Stream: Internet Engineering Task Force (IETF)
RFC: 8977
Category: Standards Track
Published: January 2021
ISSN: 2070-1721
Authors: M. Loffredo *IIT-CNR/Registro.it* M. Martinelli *IIT-CNR/Registro.it* S. Hollenbeck *Verisign Labs*

RFC 8977
Registration Data Access Protocol (RDAP) Query Parameters for Result Sorting and Paging

Abstract

The Registration Data Access Protocol (RDAP) does not include core functionality for clients to provide sorting and paging parameters for control of large result sets. This omission can lead to unpredictable server processing of queries and client processing of responses. This unpredictability can be greatly reduced if clients can provide servers with their preferences for managing large responses. This document describes RDAP query extensions that allow clients to specify their preferences for sorting and paging result sets.

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/rfc8977.

Copyright Notice

Copyright (c) 2021 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 Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction

The availability of functionality for result sorting and paging provides benefits to both clients and servers in the implementation of RESTful services [REST]. These benefits include:

- reducing the server response bandwidth requirements;
- improving server response time;
- improving query precision and, consequently, obtaining more relevant results;
- decreasing server query processing load;
- reducing client response processing time.

Approaches to implementing features for result sorting and paging can be grouped into two main categories:

1. sorting and paging are implemented through the introduction of additional parameters in the query string (e.g., the Open Data Protocol (ODATA) [ODATA-PART1]);
2. information related to the number of results and the specific portion of the result set to be returned, in addition to a set of ready-made links for the result set scrolling, are inserted in the HTTP header of the request/response [RFC7231].

However, there are some drawbacks associated with the use of the HTTP header. First, the header properties cannot be set directly from a web browser. Moreover, in an HTTP session, the information on the status (i.e., the session identifier) is usually inserted in the header or a cookie, while the information on the resource identification or the search type is included in the query string. Finally, providing custom information through HTTP headers assumes the client to have a prior knowledge of the server implementation, which is widely considered a Representational State Transfer (REST) design anti-pattern. As a result, this document describes a specification based on the use of query parameters.

Currently, the RDAP protocol [RFC7482] defines two query types:

lookup: the server returns only one object;

search: the server returns a collection of objects.

While the lookup query does not raise issues regarding response size management, the search query can potentially generate a large result set that is often truncated according to server limits. Besides, it is not possible to obtain the total number of objects found that might be returned in a search query response [RFC7483]. Lastly, there is no way to specify sort criteria to return the most relevant objects at the beginning of the result set. Therefore, the client might traverse the whole result set to find the relevant objects or, due to truncation, might not find them at all.

The specification described in this document extends RDAP query capabilities to enable result sorting and paging by adding new query parameters that can be applied to RDAP search path segments. The service is implemented using the Hypertext Transfer Protocol (HTTP) [RFC7230] and the conventions described in [RFC7480].

The implementation of the new parameters is technically feasible, as operators for counting, sorting, and paging rows are currently supported by the major relational database management systems.

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 Query Parameter Specification

The new query parameters are **OPTIONAL** extensions of path segments defined in [RFC7482]. They are as follows:

"count": a boolean value that allows a client to request the return of the total number of objects found;

"sort": a string value that allows a client to request a specific sort order for the result set;

"cursor": a string value representing a pointer to a specific fixed size portion of the result set.

Augmented Backus-Naur Form (ABNF) [RFC5234] is used in the following sections to describe the formal syntax of these new parameters.

2.1. Sorting and Paging 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 and, 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 parameters described in this specification **SHOULD** provide additional information in their responses about both the available sorting criteria and possible pagination. Such information is collected in two **OPTIONAL** response elements named "sorting_metadata" and "paging_metadata".

The "sorting_metadata" element contains the following properties:

"currentSort": "String" (**OPTIONAL**)
: Either the value of "sort" parameter as specified in the query string or the sort applied by default, if any;

"availableSorts": "AvailableSort[]" (**OPTIONAL**)
: An array of objects, with each element describing an available sort criterion. The AvailableSort object includes the following members:

  "property": "String" (**REQUIRED**)
  : The name that can be used by the client to request the sort criterion;

  "default": "Boolean" (**REQUIRED**)
  : Whether the sort criterion is applied by default. An RDAP server **MUST** define only one default sorting property for each object class;

  "jsonPath": "String" (**OPTIONAL**)
  : The JSONPath expression of the RDAP field corresponding to the property;

  "links": "Link[]" (**OPTIONAL**)
  : An array of links as described in [RFC8288] containing the query string that applies the sort criterion.

