Hydra Core Vocabulary

A Vocabulary for Hypermedia-Driven Web APIs

Unofficial Draft 02 November 2020

Latest editor's draft:
http://www.hydra-cg.com/spec/latest/core/

Editor:
Markus Lanthaler (Google)

Author:
Markus Lanthaler (Google)

This document is also available in these non-normative formats: JSON-LD and Turtle

Abstract

Hydra is a lightweight vocabulary to create hypermedia-driven Web APIs. By specifying a number of concepts commonly used in Web APIs it enables the creation of generic API clients.

Status of This Document

This document is draft of a potential specification. It has no official standing of any kind and does not represent the support or consensus of any standards organization.

ISSUE 1

This entire document is a work in progress and several sections are incomplete, missing, or outdated. All open issues and decisions are documented in our issue tracker. If you have questions, please don't hesitate to join the Hydra W3C Community Group and post to the mailing list.

This specification was published by the Hydra W3C Community Group. It is not a W3C Standard nor is it on the W3C Standards Track. Please note that under the W3C Community Contributor License Agreement (CLA) there is a limited opt-out and other conditions apply. Learn more about W3C Community and Business Groups.

To participate in the development of this specification, please join the Hydra W3C

Community Group. If you have questions, want to suggest a feature, or raise an issue, please send a mail to the public-hydra@w3.org mailing list.

Table of Contents

1. Introduction

2. Conformance

3. Hydra at a Glance

4. Using Hydra
4.1 Adding Affordances to Representations
4.2 Documenting a Web API
4.3 Discovering a Hydra-powered Web API
4.4 Api versions

5. Advanced Concepts
5.1 Collections
5.2 Member assertions
5.3 Templated Links
5.4 Description of HTTP Status Codes and Errors
5.5 Client initiated pagination

6. Classes

7. Properties

8. Acknowledgements

A. The Hydra Core Vocabulary in JSON-LD

B. References
B.1 Normative references
B.2 Informative references

§ 1. Introduction

This section is non-normative.

Coping with the ever-increasing amount of data becomes increasingly challenging. To alleviate the information overload put on people, systems are progressively being connected directly to each other. They exchange, analyze, and manipulate humongous amounts of data without any human interaction. Most current solutions, however, do not exploit the whole potential of the architecture of the World Wide Web and completely ignore the possibilities offered by Linked Data technologies.

The combination of the REST architectural style and the Linked Data principles offer opportunities to advance the Web of machines in a similar way that hypertext did for the human Web. Most building blocks exist already and are in place but they are rarely used together. Hydra tries to fill that gap. It allows data to be enriched with machine-readable affordances which enable interaction. This not only addresses the problem that Linked Data is still mostly read-only, but it also paves the way for a completely new breed of interoperable Web APIs. The fact that it enables the creation of composable contracts means that interaction models of Web APIs can be reused at an unprecedented granularity.

§ 2. Conformance

This specification describes the conformance criteria for ***Hydra API documentations*** and ***Hydra clients***. These criteria are relevant to authors, authoring tool implementers, and client implementers. All authoring guidelines, diagrams, examples, and notes in this specification are non-normative, as are all sections explicitly marked as non-normative. Everything else in this specification is normative.

ISSUE 2

Conformance for Hydra clients should probably not be specified in this document.

ISSUE 3

Add normative statements

The key words *MUST*, *MUST NOT*, *REQUIRED*, *SHALL*, *SHALL NOT*, *SHOULD*, *SHOULD NOT*, *RECOMMENDED*, *NOT RECOMMENDED*, *MAY*, and *OPTIONAL* in this specification have the meaning defined in [RFC2119].

§ 3. Hydra at a Glance

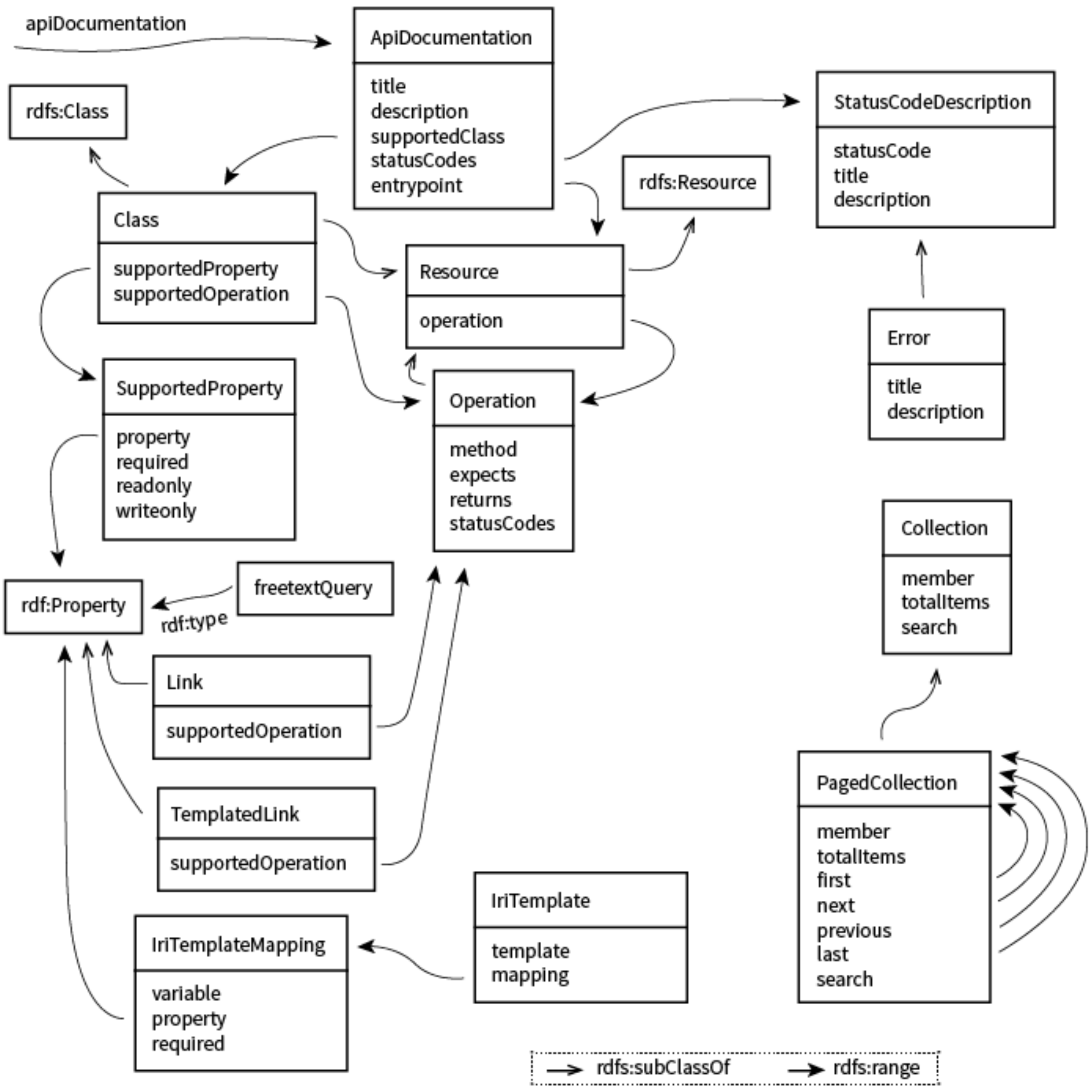
This section is non-normative.

The basic idea behind Hydra is to provide a vocabulary which enables a server to advertise valid state transitions to a client. A client can then use this information to construct HTTP requests which modify the server’s state so that a certain desired goal is achieved. Since all the information about the valid state transitions is exchanged in a machine-processable way at runtime instead of being hardcoded into the client at design time, clients can be decoupled from the server and adapt to changes more easily.

The namespace of the Hydra core vocabulary is `http://www.w3.org/ns/hydra/core#`, and the suggested prefix is `hydra`. The figure below illustrates the vocabulary (the figure’s goal is to show how Hydra is used rather than its precise definition).

ISSUE 4

Is this illustration clear enough or is it confusing? Feedback would be much appreciated.



