this specification by opening pull requests, reporting bugs, asking smart questions, drafting or reviewing text, and evaluating open issues:

Pierre-Antoine Champin, and Vladimir Alexiev.

FAQ

This section is to be removed before publishing as an RFC.

Q: Why this document? There’s currently no standard way to provide machine-readable semantic information in [OAS] / [JSONSCHEMA] to be used at contract time.

Q: Does this document support the exchange of JSON-LD resources? Thi s document is focused on annotating schemas that are used at contract/design time, so that application can exchange compact JSON object without dereferencing nor interpreting external resources at runtime.

While you can use the provided semantic information to generate JSON-LD objects, it is not the primary goal of this specification: context information are not expected to be dereferenced at runtime (see security considerations in JSON-LD) and the semantics of exchanged messages is expected to be constrained inside the application.

Q: Why don’t use existing [JSONSCHEMA] keywords like externalDocs ? We already tried, but this was actually squatting a keyword designed for human readable documents (https://github.com/OAI/OpenAPI-Specification/blob/main/versions/3.1.0.md#externalDocumentationObject).

Q: Why using x- keywords? OpenAPI 3.0 considers invalid unregistered keywords that don’t start with x-, and we want a solution that is valid for all OAS versions >= 3.0.

Q: Why not using a full-semantic approach? This approach allows API providers to attach metadata to their specification without modifying their actual services nor their implementation, since custom keywords are ignored by OpenAPI toolings like Gateways and code generators.

Q: Why not defining a mechanism to attach semantic information to

non-object schemas (e.g. JSON Strings) like other implementations? T

his is actually problematic. Look at this example that reuses the TaxCode schema and semantic in different properties.

Q: Why don’t use SHACL or OWL restrictions instead of JSON Schema? Web and mobile developers consider JSON Schema is easier to use than SHACL. Moreover, OWL restrictions are about semantics, and are not designed to restrict the syntax.

Q: Why don’t design for composability first? JSON-LD is a complex specification. Consider the following schemas, where Contract references TaxCode.

```
TaxCode:
  type: string
  $linkedData:
    "@id": "https://w3id.org/italia/onto/CPV/taxCode"
    "term": "taxCode"
Contract:
  ...
  properties:
    employer_tax_code:
      # Beware! TaxCode.$linkedData.term == 'taxCode'
      $ref: "#/components/schemas/TaxCode"
    employee_tax_code:
      # Here we are reusing not only the schema,
      #   but even the same term.
      $ref: "#/components/schemas/TaxCode"
```

The result will be that only one of the properties will be correctly annotated. For this reason, composability is limited to the object level.

Q: Can the value of x-jsonld-type be an rdf:Property? Would this allow to reuse the same schema in different objects without modifying the @context? Under normal circumstances, i.e. when designing public or financial service APIs, you don’t want x-jsonld-type to be an rdf:Property. The value of x-jsonld-type usually maps to a owl:Class, not an owl:DataTypeProperty; for example a sensible value for x-jsonld-type would be rdfs:Literal (that is, the rdfs:range of CPV:taxCode), but this would be mostly a syntactic information, which instead is provided by JSON Schema.

```
       TaxCode:
         type: string
         x-jsonld-type: "https://w3id.org/italia/onto/CPV/taxCode"
         description: |-
           This example is ambiguous, because:

           1. it treats a CPV:taxCode as an owl:Class,
              while it's an owl:DataTypeProperty;
           2. the 'rdfs:range' for CPV:taxCode is 'rdfs:Literal'.
```

Figure 13: The above code is ambiguous, because the rdfs:range of CPV:taxCode is rdfs:Literal

Change Log

This section is to be removed before publishing as an RFC.

TBD

Author's Address

Roberto Polli
Digital Transformation Department, Italian Government
Italy
Email: robipolli@gmail.com