At least one of the "currentSort" and "availableSorts" properties **MUST** be present.

The "paging_metadata" element contains the following fields:

"totalCount": "Numeric" (**OPTIONAL**)
: A numeric value representing the total number of objects found. It **MUST** be provided if and only if the query string contains the "count" parameter;

"pageSize": "Numeric" (**OPTIONAL**)
: A numeric value representing the number of objects that should have been returned in the current page. It **MUST** be provided if and only if the total number of objects exceeds the page size. This property is redundant for RDAP clients because the page size can be derived from the length of the search results array, but it can be helpful if the end user interacts with the server through a web browser;

"pageNumber": "Numeric" (**OPTIONAL**)
: A numeric value representing the number of the current page in the result set. It **MUST** be provided if and only if the total number of objects found exceeds the page size;

"links": "Link[]" (**OPTIONAL**)
An array of links as described in [RFC8288] containing the reference to the next page. In this specification, only forward pagination is described because it is all that is necessary to traverse the result set.

2.1.1. RDAP Conformance

Servers returning the "paging_metadata" element in their response **MUST** include the string literal "paging" in the rdapConformance array. Servers returning the "sorting_metadata" element **MUST** include the string literal "sorting".

2.2. "count" Parameter

Currently, the RDAP protocol does not allow a client to determine the total number of the results in a query response when the result set is truncated. This is inefficient because the user cannot determine if the result set is complete.

The "count" parameter provides additional functionality that allows a client to request information from the server that specifies the total number of objects matching the search pattern.

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

```
https://example.com/rdap/domains?name=example*.com&count=true
```

The ABNF syntax is the following:

```
   count = "count=" ( trueValue / falseValue )
   trueValue = ("true" / "yes" / "1")
   falseValue = ("false" / "no" / "0")
```

A trueValue means that the server **MUST** provide the total number of the objects in the "totalCount" field of the "paging_metadata" element (Figure 1). A falseValue means that the server **MUST NOT** provide this number.

```
{
  "rdapConformance": [
        "rdap_level_0",
        "paging"
  ],
  ...
  "paging_metadata": {
    "totalCount": 43
  },
  "domainSearchResults": [
    ...
  ]
}
```

Figure 1: Example of RDAP response with "paging_metadata" element containing the "totalCount" field

2.3. "sort" Parameter

The RDAP protocol does not provide any capability to specify the result set sort criteria. A server could implement a default sorting scheme according to the object class, but this feature is not mandatory and might not meet user requirements. Sorting can be addressed by the client, but this solution is rather inefficient. Sorting features provided by the RDAP server could help avoid truncation of relevant results.

The "sort" parameter allows the client to ask the server to sort the results according to the values of one or more properties and according to the sort direction of each property. The ABNF syntax is the following:

```
   sort = "sort=" sortItem *( "," sortItem )
   sortItem = property-ref [":" ( "a" / "d" ) ]
   property-ref = ALPHA *( ALPHA / DIGIT / "_" )
```

"a" means that an ascending sort **MUST** be applied, "d" means that a descending sort **MUST** be applied. If the sort direction is absent, an ascending sort **MUST** be applied.

The following are examples of RDAP queries including the "sort" parameter:

```
https://example.com/rdap/domains?name=example*.com&sort=name

https://example.com/rdap/
domains?name=example*.com&sort=registrationDate:d

https://example.com/rdap/
domains?name=example*.com&sort=lockedDate,name
```

Except for sorting IP addresses and values denoting dates and times, servers **MUST** implement sorting according to the JSON value type of the RDAP field the sorting property refers to. That is, JSON strings **MUST** be sorted lexicographically and JSON numbers **MUST** be sorted numerically. Values denoting dates and times **MUST** be sorted in chronological order. If IP addresses are represented as JSON strings, they **MUST** be sorted based on their numeric conversion.