The Hydra core vocabulary

ISSUE 5

Add ranges for Operation "members"

An alphabetical index of the classes and properties of Hydra is given below. All the terms are hyperlinked to their detailed description for quick reference.

hydra:ApiDocumentation hydra:BaseUriSource hydra:Class hydra:Collection hydra:Error hydra:IriTemplate hydra:IriTemplateMapping hydra:Link hydra:Operation hydra:PartialCollectionView hydra:Resource hydra:Status hydra:SupportedProperty hydra:TemplatedLink hydra:VariableRepresentation

hydra:apiDocumentation hydra:collection hydra:description hydra:entrypoint hydra:expects hydra:expectsHeader hydra:first hydra:freetextQuery hydra:last hydra:limit hydra:mapping hydra:member hydra:method hydra:next hydra:offset hydra:operation hydra:pageIndex hydra:pageReference hydra:possibleStatus hydra:previous hydra:property hydra:readable hydra:required hydra:resolveRelativeUsing hydra:returns hydra:returnsHeader hydra:search hydra:statusCode hydra:supportedClass hydra:supportedOperation hydra:supportedProperty hydra:template hydra:title hydra:totalItems hydra:variable hydra:variableRepresentation hydra:view hydra:writable

ISSUE 6

The used prefixes should be documented somewhere.

§ 4. Using Hydra

Throughout this section, a simple Web API featuring an issue tracker will be used to illustrate how Hydra can be used. The Web API enables its users to file new issues, modify or delete existing ones, and to comment them. For the sake of simplicity, orthogonal aspects such as authentication or authorization are not covered.

§ 4.1 Adding Affordances to Representations

The exemplary Web API has to expose representations of issues and comments. To enable interaction with those resources, a client has to know which operations the server supports. In human-facing websites such affordances are typically exposed by links and forms and described in natural language. Unfortunately, machines can not interpret such information easily. The solution that presents itself is to reduce the language to a small number of unambiguous concepts which are easily recognizable by a machine client. Hydra formalizes such concepts.

The simplest and most important affordance on the Web are hyperlinks. Without them, it would be impossible to browse the Web. Users typically select the link based on the text it is labeled with. To give machines a similar understanding, links can be annotated with a link relation type—a registered token or a URI identifying the semantics of the link. The following example shows how such a typed link is used in HTML to reference a stylesheet.

EXAMPLE 1: A typed link referencing a stylesheet as used in HTML

```
<link rel="stylesheet" href="http://www.example.com/styles.css" />
```

In Linked Data, the link relation type corresponds to the property itself. An example in JSON-LD would thus look as follows.

EXAMPLE 2: Referencing a stylesheet in JSON-LD

```
{
  "urn:iana:link-relations:stylesheet": { "@id": "http://www.example.com/styles.css" }
}
```

Generally, a client decides whether to follow a link or not based on the link relation (or property in the case of Linked Data) which defines its semantics. There are however also clients such as Web crawlers which simply follow every link intended to be dereferenced. In HTML this usually means that all links in anchor elements (the `<a>` tag) are followed but most references in link elements (the `<link>` tag), such as used in the example above, are ignored. Since in RDF serializations no such distinction exists, the best a client can do is to blindly try to dereference all URIs. It would thus be beneficial to describe in a machine-readable manner if a property represents a link intended to be dereferenced or solely an identifier. Hydra's *Link* class does just that. It can be used to define properties that represent dereferenceable links. In the exemplary Web API used throughout this section, it can be used to define a property linking issues to their comments:

EXAMPLE 3: Defining properties representing hyperlinks using Hydra's Link class

```
{
  "@context": "http://www.w3.org/ns/hydra/context.jsonld",
  "@id": "http://api.example.com/vocab#comments",
  "@type": "Link"
}
```

In the example above, a property identified with the URL `http://api.example.com/vocab#comments` is defined to be of the type *Link*. This is enough information for a client understanding Hydra to know that the value of the `comments` property in the following example is intended to be dereferenced.

NOTE

It is recommended to dereference resources that are within an API's domain. This may prevent possible issues with cross-site scripting or obtaining resources which might have no meaning to the client or such that the client would be unable to interpret. Still, there is no formal prohibition of dereferencing resources linked with well-known properties, e.g. *rdf:seeAlso*.

EXAMPLE 4: Using a property defined to be a hyperlink

```
{
  "@context": {
    "comments": "http://api.example.com/vocab#comments"
  },
  "@id": "http://api.example.com/an-issue",
  "title": "An exemplary issue linking to its comments",
  "comments": { "@id": "http://api.example.com/an-issue/comments" }
}
```

In the example above, the value of the `comments` property is a JSON object with an `@id` member. This is JSON-LD's convention to distinguish between strings and IRIs. By using JSON-LD's type-coercion feature, the representation can be made even more idiomatic:

EXAMPLE 5: Using JSON-LD's type-coercion feature to create idiomatic representations

```
{
  "@context": {
    "comments": { "@id": "http://api.example.com/vocab#comments", "@type": "@id" }
  },
  "@id": "http://api.example.com/an-issue",
  "title": "An exemplary issue linking to its comments",
  "comments": "http://api.example.com/an-issue/comments"
}
```

While links are enough to build read-only Web APIs, more powerful affordances are required to build read-write Web APIs. Thus, Hydra introduces the notion of operations. Simply speaking, an *Operation* represents the information necessary for a client to construct valid HTTP requests in order to manipulate the server's resource state. As such, the only required property of an *Operation* is its HTTP *method*. Optionally, it is also possible to describe what information the server *expects* or *returns*, including additional information about HTTP status codes that might be returned. This helps a developer to understand what to expect when invoking an operation. This information has, however, not to be considered as being complete; it is merely a hint. Developers should, e.g., expect that other HTTP status codes might be returned and program their clients accordingly.

The following example illustrates how representations can be augmented with information that enables clients to interact with them.

EXAMPLE 6: A representation of an issue augmented with a delete operation

```
{
  "@context": "http://www.w3.org/ns/hydra/context.jsonld",
  "@id": "/an-issue",
  "title": "An exemplary issue representation",
  "description": "This issue can be deleted with an HTTP DELETE request",
  "operation": [
    {
      "@type": "Operation",
      "method": "DELETE"
    }
  ]
}
```

The example above references Hydra's context to map properties such as `operation` and `method` and values like `Operation` to URLs that unambiguously identify these concepts. It would be similarly valid JSON-LD if these mappings would be directly embedded into the representation or if the full URLs would be used instead. Typically, however, the context is the same for a lot of representations in a Web API and it thus makes sense to reduce the response size by leveraging a remote context that can easily be cached by a client.

§ 4.2 Documenting a Web API

In Web APIs, most representations are typically very similar. Furthermore, resources often support the same operations. It thus makes sense, to collect this information in a central documentation. Traditionally, this has been done in natural language which forces developers to hardcode that knowledge into their clients. Hydra addresses this issue by making the documentation completely machine-processable. The fact that all definitions can be identified by URLs enables reuse at unprecedented granularity.

Hydra's *ApiDocumentation* class builds the foundation for the description of a Web API. As shown in the following example, Hydra describes a API by giving it a title, a short description, and documenting its main entry point. Furthermore, the classes known to be supported by the Web API and additional information about status codes that might be returned can be documented. This information may be used to automatically generate documentations in natural language.

EXAMPLE 7: The overall structure of a Hydra API documentation

```
{
  "@context": "http://www.w3.org/ns/hydra/context.jsonld",
  "@id": "http://api.example.com/doc/",
  "@type": "ApiDocumentation",
  "title": "The name of the API",
  "description": "A short description of the API",
  "entrypoint": "URL of the API's main entry point",
  "supportedClass": [
    ... Classes known to be supported by the Web API ...
  ],
  "possibleStatus": [
    ... Statuses that should be expected and handled properly ...
  ]
}
```

In Linked Data, properties are, just as everything else, identified by IRIs and thus have global scope which implies that they have independent semantics. In contrast, properties in data models as used in common programming languages are class-dependent. Their semantics depend on the class they belong to. In data models classes are typically described by the properties they expose whereas in Linked Data properties define to which classes they belong. If no class is specified, it is assumed that a property may apply to every class.

These differences have interesting consequences. For example, the commonly asked question of which properties can be applied to an instance of a specific class can typically not be answered for Linked Data. Strictly speaking, any property which is not explicitly forbidden could be applied. This stems from the fact that Linked Data works under an open-world assumption whereas data models used by programmers typically work under a closed-world assumption. The difference is that when a closed world is assumed, everything that is not known to be true is false or vice-versa. With an open-world assumption the failure to derive a fact does not automatically imply the opposite; it embraces the fact that the knowledge is incomplete.

ISSUE 7

Mention that Hydra classes are dereferenceable resources.

Since Hydra uses classes to describe the information expected or returned by an operation, it also defines a concept to describe the properties known to be supported by a class. The following example illustrates this feature. Instead of referencing properties directly, *supportedProperty* references an intermediate data structure, namely instances of the *SupportedProperty* class. This makes it possible to define whether a specific property is required or whether it is read-only or write-only depending on the class it is associated with.

