Internet Engineering Task Force (IETF)
Request for Comments: 8982
Category: Standards Track
ISSN: 2070-1721

M. Loffredo
M. Martinelli
IIT-CNR/Registro.it

February 2021

Registration Data Access Protocol (RDAP) Partial Response

Abstract

The Registration Data Access Protocol (RDAP) does not include capabilities to request partial responses. Servers will only return full responses that include all of the information that a client is authorized to receive. A partial response capability that limits the amount of information returned, especially in the case of search queries, could bring benefits to both clients and servers. This document describes an RDAP query extension that allows clients to specify their preference for obtaining a partial response.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc8982.

Table of Contents

1. Introduction

The use of partial responses in RESTful API [REST] design is very common. The rationale is quite simple: instead of returning objects in API responses with all data fields, only a subset of the fields in each result object is returned. The benefit is obvious: less data transferred over the network means less bandwidth usage, faster server responses, less CPU time spent both on the server and the client, and less memory usage on the client.

Currently, RDAP does not provide a client with any way to request a partial response. Servers can only provide the client with a full response [RFC7483]. Servers cannot limit the amount of information returned in a response based on a client's preferences, and this creates inefficiencies.

The protocol described in this specification extends RDAP search capabilities to enable partial responses through the provisioning of predefined sets of fields that clients can submit to an RDAP service by adding a new query parameter. The service is implemented using the Hypertext Transfer Protocol (HTTP) [RFC7230] and the conventions described in [RFC7480].

1.1. Conventions Used in This Document

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.

2. RDAP Path Segment Specification

The path segment defined in this section is an OPTIONAL extension of search path segments defined in [RFC7482]. This document defines an RDAP query parameter, "fieldSet", whose value is a non-empty string identifying a server-defined set of fields returned in place of the full response. The field sets supported by a server are usually described in out-of-band documents (e.g., RDAP profile) together with other features. Moreover, this document defines in Section 2.1 an in-band mechanism by means of which servers can provide clients with basic information about the supported field sets.

The following is an example of an RDAP query including the "fieldSet" parameter:

https://example.com/rdap/domains?name=example*.com&fieldSet=afieldset

This solution can be implemented by RDAP providers with less effort than field selection and is easily requested by clients. The considerations that have led to this solution are described in more detail in Appendix A.

2.1. Subsetting Metadata

According to most advanced principles in REST design, collectively known as "Hypermedia as the Engine of Application State" (HATEOAS) [HATEOAS], a client entering a REST application through an initial URI should use server-provided links to dynamically discover

available actions and access the resources it needs. In this way, the client is not required to have prior knowledge of the service nor, consequently, to hard-code the URIs of different resources. This allows the server to make URI changes as the API evolves without breaking clients. Definitively, a REST service should be as self-descriptive as possible.

Therefore, servers implementing the query parameter described in this specification SHOULD provide additional information in their responses about the available field sets. Such information is collected in a new JSON data structure named "subsetting_metadata" containing the following properties:

"currentFieldSet": "String" (REQUIRED)
: either the value of the "fieldSet" parameter as specified in the query string, or the field set applied by default.

"availableFieldSets": "AvailableFieldSet[]" (OPTIONAL)
: an array of objects, with each element describing an available field set. The AvailableFieldSet object includes the following members:

  "name": "String" (REQUIRED)
  : the field set name.

  "default": "Boolean" (REQUIRED)
  : indicator of whether the field set is applied by default. An RDAP server MUST define only one default field set.

  "description": "String" (OPTIONAL)
  : a human-readable description of the field set.

  "links": "Link[]" (OPTIONAL)
  : an array of links as described in [RFC8288] containing the query string that applies the field set (see Section 2.1.2).

2.1.1. RDAP Conformance

Servers returning the "subsetting_metadata" section in their responses MUST include "subsetting" in the rdapConformance array.

2.1.2. Representing Subsetting Links

An RDAP server MAY use the "links" array of the "subsetting_metadata" element to provide ready-made references [RFC8288] to the available field sets (Figure 1). The target URI in each link is the reference to an alternative to the current view of results identified by the context URI.