EXAMPLE 8: Defining a class and documenting its supported properties

```
{
  "@context": "http://www.w3.org/ns/hydra/context.jsonld",
  "@id": "http://api.example.com/doc/#Comment",
  "@type": "Class",
  "title": "The name of the class",
  "description": "A short description of the class.",
  "supportedProperty": [
    ... Properties known to be supported by the class ...
    {
      "@type": "SupportedProperty",
      "property": "#property", // The property
      "required": true, // Is the property required in a request to be valid?
      "readable": false, // Can the client retrieve the property's value?
      "writable": true // Can the client change the property's value?
    }
  ]
}
```

All instances of a specific class typically support the same operations. Hydra therefore features a *supportedOperation* property which defines the operations supported by all instances of a class.

EXAMPLE 9: Defining a class and documenting its supported operations

```
{
  "@context": "http://www.w3.org/ns/hydra/context.jsonld",
  "@id": "http://api.example.com/doc/#Comment",
  "@type": "Class",
  "title": "The name of the class",
  "description": "A short description of the class.",
  "supportedProperty": [
    ... Properties known to be supported by the class ...
  ],
  "supportedOperation": [
    ... Operations known to be supported by instances of the class ...
  ]
}
```

The same feature can be used to describe the operations supported by values of a *Link* property. This is often helpful when certain operations depend on the permissions of the current user. It makes it, e.g., possible to show a "delete" link only if the current user has the permission to delete the resource. Otherwise, the link would simply be hidden in the representation.

Example shown below describes the operation's expected and returned value as a dereferencable resource (an RDF resource of a given class), but the vocabulary is not limited to only those originating from RDF and is enabled to other types of resources.

EXAMPLE 10: Documenting the supported operations of link properties

```
{
  "@context": "http://www.w3.org/ns/hydra/context.jsonld",
  "@id": "http://api.example.com/doc/#comments",
  "@type": "Link",
  "title": "Comments",
  "description": "A link to comments with an operation to create a new comment.",
  "supportedOperation": [
    {
      "@type": "Operation",
      "title": "Creates a new comment",
      "method": "POST",
      "expects": "http://api.example.com/doc/#Comment",
      "returns": "http://api.example.com/doc/#Comment",
      "possibleStatus": [
        ... Statuses that should be expected and handled properly ...
      ]
    }
  ]
}
```

In addition to expected/returned resources, it is also possible to express similar features for headers with *returnsHeader* and *expectsHeader* predicates which provides a simple set of header names. Client *SHOULD* apply respective header semantics when creating or receiving a request natural for the protocol in use.

EXAMPLE 11: Making a use of a returned header

```
{
  "@context": "http://www.w3.org/ns/hydra/context.jsonld",
  "@id": "http://api.example.com/doc/#comments",
  "@type": "Link",
  "supportedOperation": [
    {
      "@type": "Operation",
      "method": "POST",
      "expects": "http://api.example.com/doc/#Comment",
      "returns": "http://api.example.com/doc/#Comment",
      "returnsHeader": [
        "Content-Type",
        "Content-Length"],
      "expectsHeader": [
        "Authorization"
      ]
    }
  ]
}
```

The example above enable an HTTP client to prepare a proper cross-site pre-flight request so the server exposes enlisted headers for the client. The client is also aware of the user authentication requirement necessary for the operation invocation.

To wrap up everything altogether, it is also possible to attach atomic operations supported by, well, supported property itself. This might come in handy for scenarios, when resource can be partially modified. It can be achieved with two approaches, both having advantages and disadvantages.

First approach would involve adding a *supportedOperation* to the intermediate structure of *SupportedProperty*. This way prevents from leaking API specific features from the API itself to i.e. externally defined properties. Data aggregators won't assume that each instance with a given property could have such an operation.

Another approach would require the API to elevate a specific property to *Link*, which can accept a *supportedOperation*. This is more intuitive in APIs operating with internally used vocabularies where assumption that every instance with that very specific property has the operation attached available.

Direct usage of *supportedOperation* on *rdf:Property* without elevating it to the *Link* *SHOULD NOT* be implemented as clients may not discover such a construct correctly.

These are the simple example scenarios and possible usages are not limited to those described above.

Keep in mind that operations specified in an *ApiDocumentation* may fail at runtime as either resources or the *ApiDocumentation* itself have changed since they have been retrieved. Also operation details like *returns* or *possibleStatus* may vary at runtime, which means client *SHOULD* verify received payloads at runtime. A simple strategy to try to recover from such a situation is to reload the *ApiDocumentation* and redo all pre-computations that were based on the *ApiDocumentation* (or at least those that lead to the current failure). Another, simpler approach would require an application to show an error message with option to return to a previous or home screen.

ISSUE 8

Describe the various properties of an operation.

Hydra also allows enriching both *ApiDocumentation* and hypermedia controls with human-readable descriptions by applying *title* and *description* (as shown in the examples above). The former states a name of such a decorated element that could be displayed as a label. The latter provides its description to be presented i.e. as a hint.

Aforementioned *title* and *description* *SHOULD* take precedence over standard *rdfs:label* and *rdfs:comment*.

There is one more feature related to how Linked Data works. Consider the example below written in turtle syntax:

EXAMPLE 12: RDF as a graph

```
# An example API documentation itself with all the standard bits
@base <http://some.app/> .
@prefix api: <http://some.api/> .
@prefix ex: <http://ontology.example/> .
@prefix hydra: <http://www.w3.org/ns/hydra/core#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

<api>
  a hydra:ApiDocumentation ;
  hydra:supportedClass api:ClassOne, api:ClassTwo .

# Anything else
ex:SomeType a rdfs:Class .
```

and how it could be transformed with JSON-LD framing process:

EXAMPLE 13: RDF as a graph continued

```
{
  "@context": {  },
  "@graph": [
    {
      "@id": "http://some.app/api",
      "@type": "hydra:ApiDocumentation",
      "hydra:supportedClass": [
        {
          "@id": "api:ClassTwo"
        },
        {
          "@id": "api:ClassOne"
        }
      ]
    }
  ]
}
```

As you can see, additional details about *ex:SomeType* went missing, while this shouldn't happen. The fact that the IRI mentioned is an *rdfs:Class* may be meaningful for a correct interpretation of the received payload and this is a sole reason of why a Client *SHOULD NOT* disregard other parts of the payload that are not directly related to the API documentation or other hypermedia controls.

§ 4.3 Discovering a Hydra-powered Web API

The first step when trying to access a Web API is to find an entry point. Typically, this is done by looking for documentation on the API publisher's homepage. Hydra enables the API's main entry point to be discovered automatically if the API publisher marks his responses with a special HTTP Link Header as defined in [RFC5988]. A Hydra client would look for a Link Header with a relation type `http://www.w3.org/ns/hydra/core#apiDocumentation` (this is the IRI identifying the *hydra:apiDocumentation* property).

In the following example, a Hydra client simply accesses the homepage of an API publisher (`http://www.example.com`) to find the entry point of its API. A client may use an HTTP GET or HEAD request. The difference between the two is that the former may return a message-body in the response whereas the latter will not; otherwise they are identical.

EXAMPLE 14: Discovering Hydra API documentation documents

```
HEAD / HTTP/1.1
Host: www.example.com

====================================

HTTP/1.1 200 OK
...
Content-Type: text/html; charset=utf-8
Link: <http://api.example.com/doc/>; rel="http://www.w3.org/ns/hydra/core#apiDocumentat
```

The response in the example above contains an HTTP Link Header pointing to `http://api.example.com/doc/`. Retrieving that resource, the client would obtain a Hydra API documentation defining the API's main entry point:

EXAMPLE 15: Retrieving a Hydra API documentation to obtain the main entry point

```
GET /doc/ HTTP/1.1
Host: api.example.com
Accept: application/ld+json

====================================

HTTP/1.1 200 OK
...
Content-Type: application/ld+json

{
  "@context": "http://www.w3.org/ns/hydra/context.jsonld",
  "@id": "http://api.example.com/doc/",
  "title": "The example.com API",
  "entrypoint": "http://api.example.com/",
  ...
}
```

Please note that in most cases the entry point will already be known to the client. Thus, the discovery of the API documentation using HTTP Link Headers is typically not necessary as the concepts used in responses from the API will dereference to their documentation.

In another scenario the *ApiDocumentation* would be discovered from a bookmarked resource's representation. Api implementation *SHOULD* emit the HTTP *Link* header on every Api response, making the *ApiDocumentation* (and entry points it defines) discoverable all the time.

EXAMPLE 16: Retrieving a Hydra API documentation from another Api response

```
GET /api/items HTTP/1.1
Host: api.example.com
Accept: application/ld+json

====================================

HTTP/1.1 200 OK
...
Content-Type: application/ld+json
Link: <http://api.example.com/doc/>; rel="http://www.w3.org/ns/hydra/core#apiDocumentat

{
  "@context": "http://www.w3.org/ns/hydra/context.jsonld",
  "@id": "http://api.example.com/api/items",
  "title": "Items collection",
  ...
}
```

§ 4.4 Api versions

It is common to provide a separate API address after a breaking changes update. This prevents current clients not to get broken as these may not support these changes.