The "value", "rel", and "href" JSON values MUST be specified. All other JSON values are OPTIONAL.

```
{
  "rdapConformance": [
    "rdap_level_0",
    "subsetting"
  ],
  ...
  "subsetting_metadata": {
    "currentFieldSet": "afieldset",
    "availableFieldSets": [
      {
      "name": "anotherfieldset",
      "description": "Contains some fields",
      "default": false,
      "links": [
```

```
        {
        "value": "https://example.com/rdap/domains?name=example*.com
                  &fieldSet=afieldset",
        "rel": "alternate",
        "href": "https://example.com/rdap/domains?name=example*.com
                 &fieldSet=anotherfieldset",
        "title": "Result Subset Link",
        "type": "application/rdap+json"
        }
      ]
      },
    ...
    ]
  },
  ...
  "domainSearchResults": [
    ...
  ]
}
```

Figure 1: Example of a "subsetting_metadata" Instance

3. Dealing with Relationships

Representation of second-level objects within a field set produces additional considerations. Since the representation of the topmost returned objects will vary according to the field set in use, the response may contain no relationships (e.g., for an abbreviated field set) or may contain associated objects as in a normal RDAP query response. Each field set can indicate the format of the additional objects to be returned, in the same manner that the format of the topmost objects is controlled by the field set.

4. Basic Field Sets

This section defines three basic field sets that servers MAY implement to facilitate their interaction with clients:

"id": The server provides only the key field; "handle" for entities, and "ldhName" for domains and nameservers. If a returned domain or nameserver is an Internationalized Domain Name (IDN) [RFC5890], then the "unicodeName" field MUST additionally be included in the response. This field set could be used when the client wants to obtain a collection of object identifiers (Figure 2).

"brief": The field set contains the fields that can be included in a "short" response. This field set could be used when the client is asking for a subset of the full response that provides only basic knowledge of each object.

"full": The field set contains all of the information the server can provide for a particular object.

The "objectClassName" field is implicitly included in each of the above field sets. RDAP providers SHOULD include a "links" field indicating the "self" link relationship. RDAP providers MAY also add any property providing service information.

Fields included in the "brief" and "full" field set responses MUST take into account the user’s access and authorization levels.

```
{
  "rdapConformance": [
    "rdap_level_0",
    "subsetting"
  ],
```

```
  ...
  "domainSearchResults": [
    {
      "objectClassName": "domain",
      "ldhName": "example1.com",
      "links": [
        {
        "value": "https://example.com/rdap/domain/example1.com",
        "rel": "self",
        "href": "https://example.com/rdap/domain/example1.com",
        "type": "application/rdap+json"
        }
      ]
    },
    {
      "objectClassName": "domain",
      "ldhName": "example2.com",
      "links": [
        {
        "value": "https://example.com/rdap/domain/example2.com",
        "rel": "self",
        "href": "https://example.com/rdap/domain/example2.com",
        "type": "application/rdap+json"
        }
      ]
    },
    ...
  ]
}
```

Figure 2: Example of RDAP Response According to the "id" Field Set

5. Negative Answers

Each request including an empty or unsupported "fieldSet" value MUST produce an HTTP 400 (Bad Request) response code. Optionally, the response MAY include additional information regarding the supported field sets in the HTTP entity body (Figure 3).

```
{
    "errorCode": 400,
    "title": "Field set 'unknownfieldset' is not valid",
    "description": [
        "Supported field sets are: 'afieldset', 'anotherfieldset'."
    ]

}
```

Figure 3: Example of RDAP Error Response Due to an Invalid Field Set Included in the Request

6. IANA Considerations

IANA has registered the following value in the "RDAP Extensions" registry:

Extension identifier: subsetting
Registry operator: Any
Published specification: RFC 8982
Contact: IETF <iesg@ietf.org>
Intended usage: This extension describes a best practice for partial response provisioning.

7. Security Considerations

A search query typically requires more server resources (such as

memory, CPU cycles, and network bandwidth) when compared to a lookup query. This increases the risk of server resource exhaustion and subsequent denial of service. This risk can be mitigated by supporting the return of partial responses combined with other strategies (e.g., restricting search functionality, limiting the rate of search requests, and truncating and paging results).

Support for partial responses gives RDAP operators the ability to implement data access control policies based on the HTTP authentication mechanisms described in [RFC7481]. RDAP operators can vary the information returned in RDAP responses based on a client's access and authorization levels. For example:

* the list of fields for each set can differ based on the client's access and authorization levels;

* the set of available field sets could be restricted based on the client's access and authorization levels.

Servers can also define different result limits according to the available field sets, so a more flexible truncation strategy can be implemented. The new query parameter presented in this document provides RDAP operators with a way to implement a server that reduces inefficiency risks.

8. References

8.1. Normative References

[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/info/rfc2119>.

[RFC5890] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", RFC 5890, DOI 10.17487/RFC5890, August 2010, <https://www.rfc-editor.org/info/rfc5890>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <https://www.rfc-editor.org/info/rfc7230>.

[RFC7480] Newton, A., Ellacott, B., and N. Kong, "HTTP Usage in the Registration Data Access Protocol (RDAP)", RFC 7480, DOI 10.17487/RFC7480, March 2015, <https://www.rfc-editor.org/info/rfc7480>.