With hypermedia provided in each response payload, it may be unnecessary to provide such an alternative API. This is due to fact the client follows what the server provides and with proper margin for errors implemented within that client, even breaking changes can be published on the fly.

Still, Hydra does neither have any special support for API versions, nor prevents them. It's fully an implementers decision on if and how to provide the API features.

§ 5. Advanced Concepts

ISSUE 9

Describe Hydra's Resource class? Or should that better be described somewhere in the beginning?

§ 5.1 Collections

In many situations, it makes sense to expose resources that reference a set of somehow related resources. Results of a search query or entries of an address book are just two examples. To simplify such use cases, Hydra defines the two classes *hydra:Collection* and *hydra:PartialCollectionView*.

A *hydra:Collection* can be used to reference a set of resources as follows:

EXAMPLE 17: Referencing related resources using a Hydra Collection

```
{
  "@context": "http://www.w3.org/ns/hydra/context.jsonld",
  "@id": "http://api.example.com/an-issue/comments",
  "@type": "Collection",
  "totalItems": "4980",
  "member": [
    {
      "@id": "/comments/429"
    },
    {
      "@id": "/comments/781",
      "title": "Properties may be embedded directly in the collection"
    },
    ...
  ]
}
```

As shown in the example above, member items can either consist of solely a link or also include some properties. In some cases embedding member properties directly in the collection is beneficial as it may reduce the number of HTTP requests necessary to get enough information to process the result.

Since collections may become very large, Web APIs often chose to split a collection into multiple pages. In Hydra, that can be achieved with a *hydra:PartialCollectionView*. It describes a specific view on the collection which represents only a subset of the collection's members. A *PartialCollectionView* may contain links to the *first*, *next*, *previous*, and *last PartialCollectionView* which allows a client to find all members of a *Collection*.

EXAMPLE 18: A Hydra PartialCollectionView splits a collection into multiple views

```
{
  "@context": "http://www.w3.org/ns/hydra/context.jsonld",
  "@id": "http://api.example.com/an-issue/comments",
  "@type": "Collection",
  "totalItems": "4980",
  "member": [
    ... a subset of the members of the Collection ...
  ],
  "view": {
    "@id": "http://api.example.com/an-issue/comments?page=3",
    "@type": "PartialCollectionView",
    "first": "/an-issue/comments?page=1",
    "previous": "/an-issue/comments?page=2",
    "next": "/an-issue/comments?page=4",
    "last": "/an-issue/comments?page=498"
  }
}
```

ISSUE 10

Say that all these properties are optional? What about *first* and, perhaps more interestingly, *last*?

§ 5.2 Member assertions

A *memberAssertion* is a way to declare additional, implicit statements about members of a collection. Statements which may otherwise be missing from the respective member resources inlined in a collection's representation.

EXAMPLE 19: Manages block describes relation with another resource

```
{
  "@context": "http://www.w3.org/ns/hydra/context.jsonld",
  "@id": "http://api.example.com/an-issue/comments",
  "@type": "Collection",
  "memberAssertion": {
    "subject": "http://api.example.com/an-issue",
    "property": "http://api.example.com/vocab#comment"
  },
  "member": [
    {
      "@id": "/comments/429"
    }
  ]
}
```

In the above example, adding a `memberAssertion` node to the collection instructs the client that every member of this collection is linked to the `subject` by the `property`. It could be written as a SPARQL triple pattern below, where `?m` would be substituted by each member of the collection.

EXAMPLE 20

```
<http://api.example.com/an-issue> <http://api.example.com/vocab#comment> ?m
```

A *memberAssertion MUST* use two and only two of the `subject`, `property` and `object` predicates. There `memberAssertion` predicate *MAY* have more than one such blocks, each expressing different relations between the collection members and other resources.

NOTE

It's important to point out that the `subject`, `property` and `object` predicates are defined within the Hydra namespace and are not `rdf` terms.

§ 5.3 Templated Links

Sometimes, it is impossible for a server to construct a URL because the URL depends on information only known by the client. A typical use case are URLs which enable a client to query the server. In such a case, the server cannot construct the URL because it does not know the query the client is interested in. What the server can do however, is to give the client a template to construct such a URL at runtime. In Hydra, the *IriTemplate* class is used to do so.

An *IriTemplate* consists of a *template* literal and a set of *mappings*. Each *IriTemplateMapping* maps a *variable* used in the template to a *property* and may optionally specify whether that variable is *required* or not. The syntax of the template literal is specified by its datatype and defaults to the [RFC6570] URI Template syntax, which can be explicitly indicated by *hydra:Rfc6570Template*.

EXAMPLE 21: Description of an IRI Template

```
{
  "@context": "http://www.w3.org/ns/hydra/context.jsonld",
  "@type": "IriTemplate",
  "template": "http://api.example.com/issues{?q}",
  "variableRepresentation": "BasicRepresentation",
  "mapping": [
    {
      "@type": "IriTemplateMapping",
      "variable": "q",
      "property": "hydra:freetextQuery",
      "required": true
    }
  ]
}
```

The example above maps the variable `q` to Hydra's *freetextQuery* property and marks it as required. As its name suggests, the *freetextQuery* property can be used for free text queries.

A template syntax only details how to fill out simple string values, but not how to derive such string values from typed values, language-tagged strings, or IRIs. Hydra addresses this by specifying how such values are to be serialized as strings. The serialization of an *IriTemplate*'s variables can be described by setting the *variableRepresentation* property to *BasicRepresentation* or *ExplicitRepresentation*. The *BasicRepresentation* represents values by their lexical form. It omits type and language information and does not differentiate between IRIs and literals. The *ExplicitRepresentation*, on the other hand, includes type and language information and differentiates between IRIs and literals by serializing values as follows:

- IRIs are represented as-is.
- Literals, i.e., (typed) values and language-tagged strings are represented by their lexical form, surrounded by a single pair of doubles quotes (`"`).
- If a literal has a language, a single `@` symbol is appended after the double-quoted lexical form, followed by a non-empty [BCP47] language code.
- If a literal has a type, two *caret* symbols (`^^`) are appended after the double-quoted literal, followed by the full datatype IRI.

In both representations characters *MUST NOT* be escaped. In case the representation format is not explicitly described, clients *SHOULD* use the *BasicRepresentation* by default.

⚠ Warning

Although *ExplicitRepresentation* use of `@` and `^^` is similar, it is *not* the same as the [Turtle] representation for literals. Turtle literals require escaping of special characters, surround datatype IRIs with angular brackets (`<` and `>`), and also allow single quotes (`'`) to indicate literals. *ExplicitRepresentation* values must not be escaped, IRIs must not be surrounded by any character, and only double quotes can indicate literals.

Below are some example values serialized in the different representations as well as the result of expanding the IRI template `http://example.com/find/{value}` with the respective value.

EXAMPLE 22: The different variable representations

The IRI `http://www.hydra-cg.com/`

BasicRepresentation: `http://www.hydra-cg.com/`; resulting IRI: `http://example.com/find/http%3A%2F%2Fwww.hydra-cg.com%2F`

ExplicitRepresentation: `http://www.hydra-cg.com/`; resulting IRI: `http://example.com/find/http%3A%2F%2Fwww.hydra-cg.com%2F`

The string `A simple string`

BasicRepresentation: `A simple string`; resulting IRI: `http://example.com/find/A%20simple%20string`

ExplicitRepresentation: `"A simple string"`; resulting IRI: `http://example.com/find/%22A%20simple%20string%22`

The string `A string " with a quote`

BasicRepresentation: `A string " with a quote`; resulting IRI: `http://example.com/find/A%20string%20%22%20with%20a%20quote`

ExplicitRepresentation: `"A string " with a quote"`; resulting IRI: `http://example.com/find/%22A%20string%20%22%20with%20a%20quote%22`

The language-tagged string `A simple string` with language English

BasicRepresentation: `A simple string`; resulting IRI: `http://example.com/find/A%20simple%20string`

ExplicitRepresentation: `"A simple string"@en`; resulting IRI: `http://example.com/find/%22A%20simple%20string%22%40en`

The decimal value `5.5`

BasicRepresentation: `5.5`; resulting IRI: `http://example.com/find/5.5`

ExplicitRepresentation: `"5.5"^^http://www.w3.org/2001/XMLSchema#decimal`; resulting IRI: `http://example.com/find/%225.5%22%5E%5Ehttp%3A%2F%2Fwww.w3.org%2F2001%2FXMLSchema%23decimal`

Similar to how Hydra's *Link* class allows the definition of properties that represent hyperlinks as described in § 4.1 Adding Affordances to Representations, the *TemplatedLink* class allows the definition of properties whose value are IRI templates. Hydra predefines one such property, namely the *search* property which can be used to document available search interfaces.

EXAMPLE 23: The definition of Hydra's search property (extract)

```
{
  "@context": "http://www.w3.org/ns/hydra/context.jsonld",
  "@id": "hydra:search",
  "@type": "hydra:TemplatedLink"
}
```

Due to fact that hydra is on top of an RDF, which is a graph, it may happen that a related resource (an object of the relation) may not be fully described in the resource's payload. In case of an *IriTemplate* expected as a related resource, client may discover no additional statements describing it. These rules should be considered when working with *IriTemplate*s (and other hypermedia resources in general):

- in case of an object expected to be a hypermedia resource does not have all the necessary statements for which it is a subject, client *SHOULD* look in the API documentation for more details
- in case the mentioned object, after consulting an API documentation, still does not have all the necessary statements for which it is a subject and both mentioned object's Url and Url of the initially obtained resource has the same scheme and authority (by means of RFC 3986 sections 3.1 and 3.2), client *SHOULD* dereference that Url. If the resource does not have the same scheme and authority the client *MAY* choose to derefernce it (for example if the resource originates from another API well-known to the client)
- in case the mentioned object, still does not have all the necessary statements for which it is a subject (i.e. de-referencing it failed or statements are missing), client *SHOULD* either ignore whole statement (i.e. for display purposes) or throw an exception (i.e. an Iri template is about to be resolved and de-referenced)

Example of each of the situations are as follows:

EXAMPLE 24: IriTemplate details extraction

```
HTTP/1.1 200 OK
Content-Type: application/ld+json
Link: <http://api.example.com/doc/>; rel="http://www.w3.org/ns/hydra/core#apiDocumentat

{
  "@context": "http://www.w3.org/ns/hydra/context.jsonld",
  "@graph": [{
    "@id": "http://api.example.com/people",
    "@type": "hydra:Collection",
    "api:personByName": "api:PersonByNameTemplate"
  }, {
    "@id": "http://api.example.com/events",
    "@type": "hydra:Collection",
    "api:eventByName": "api:EventByNameTemplate"
  }
}

HTTP/1.1 200 OK
Content-Type: application/ld+json

{
  "@context": "http://www.w3.org/ns/hydra/context.jsonld",
  "@graph": [{
    "@id": "api:PersonByNameTemplate",
    "@type": "hydra:IriTemplate",
    "template": "http://api.example.com/people/{name}",
    ...
  }]
}
```

where

- resource *http://api.example.com/people* should have an *IriTemplate* available as there is a complete definition of the template available at *http://api.example.com /doc/*.
- resource *http://api.example.com/events* should not have an Iri template exposed as there are no additional details available, neither in the initial resources' payload nor in the API documentation.

IRI expansion should be performed with respect to the specification behind the IRI template type (RFC 6570 by default), and the product of this process *SHOULD* be an IRI. When the produced IRI is relative, a case client *SHOULD* stick to RFC 3986 sections 5.1.3 and 5.1.4 to be compatible with most RDF serializations that support relative IRIs. Still, it may be preferred to use another base URI for the expansion process, which makes a *resolveRelativeTo* term useful. It allows to switch the IRI template expansion algorithm so the base URI is established using current link context, which is a subject of the relation pointing to an *IriTemplate* instance. In case that subject is a relative URI, default behavior *SHOULD* be used as fallback.

The example below allows to make the product of an IRI template expansion relative to the *http://api.example.com/an-issue/* resource by using it as its base URI, which further enables the *some:operation* to be moved to i.e. API documentation level rather to inline it.

EXAMPLE 25: Custom base Uri resolution for an Iri template

```
{
  "@context": "http://www.w3.org/ns/hydra/context.jsonld",
  "@id": "http://api.example.com/an-issue/",
  "@type": "Collection",
  "some:operation": {
    "@type": "IriTemplate",
    "template": "/{id}",
    "resolveRelativeTo": "LinkContext",
    "variable": "id",
    "mapping": {...}
  }
}
```

§ 5.4 Description of HTTP Status Codes and Errors

HTTP status codes have well defined semantics and can be used to signal the outcome of an operation. Unfortunately, however, HTTP status codes by themselves are often not specific enough, making it difficult to understand the real cause of an error. For instance, a `429 Too Many Requests` response is rarely informative enough by itself. To address this issue, Hydra defines a *Status* class which allows additional information to be associated with an HTTP status code.

EXAMPLE 26: Associating additional information to an HTTP status code

```
{
  "@context": "http://www.w3.org/ns/hydra/context.jsonld",
  "@type": "Status",
  "possibleStatus": 429,
  "title": "Too Many Requests",
  "description": "A maximum of 500 requests per hour and user is allowed.",
  ...
}
```

An *ApiDocumentation* or an *Operation* may document the status codes that might be returned by the server using the *possibleStatus* property as described in § 4.2

Documenting a Web API. This allows a developer to understand what to expect when invoking an operation. It has, however, not to be considered as an extensive list of all potentially returned status codes; it is merely a hint. Developers should expect to encounter other HTTP status codes as well.

A server may also return a *Status* directly in a response. When doing so, it often makes sense to subclass the *Status* to make its semantics more explicit. Hydra defines just one such subclass, namely the *Error* class. This provides an extensible framework to communicate error details to a client.

Furthermore, a *Status* or *Error* returned by the server can also be given an identifier. When dereferenced, the *Error* resource can provide more detailed information or possible ways to resolve the problem, if applicable.

EXAMPLE 27: Using Hydra's Error class to describe errors

```
HTTP/1.1 400 Bad Request
Content-Type: application/ld+json

{
  "@context": "http://www.w3.org/ns/hydra/context.jsonld",
  "@type": "Error",
  "@id": "http://api.example.com/error-details/1234",
  "title": "An error occurred",
  "description": "Typically, a specialization of this class is used in practice.",
  ...
}
```

§ 5.5 Client initiated pagination

There are situations when a client would like to provide a specific collection limitations, i.e. by providing query-language like member offset and limit or some specific page index and number of members per page. This is doable with *offset*/*limit* or *pageIndex*/*limit* predicates.

With those, it is possible to bind a template variables mapped with externally obtained values (i.e. user interaction) the same way as with other mappings.

While the predicates enlisted above accepts non-negative integer numbers, there is also a possibility of providing a custom page reference expressed via *pageReference* predicate. It is possible to provide a custom page identifier (i.e. a GUID or a letter) instead of a number.

§ 6. Classes

hydra:ApiDocumentation

The Hydra API documentation class

Subclass of: hydra:Resource

Status: testing

hydra:BaseUriSource

Provides a base abstract for base Uri source for Iri template resolution.

Subclass of: hydra:Resource

Status: testing

hydra:Class

The class of Hydra classes.

Subclass of: rdfs:Class

Status: testing

hydra:Collection

A collection holding references to a number of related resources.

Subclass of: hydra:Resource

Status: testing

hydra:Error

A runtime error, used to report information beyond the returned status code.

Subclass of: hydra:Status

Status: testing

hydra:IriTemplate

The class of IRI templates.

Status: testing

hydra:IriTemplateMapping

A mapping from an IRI template variable to a property.

Status: testing

hydra:Link

The class of properties representing links.

Subclass of: hydra:Resource, rdf:Property

Status: testing

hydra:Operation

An operation.

Status: testing

hydra:PartialCollectionView

A PartialCollectionView describes a partial view of a Collection. Multiple PartialCollectionViews can be connected with the the next/previous properties to allow a client to retrieve all members of the collection.

Subclass of: hydra:Resource

Status: testing

hydra:Resource

The class of dereferenceable resources by means a client can attempt to dereference; however, the received responses should still be verified.

Status: testing

hydra:Status

Additional information about a status code that might be returned.

Status: testing

hydra:SupportedProperty

A property known to be supported by a Hydra class.

Status: testing

hydra:TemplatedLink

A templated link.

Subclass of: hydra:Resource, rdf:Property

Status: testing

hydra:VariableRepresentation

A representation specifies how to serialize variable values into strings.

Status: testing

§ 7. Properties

hydra:apiDocumentation

A link to the API documentation

Domain: hydra:Resource

Range: hydra:ApiDocumentation

Status: testing

hydra:collection

Collections somehow related to this resource.

Range: hydra:Collection

Status: testing

hydra:description

A description.