[RFC7481] Hollenbeck, S. and N. Kong, "Security Services for the Registration Data Access Protocol (RDAP)", RFC 7481, DOI 10.17487/RFC7481, March 2015, <https://www.rfc-editor.org/info/rfc7481>.

[RFC7482] Newton, A. and S. Hollenbeck, "Registration Data Access Protocol (RDAP) Query Format", RFC 7482, DOI 10.17487/RFC7482, March 2015, <https://www.rfc-editor.org/info/rfc7482>.

[RFC7483] Newton, A. and S. Hollenbeck, "JSON Responses for the Registration Data Access Protocol (RDAP)", RFC 7483, DOI 10.17487/RFC7483, March 2015, <https://www.rfc-editor.org/info/rfc7483>.

[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/info/rfc8174>.

[RFC8288] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <https://www.rfc-editor.org/info/rfc8288>.

8.2. Informative References

[CQL] Whitaker, G., "Catnap Query Language Reference", commit d4f402c, September 2017, <https://github.com/gregwhitaker/catnap/wiki/Catnap-Query-Language-Reference>.

[HATEOAS] Jedrzejewski, B., "HATEOAS - a simple explanation", February 2018, <https://www.e4developer.com/2018/02/16/hateoas-simple-explanation/>.

[REST] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", Ph.D. Dissertation, University of California, Irvine, 2000, <https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf>.

Appendix A. Approaches to Partial Response Implementation

Looking at the implementation experiences of partial responses offered by data providers on the web, two approaches are observed:

* the client explicitly describes the data fields to be returned;

* the client describes a name identifying a server-defined set of data fields.

The former is more flexible than the latter because clients can specify all the data fields they need. However, it has some drawbacks:

* Fields have to be declared according to a given syntax. This is a simple task when the data structure of the object is flat, but it is much more difficult when the object has a tree structure like that of a JSON object. The presence of arrays and deep nested objects complicate both the syntax definition of the query and, consequently, the processing required on the server side.

* Clients need to recognize the returned data structure to avoid cases when the requested fields are invalid.

* The request of some fields might not match the client’s access and authorization levels. Clients might request unauthorized fields, and servers have to define a strategy for responding such as always returning an error response or returning a response that ignores the unauthorized fields.

A.1. Specific Issues Raised by RDAP

In addition to those listed above, RDAP responses raise some specific issues:

* Relevant entity object information is included in a jCard, but such information cannot be easily selected because it is split into the items of a jagged array.

* RDAP responses contain some properties providing service information (e.g., rdapConformance, links, notices, remarks, etc.), which are not normally selected but are just as important. They could be returned anyway but, in this case, the server would

provide unrequested data.

It is possible to address these issues. For example, the Catnap Query Language [CQL] is a comprehensive expression language that can be used to customize the JSON response of a RESTful web service. Application of CQL to RDAP responses would explicitly identify the output fields that would be acceptable when a few fields are requested but it would become very complicated when processing a larger number of fields. In the following, two CQL expressions for a domain search query are shown (Figure 4). In the first, only objectClassName and ldhName are requested. In the second, the fields of a possible WHOIS-like response are listed.

```
https://example.com/rdap/domains?name=example*.com
        &fields=domainSearchResults(objectClassName,ldhName)

https://example.com/rdap/domains?name=example*.com
        &fields=domainSearchResults(objectClassName,ldhName,
                unicodeName,
                status,
                events(eventAction,eventDate),
                entities(objectClassName,handle,roles),
                nameservers(objectClassName,ldhName))
```

Figure 4: Examples of CQL Expressions for a Domain Search Query

The field set approach seems to facilitate RDAP interoperability. Servers can define basic field sets that, if known to clients, can increase the probability of obtaining a valid response. The usage of field sets makes the query string less complex. Moreover, the definition of predefined sets of fields makes it easier to establish result limits.

Finally, considering that there is no real need for RDAP users to have the maximum flexibility in defining all the possible sets of logically connected fields (e.g., users interested in domains usually need to know the status, the creation date, and the expiry date of each domain), the field set approach is preferred.

Acknowledgements

The authors would like to acknowledge Scott Hollenbeck, Tom Harrison, Karl Heinz Wolf, Jasdip Singh, Patrick Mevzek, Benjamin Kaduk, Roman Danyliw, Murray Kucherawy, Erik Kline, and Robert Wilton for their contribution to this document.

Authors' Addresses

Mario Loffredo
IIT-CNR/Registro.it
Via Moruzzi,1
56124 Pisa
Italy

Email: mario.loffredo@iit.cnr.it
URI: https://www.iit.cnr.it

Maurizio Martinelli
IIT-CNR/Registro.it
Via Moruzzi,1
56124 Pisa
Italy

Email: maurizio.martinelli@iit.cnr.it
URI: https://www.iit.cnr.it