Range: undefined

Subproperty of: rdfs:comment

Status: testing

hydra:entrypoint

A link to main entry point of the Web API

Domain: hydra:ApiDocumentation

Range: undefined

Status: testing

hydra:expects

The information expected by the Web API.

Domain: hydra:Operation

Status: testing

hydra:expectsHeader

Specification of the header expected by the operation.

Domain: hydra:Operation

Range: undefined

Status: testing

hydra:first

The first resource of an interlinked set of resources.

Domain: hydra:Resource

Range: undefined

Status: testing

hydra:freetextQuery

A property representing a freetext query.

Domain: hydra:Resource

Range: undefined

Status: testing

hydra:last

The last resource of an interlinked set of resources.

Domain: hydra:Resource

Range: undefined

Status: testing

hydra:limit

Instructs to limit set only to N elements.

Range: undefined

Status: testing

hydra:mapping

A variable-to-property mapping of the IRI template.

Domain: hydra:IriTemplate

Range: hydra:IriTemplateMapping

Status: testing

hydra:member

A member of the collection

Domain: hydra:Collection

Status: testing

hydra:method

The HTTP method.

Domain: hydra:Operation

Range: undefined

Status: testing

hydra:next

The resource following the current instance in an interlinked set of resources.

Domain: hydra:Resource

Range: undefined

Status: testing

hydra:offset

Instructs to skip N elements of the set.

Range: undefined

Status: testing

hydra:operation

An operation supported by the Hydra resource

Domain: hydra:Resource

Range: hydra:Operation

Status: testing

hydra:pageIndex

Instructs to provide a specific page of the collection at a given index.

Range: undefined

Subproperty of: [object Object]

Status: testing

hydra:pageReference

Instructs to provide a specific page reference of the collection.

Status: testing

hydra:possibleStatus

A status that might be returned by the Web API (other statuses should be expected and properly handled as well)

Range: hydra:Status

Status: testing

hydra:previous

The resource preceding the current instance in an interlinked set of resources.

Domain: hydra:Resource

Range: undefined

Status: testing

hydra:property

A property

Range: undefined

Status: testing

hydra:readable

True if the client can retrieve the property's value, false otherwise.

Domain: hydra:SupportedProperty

Range: undefined

Status: testing

hydra:required

True if the property is required, false otherwise.

Range: undefined

Status: testing

hydra:resolveRelativeUsing

undefined

Domain: hydra:IriTemplate

Range: hydra:BaseUriSource

Status: testing

hydra:returns

The information returned by the Web API on success

Domain: hydra:Operation

Status: testing

hydra:returnsHeader

Name of the header returned by the operation.

Domain: hydra:Operation

Range: undefined

Status: testing

hydra:search

A IRI template that can be used to query a collection.

Domain: hydra:Resource

Range: hydra:IriTemplate

Status: testing

hydra:statusCode

The HTTP status code

Domain: hydra:Status

Range: undefined

Status: testing

hydra:supportedClass

A class known to be supported by the Web API

Domain: hydra:ApiDocumentation

Range: undefined

Status: testing

hydra:supportedOperation

An operation supported by instances of the specific Hydra class or the target of the Hydra link

Range: hydra:Operation

Status: testing

hydra:supportedProperty

The properties known to be supported by a Hydra class

Domain: undefined

Range: hydra:SupportedProperty

Status: testing

hydra:template

A templated string with placeholders. The literal's datatype indicates the template syntax; if not specified, hydra:Rfc6570Template is assumed.

Domain: hydra:IriTemplate

Range: hydra:Rfc6570Template

Status: testing

hydra:title

A title, often used along with a description.

Range: undefined

Subproperty of: rdfs:label

Status: testing

hydra:totalItems

The total number of items referenced by a collection.

Domain: hydra:Collection

Range: undefined

Status: testing

hydra:variable

An IRI template variable

Domain: hydra:IriTemplateMapping

Range: undefined

Status: testing

hydra:variableRepresentation

The representation format to use when expanding the IRI template.

Domain: hydra:IriTemplateMapping

Range: hydra:VariableRepresentation

Status: testing

hydra:view

A specific view of a resource.

Status: testing

hydra:writable

True if the client can change the property's value, false otherwise.

Domain: hydra:SupportedProperty

Range: undefined

Status: testing

§ 8. Acknowledgements

This section is non-normative.

The authors would like to thank the following individuals for contributing their ideas and providing feedback for writing this specification: Arnau Siches, elf Pavlik, Karol Szczepański, Mark Baker, Martijn Faassen, Matthias Lehmann, Ruben Verborgh, Ryan J. McDonough, Sam Goto, Thomas Hoppe, Tomasz Pluskiewicz, @wasabiwimp (on GitHub).

§ A. The Hydra Core Vocabulary in JSON-LD

This section is non-normative.

```
{
  "@context": {
    "hydra": "http://www.w3.org/ns/hydra/core#",
    "rdf": "http://www.w3.org/1999/02/22-rdf-syntax-ns#",
    "rdfs": "http://www.w3.org/2000/01/rdf-schema#",
    "xsd": "http://www.w3.org/2001/XMLSchema#",
    "owl": "http://www.w3.org/2002/07/owl#",
    "vs": "http://www.w3.org/2003/06/sw-vocab-status/ns#",
    "defines": {
      "@reverse": "rdfs:isDefinedBy"
    },
    "comment": "rdfs:comment",
    "label": "rdfs:label",
    "domain": {
      "@id": "rdfs:domain",
      "@type": "@id"
    },
    "range": {
      "@id": "rdfs:range",
      "@type": "@id"
    },
    "subClassOf": {
      "@id": "rdfs:subClassOf",
      "@type": "@id",
      "@container": "@set"
    },
    "subPropertyOf": {
      "@id": "rdfs:subPropertyOf",
      "@type": "@id",
      "@container": "@set"
    },
    "seeAlso": {
        "@id": "rdfs:seeAlso",
        "@type": "@id"
      },
      "status": "vs:term_status"
    },
    "@id": "http://www.w3.org/ns/hydra/core",
    "defines": [
      {
        "@id": "hydra:Resource",
        "@type": "hydra:Class",
        "comment": "The class of dereferenceable resources by means a client can attempt to
dereference; however, the received responses should still be verified.",
        "label": "Hydra Resource",
        "status": "testing"
      },
      {
        "@id": "hydra:Class",
        "@type": [
          "hydra:Resource",
          "rdfs:Class"
        ],
        "comment": "The class of Hydra classes.",
        "label": "Hydra Class",
        "subClassOf": [
          "rdfs:Class"
        ],
        "status": "testing"
      },
      {
        "@id": "hydra:Link",
        "@type": "hydra:Class",
        "comment": "The class of properties representing links.",
        "label": "Link",
        "subClassOf": [
          "hydra:Resource",
          "rdf:Property"
        ],
        "status": "testing"
      },
      {
        "@id": "hydra:apiDocumentation",
        "@type": "hydra:Link",
        "comment": "A link to the API documentation",
        "domain": "hydra:Resource",
        "label": "apiDocumentation",
        "range": "hydra:ApiDocumentation",
        "status": "testing"
      },
      {
        "@id": "hydra:ApiDocumentation",
        "@type": "hydra:Class",
      "comment": "The Hydra API documentation class",
      "label": "ApiDocumentation",
      "subClassOf": [
        "hydra:Resource"
      ],
      "status": "testing"
    },
    {
      "@id": "hydra:entrypoint",
      "@type": "hydra:Link",
      "comment": "A link to main entry point of the Web API",
      "domain": "hydra:ApiDocumentation",
      "label": "entrypoint",
      "range": "hydra:Resource",
      "status": "testing"
    },
    {
      "@id": "hydra:supportedClass",
      "@type": "hydra:Link",
      "comment": "A class known to be supported by the Web API",
      "domain": "hydra:ApiDocumentation",
      "label": "supported classes",
      "range": "rdfs:Class",
      "status": "testing"
    },
    {
      "@id": "hydra:possibleStatus",
      "@type": "hydra:Link",
      "http://schema.org/domainIncludes": [
        {
          "@id": "hydra:ApiDocumentation"
        },
        {
          "@id": "hydra:Operation"
        }
      ],
      "comment": "A status that might be returned by the Web API (other statuses should
be expected and properly handled as well)",
      "label": "possible status",
      "range": "hydra:Status",
      "status": "testing"
    },
    {
      "@id": "hydra:supportedProperty",
      "@type": "hydra:Link",
      "comment": "The properties known to be supported by a Hydra class",
      "domain": "rdfs:Class",
      "label": "supported properties",
      "range": "hydra:SupportedProperty",
      "status": "testing"
    },
    {
      "@id": "hydra:SupportedProperty",
      "@type": "hydra:Class",
      "comment": "A property known to be supported by a Hydra class.",
      "label": "Supported Property",
      "status": "testing"
    },
    {
      "@id": "hydra:property",
      "@type": "rdf:Property",
      "http://schema.org/domainIncludes": [
        {
          "@id": "hydra:SupportedProperty"
        },
        {
          "@id": "hydra:IriTemplateMapping"
        }
      ],
      "comment": "A property",
      "label": "property",
      "range": "rdf:Property",
      "status": "testing"
    },
    {
      "@id": "hydra:required",
      "@type": "rdf:Property",
      "http://schema.org/domainIncludes": [
        {
          "@id": "hydra:SupportedProperty"
        },
        {
          "@id": "hydra:IriTemplateMapping"
        }
      ],
      "comment": "True if the property is required, false otherwise.",
      "label": "required",
      "range": "xsd:boolean",
      "status": "testing"
    },
    {
      "@id": "hydra:readable",
      "@type": "rdf:Property",
      "comment": "True if the client can retrieve the property's value, false
otherwise.",
      "domain": "hydra:SupportedProperty",
      "label": "readable",
      "range": "xsd:boolean",
      "status": "testing"
    },
    {
      "@id": "hydra:writable",
      "@type": "rdf:Property",
      "comment": "True if the client can change the property's value, false otherwise.",
      "domain": "hydra:SupportedProperty",
      "label": "writable",
      "range": "xsd:boolean",
      "status": "testing"
    },
    {
      "@id": "hydra:writeable",
      "comment": "This property is left for compatibility purposes and hydra:writable
should be used instead.",
      "label": "writable",
      "subPropertyOf": [
        "hydra:writable"
      ],
      "status": "archaic"
    },
    {
      "@id": "hydra:supportedOperation",
      "@type": "hydra:Link",
      "http://schema.org/domainIncludes": [
        {
          "@id": "rdfs:Class"
        },
        {
          "@id": "hydra:Class"
        },
        {
          "@id": "hydra:Link"
        },
        {
          "@id": "hydra:TemplatedLink"
        },
        {
          "@id": "hydra:SupportedProperty"
        }
      ],
      "comment": "An operation supported by instances of the specific Hydra class or the
target of the Hydra link",
      "label": "supported operation",
      "range": "hydra:Operation",
      "status": "testing"
    },
    {
      "@id": "hydra:operation",
      "@type": "hydra:Link",
      "comment": "An operation supported by the Hydra resource",
      "domain": "hydra:Resource",
      "label": "operation",
      "range": "hydra:Operation",
      "status": "testing"
```

```
},
{
  "@id": "hydra:Operation",
  "@type": "hydra:Class",
  "comment": "An operation.",
  "label": "Operation",
  "status": "testing"
},
{
  "@id": "hydra:method",
  "@type": "rdf:Property",
  "comment": "The HTTP method.",
  "domain": "hydra:Operation",
  "label": "method",
  "range": "xsd:string",
  "status": "testing"
},
{
  "@id": "hydra:expects",
  "@type": "hydra:Link",
  "http://schema.org/rangeIncludes": [
    {
      "@id": "rdfs:Resource"
    },
    {
      "@id": "hydra:Resource"
    },
    {
      "@id": "rdfs:Class"
    },
    {
      "@id": "hydra:Class"
    }
  ],
  "comment": "The information expected by the Web API.",
  "domain": "hydra:Operation",
  "label": "expects",
  "status": "testing"
},
{
  "@id": "hydra:returns",
  "@type": "hydra:Link",
  "http://schema.org/rangeIncludes": [
    {
      "@id": "rdfs:Resource"
    },
    {
      "@id": "hydra:Resource"
    },
    {
      "@id": "rdfs:Class"
```

```
          },
          {
            "@id": "hydra:Class"
          }
        ],
        "comment": "The information returned by the Web API on success",
        "domain": "hydra:Operation",
        "label": "returns",
        "status": "testing"
      },
      {
        "@id": "hydra:Status",
        "@type": "hydra:Class",
        "comment": "Additional information about a status code that might be returned.",
        "label": "Status code description",
        "status": "testing"
      },
      {
        "@id": "hydra:statusCode",
        "@type": "rdf:Property",
        "comment": "The HTTP status code",
        "domain": "hydra:Status",
        "label": "status code",
        "range": "xsd:integer",
        "status": "testing"
      },
      {
        "@id": "hydra:title",
        "@type": "rdf:Property",
        "http://schema.org/domainIncludes": [
          {
            "@id": "hydra:ApiDocumentation"
          },
          {
            "@id": "hydra:Status"
          },
          {
            "@id": "hydra:Class"
          },
          {
            "@id": "hydra:SupportedProperty"
          },
          {
            "@id": "hydra:Operation"
          },
          {
            "@id": "hydra:Link"
          },
          {
            "@id": "hydra:TemplatedLink"
          }
      ],
      "comment": "A title, often used along with a description.",
      "label": "title",
      "range": "xsd:string",
      "subPropertyOf": [
        "rdfs:label"
      ],
      "status": "testing"
    },
    {
      "@id": "hydra:description",
      "@type": "rdf:Property",
      "http://schema.org/domainIncludes": [
        {
          "@id": "hydra:ApiDocumentation"
        },
        {
          "@id": "hydra:Status"
        },
        {
          "@id": "hydra:Class"
        },
        {
          "@id": "hydra:SupportedProperty"
        },
        {
          "@id": "hydra:Operation"
        },
        {
          "@id": "hydra:Link"
        },
        {
          "@id": "hydra:TemplatedLink"
        }
      ],
      "comment": "A description.",
      "label": "description",
      "range": "xsd:string",
      "subPropertyOf": [
        "rdfs:comment"
      ],
      "status": "testing"
    },
    {
      "@id": "hydra:Error",
      "@type": "hydra:Class",
      "comment": "A runtime error, used to report information beyond the returned status
code.",
      "label": "Error",
      "subClassOf": [
        "hydra:Status"
        ],
        "status": "testing"
      },
      {
        "@id": "hydra:Collection",
        "@type": "hydra:Class",
        "comment": "A collection holding references to a number of related resources.",
        "label": "Collection",
        "subClassOf": [
          "hydra:Resource"
        ],
        "status": "testing"
      },
      {
        "@id": "hydra:collection",
        "@type": "hydra:Link",
        "comment": "Collections somehow related to this resource.",
        "label": "collection",
        "range": "hydra:Collection",
        "status": "testing"
      },
      {
        "@id": "hydra:memberAssertion",
        "comment": "Semantics of each member provided by the collection.",
        "domain": "hydra:Collection",
        "label": "member assertion",
        "status": "testing"
      },
      {
        "@id": "hydra:manages",
        "comment": "This predicate is left for compatibility purposes and
hydra:memberAssertion should be used instead.",
        "label": "manages",
        "subPropertyOf": [
          "hydra:memberAssertion"
        ],
        "status": "archaic"
      },
      {
        "@id": "hydra:subject",
        "comment": "The subject.",
        "label": "subject",
        "status": "testing"
      },
      {
        "@id": "hydra:object",
        "comment": "The object.",
        "label": "object",
        "status": "testing"
      },
      {
```

```
      "@id": "hydra:member",
      "@type": "hydra:Link",
      "comment": "A member of the collection",
      "domain": "hydra:Collection",
      "label": "member",
      "status": "testing"
    },
    {
      "@id": "hydra:view",
      "@type": "hydra:Link",
      "comment": "A specific view of a resource.",
      "label": "view",
      "status": "testing"
    },
    {
      "@id": "hydra:PartialCollectionView",
      "@type": "hydra:Class",
      "comment": "A PartialCollectionView describes a partial view of a Collection.
Multiple PartialCollectionViews can be connected with the the next/previous properties to
allow a client to retrieve all members of the collection.",
      "label": "PartialCollectionView",
      "subClassOf": [
        "hydra:Resource"
      ],
      "status": "testing"
    },
    {
      "@id": "hydra:totalItems",
      "@type": "rdf:Property",
      "comment": "The total number of items referenced by a collection.",
      "domain": "hydra:Collection",
      "label": "total items",
      "range": "xsd:integer",
      "status": "testing"
    },
    {
      "@id": "hydra:first",
      "@type": "hydra:Link",
      "comment": "The first resource of an interlinked set of resources.",
      "domain": "hydra:Resource",
      "label": "first",
      "range": "hydra:Resource",
      "status": "testing"
    },
    {
      "@id": "hydra:last",
      "@type": "hydra:Link",
      "comment": "The last resource of an interlinked set of resources.",
      "domain": "hydra:Resource",
      "label": "last",
      "range": "hydra:Resource",
      "status": "testing"
    },
    {
      "@id": "hydra:next",
      "@type": "hydra:Link",
      "comment": "The resource following the current instance in an interlinked set of
resources.",
      "domain": "hydra:Resource",
      "label": "next",
      "range": "hydra:Resource",
      "status": "testing"
    },
    {
      "@id": "hydra:previous",
      "@type": "hydra:Link",
      "comment": "The resource preceding the current instance in an interlinked set of
resources.",
      "domain": "hydra:Resource",
      "label": "previous",
      "range": "hydra:Resource",
      "status": "testing"
    },
    {
      "@id": "hydra:search",
      "@type": "hydra:TemplatedLink",
      "comment": "A IRI template that can be used to query a collection.",
      "domain": "hydra:Resource",
      "label": "search",
      "range": "hydra:IriTemplate",
      "status": "testing"
    },
    {
      "@id": "hydra:freetextQuery",
      "@type": "rdf:Property",
      "comment": "A property representing a freetext query.",
      "domain": "hydra:Resource",
      "label": "freetext query",
      "range": "xsd:string",
      "status": "testing"
    },
    {
      "@id": "hydra:TemplatedLink",
      "@type": "hydra:Class",
      "comment": "A templated link.",
      "label": "Templated Link",
      "subClassOf": [
        "hydra:Resource",
        "rdf:Property"
      ],
      "status": "testing"
    },
    {
      "@id": "hydra:IriTemplate",
      "@type": "hydra:Class",
      "comment": "The class of IRI templates.",
      "label": "IRI Template",
      "status": "testing"
    },
    {
      "@id": "hydra:template",
      "@type": "rdf:Property",
      "comment": "A templated string with placeholders. The literal's datatype indicates
the template syntax; if not specified, hydra:Rfc6570Template is assumed.",
      "domain": "hydra:IriTemplate",
      "label": "template",
      "range": "hydra:Rfc6570Template",
      "seeAlso": "hydra:Rfc6570Template",
      "status": "testing"
    },
    {
      "@id": "hydra:Rfc6570Template",
      "@type": "rdfs:Datatype",
      "comment": "An IRI template as defined by RFC6570.",
      "label": "RFC6570 IRI template",
      "range": "xsd:string",
      "seeAlso": "http://tools.ietf.org/html/rfc6570",
      "status": "testing"
    },
    {
      "@id": "hydra:variableRepresentation",
      "@type": "rdf:Property",
      "comment": "The representation format to use when expanding the IRI template.",
      "domain": "hydra:IriTemplateMapping",
      "label": "variable representation",
      "range": "hydra:VariableRepresentation",
      "status": "testing"
    },
    {
      "@id": "hydra:VariableRepresentation",
      "@type": "hydra:Class",
      "comment": "A representation specifies how to serialize variable values into
strings.",
      "label": "VariableRepresentation",
      "status": "testing"
    },
    {
      "@id": "hydra:BasicRepresentation",
      "@type": "hydra:VariableRepresentation",
      "comment": "A representation that serializes just the lexical form of a variable
value, but omits language and type information.",
      "label": "BasicRepresentation",
      "status": "testing"
    },
    {
      "@id": "hydra:ExplicitRepresentation",
      "@type": "hydra:VariableRepresentation",
      "comment": "A representation that serializes a variable value including its
language and type information and thus differentiating between IRIs and literals.",
      "label": "ExplicitRepresentation",
      "status": "testing"
    },
    {
      "@id": "hydra:mapping",
      "@type": "rdf:Property",
      "comment": "A variable-to-property mapping of the IRI template.",
      "domain": "hydra:IriTemplate",
      "label": "mapping",
      "range": "hydra:IriTemplateMapping",
      "status": "testing"
    },
    {
      "@id": "hydra:IriTemplateMapping",
      "@type": "hydra:Class",
      "comment": "A mapping from an IRI template variable to a property.",
      "label": "IriTemplateMapping",
      "status": "testing"
    },
    {
      "@id": "hydra:variable",
      "@type": "rdf:Property",
      "comment": "An IRI template variable",
      "domain": "hydra:IriTemplateMapping",
      "label": "variable",
      "range": "xsd:string",
      "status": "testing"
    },
    {
      "@id": "hydra:resolveRelativeUsing",
      "@type": "rdf:Property",
      "domain": "hydra:IriTemplate",
      "label": "relative Uri resolution",
      "range": "hydra:BaseUriSource",
      "status": "testing"
    },
    {
      "@id": "hydra:BaseUriSource",
      "@type": "hydra:Class",
      "comment": "Provides a base abstract for base Uri source for Iri template
resolution.",
      "label": "Base Uri source",
      "subClassOf": [
        "hydra:Resource"
      ],
      "status": "testing"
    },
    {
      "@id": "hydra:Rfc3986",
      "@type": "hydra:BaseUriSource",
      "comment": "States that the base Uri should be established using RFC 3986 reference
resolution algorithm specified in section 5.",
      "label": "RFC 3986 based",
      "status": "testing"
    },
    {
      "@id": "hydra:LinkContext",
      "@type": "hydra:BaseUriSource",
      "comment": "States that the link's context IRI, as defined in RFC 5988, should be
used as the base Uri",
      "label": "Link context",
      "status": "testing"
    },
    {
      "@id": "hydra:offset",
      "@type": "rdf:Property",
      "comment": "Instructs to skip N elements of the set.",
      "label": "skip",
      "range": "xsd:nonNegativeInteger",
      "status": "testing"
    },
    {
      "@id": "hydra:limit",
      "@type": "rdf:Property",
      "comment": "Instructs to limit set only to N elements.",
      "label": "take",
      "range": "xsd:nonNegativeInteger",
      "status": "testing"
    },
    {
      "@id": "hydra:pageIndex",
      "@type": "rdf:Property",
      "comment": "Instructs to provide a specific page of the collection at a given
index.",
      "label": "page index",
      "range": "xsd:nonNegativeInteger",
      "subPropertyOf": [
        "hydra:pageReference"
      ],
      "status": "testing"
    },
    {
      "@id": "hydra:pageReference",
      "@type": "rdf:Property",
      "comment": "Instructs to provide a specific page reference of the collection.",
      "label": "page reference",
        "status": "testing"
      },
      {
        "@id": "hydra:returnsHeader",
        "@type": "rdf:Property",
        "comment": "Name of the header returned by the operation.",
        "domain": "hydra:Operation",
        "label": "returns header",
        "range": "xsd:string",
        "status": "testing"
      },
      {
        "@id": "hydra:expectsHeader",
        "@type": "rdf:Property",
        "comment": "Specification of the header expected by the operation.",
        "domain": "hydra:Operation",
        "label": "expects header",
        "range": "xsd:string",
        "status": "testing"
      }
    ],
    "@type": "owl:Ontology",
    "http://creativecommons.org/ns#attributionName": "Hydra W3C Community Group",
    "http://creativecommons.org/ns#attributionURL": {
      "@id": "http://www.hydra-cg.com/"
    },
    "http://creativecommons.org/ns#license": {
      "@id": "http://creativecommons.org/licenses/by/4.0/"
    },
    "http://purl.org/dc/terms/description": "The Hydra Core Vocabulary is a lightweight
  vocabulary to create hypermedia-driven Web APIs. By specifying a number of concepts
  commonly used in Web APIs it enables the creation of generic API clients.",
    "http://purl.org/dc/terms/publisher": "Hydra W3C Community Group",
    "http://purl.org/dc/terms/rights": "Copyright © 2012-2014 the Contributors to the Hydra
  Core Vocabulary Specification",
    "http://purl.org/vocab/vann/preferredNamespacePrefix": "hydra",
    "comment": "A lightweight vocabulary for hypermedia-driven Web APIs",
    "label": "The Hydra Core Vocabulary"
  }
```

§ B. References

§ B.1 Normative references

[RFC2119]
Key words for use in RFCs to Indicate Requirement Levels. S. Bradner. IETF. March 1997. Best Current Practice. URL: https://tools.ietf.org/html/rfc2119

[RFC6570]

URI Template. J. Gregorio; R. Fielding; M. Hadley; M. Nottingham; D. Orchard. IETF. March 2012. Proposed Standard. URL: https://tools.ietf.org/html/rfc6570

§ B.2 Informative references

[BCP47]
Tags for Identifying Languages. A. Phillips; M. Davis. IETF. September 2009. IETF Best Current Practice. URL: https://tools.ietf.org/html/bcp47

[RFC5988]
Web Linking. M. Nottingham. IETF. October 2010. Proposed Standard. URL: https://tools.ietf.org/html/rfc5988

[Turtle]
RDF 1.1 Turtle. Eric Prud'hommeaux; Gavin Carothers. W3C. 25 February 2014. W3C Recommendation. URL: https://www.w3.org/TR/turtle/

↑