The conversion of an IPv4 address to a number is possible since each dotted format IPv4 address is a representation of a number written in a 256-based manner; 192.168.0.1 means 1*256^0 + 0*256^1 + 168*256^2 + 192*256^3 = 3232235521. Similarly, an IPv6 address can be converted into a number by applying the base 65536. Therefore, the numerical representation of the IPv6 address 2001:0db8:85a3:0:0:8a2e:0370:7334 is 42540766452641154071740215577757643572. Built-in functions and libraries for converting IP addresses into numbers are available in most known programming languages and relational database management systems.

If the "sort" parameter presents an allowed sorting property, it **MUST** be provided in the "currentSort" field of the "sorting_metadata" element.

2.3.1. Sorting Properties Declaration

In the "sort" parameter ABNF syntax, the element named "property-ref" represents a reference to a property of an RDAP object. Such a reference could be expressed by using a JSONPath expression (named "jsonpath" in the following).

JSONPath is a syntax, originally based on the XML XPath notation [W3C.CR-xpath-31-20161213], which represents a path to select an element (or a set of elements) in a JSON document [RFC8259]. For example, the jsonpath to select the value of the ASCII name inside an RDAP domain lookup response is "$.ldhName", where $ identifies the root of the document object model (DOM). Another way to select a value inside a JSON document is the JSON Pointer [RFC6901].

While JSONPath or JSON Pointer are both commonly adopted notations to select any value inside JSON data, neither are particularly concise and easy to use (e.g., "$.domainSearchResults[*].events[?(@.eventAction='registration')].eventDate" is the jsonpath of the registration date in an RDAP domain search response).

Therefore, this specification defines the "property-ref" element in terms of names identifying RDAP properties. However, not all the RDAP properties are suitable to be used in sort criteria. These properties include:

- properties providing service information (e.g., links, notices, and remarks);
- multivalued properties (e.g., status, roles, and variants);
- properties representing relationships to other objects (e.g., entities).

On the contrary, properties expressed as values of other properties (e.g., registration date) could be used in such a context.

A list of properties an RDAP server **MAY** implement is defined. The properties are divided into two groups: object common properties and object specific properties.

- Object common properties. Object common properties are derived from merging the "eventAction" and the "eventDate" properties. The following values of the "sort" parameter are defined:
  - registrationDate
  - reregistrationDate
  - lastChangedDate
  - expirationDate
  - deletionDate
  - reinstantiationDate
  - transferDate
  - lockedDate
  - unlockedDate
- Object specific properties. Note that some of these properties are also defined as query path segments. These properties include:
  - Domain: name
  - Nameserver: name, ipv4, ipv6.
  - Entity: fn, handle, org, email, voice, country, cc, city

The correspondence between these sorting properties and the RDAP object classes is shown in Table 1. Some of the sorting properties defined for the RDAP entity class are related to jCard elements [RFC7095], but being jCard (the JSON format for vCard [RFC6350]), the corresponding definitions are included in the vCard specification.

An RDAP server **MUST NOT** use the defined sorting properties with a meaning other than the one described in Table 1.

Object class	Sorting property	RDAP property	[RFC7483]	[RFC6350]	[RFC8605]
Searchable objects	Common properties	eventAction values suffixed by "Date"	4.5		
Domain	name	unicodeName/ ldhName	5.3		
Nameserver	name	unicodeName/ ldhName	5.2		
	ipv4	v4 ipAddress	5.2		

Object class	Sorting property	RDAP property	[RFC7483]	[RFC6350]	[RFC8605]
	ipv6	v6 ipAddress	5.2		
Entity	handle	handle	5.1		
	fn	jCard fn	5.1	6.2.1	
	org	jCard org	5.1	6.6.4	
	voice	jCard tel with type="voice"	5.1	6.4.1	
	email	jCard email	5.1	6.4.2	
	country	country name in jCard adr	5.1	6.3.1	
	cc	country code in jCard adr	5.1		3.1
	city	locality in jCard adr	5.1	6.3.1	

Table 1: Sorting properties definition

Regarding the definitions in Table 1, some further considerations are needed to disambiguate some cases:

- Since the response to a search on either domains or nameservers might include both A-labels and U-labels [RFC5890] in general, a consistent sorting policy **MUST** treat the unicodeName and ldhName as two representations of the same value. The unicodeName value **MUST** be used while sorting if it is present; when the unicodeName is unavailable, the value of the ldhName **MUST** be used instead;
- The jCard "sort-as" parameter **MUST** be ignored for the sorting capability described in this document;
- Even if a nameserver can have multiple IPv4 and IPv6 addresses, the most common configuration includes one address for each IP version. Therefore, this specification makes the assumption that nameservers have a single IPv4 and/or IPv6 value. When more than one address per IP version is presented, sorting **MUST** be applied to the first value;
- Multiple events with a given action on an object might be returned. If this occurs, sorting **MUST** be applied to the most recent event;
- Except for handle values, all the sorting properties defined for entity objects can be multivalued according to the definition of vCard as given in [RFC6350]. When more than one value is presented, sorting **MUST** be applied to the preferred value identified by the

parameter pref="1". If the "pref" parameter is missing, sorting **MUST** be applied to the first value.

The "jsonPath" field in the "sorting_metadata" element is used to clarify the RDAP response field the sorting property refers to. The mapping between the sorting properties and the jsonpaths of the RDAP response fields is shown below. The JSONPath operators used herein are described in Appendix A.

- Searchable objects
  - registrationDate

    ```
    $.domainSearchResults[*].events[?(@.eventAction=="registration"
    )].eventDate
    ```

  - reregistrationDate

    ```
    $.domainSearchResults[*].events[?(@.eventAction=="reregistrati
    on")].eventDate
    ```

  - lastChangedDate

    ```
    $.domainSearchResults[*].events[?(@.eventAction=="last
    changed")].eventDate
    ```

  expirationDate
  $.domainSearchResults[*].events[?(@.eventAction=="expiration")].eventDate

  deletionDate
  $.domainSearchResults[*].events[?(@.eventAction=="deletion")].eventDate

  reinstantiationDate
  $.domainSearchResults[*].events[?(@.eventAction=="reinstantiation")].eventDate

  transferDate
  $.domainSearchResults[*].events[?(@.eventAction=="transfer")].eventDate

  lockedDate
  $.domainSearchResults[*].events[?(@.eventAction=="locked")].eventDate

  unlockedDate
  $.domainSearchResults[*].events[?(@.eventAction=="unlocked")].eventDate

- Domain

  name
  $.domainSearchResults[*].[unicodeName,ldhName]

- Nameserver

  name
  : $.nameserverSearchResults[*].[unicodeName,ldhName]

  ipv4
  : $.nameserverSearchResults[*].ipAddresses.v4[0]

  ipv6
  : $.nameserverSearchResults[*].ipAddresses.v6[0]

- Entity

  handle
  : $.entitySearchResults[*].handle

  fn
  : $.entitySearchResults[*].vcardArray[1][?(@[0]=="fn")][3]

  org
  : $.entitySearchResults[*].vcardArray[1][?(@[0]=="org")][3]

  voice
  : $.entitySearchResults[*].vcardArray[1][?(@[0]=="tel" && @[1].type=="voice")][3]

  email
  : $.entitySearchResults[*].vcardArray[1][?(@[0]=="email")][3]

  country
  : $.entitySearchResults[*].vcardArray[1][?(@[0]=="adr")][3][6]

  cc
  : $.entitySearchResults[*].vcardArray[1][?(@[0]=="adr")][1].cc

  city
  : $.entitySearchResults[*].vcardArray[1][?(@[0]=="adr")][3][3]

Additional notes on the provided jsonpaths:

- Those related to the event dates are defined only for the "domain" object. To obtain the equivalent jsonpaths for "entity" and "nameserver", the path segment "domainSearchResults" must be replaced with "entitySearchResults" and "nameserverSearchResults". respectively;
- Those related to jCard elements are specified without taking into account the "pref" parameter. Servers that sort those values identified by the "pref" parameter **SHOULD** update a jsonpath by adding an appropriate filter. For example, if the email values identified by pref="1" are considered for sorting, the jsonpath of the "email" sorting property should be $.entitySearchResults[*].vcardArray[1][?(@[0]=="email" && @[1].pref=="1")][3].

2.3.2. Representing Sorting Links

An RDAP server **MAY** use the "links" array of the "sorting_metadata" element to provide ready-made references [RFC8288] to the available sort criteria (Figure 2). Each link represents a reference to an alternate view of the results.

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

```
{
  "rdapConformance": [
    "rdap_level_0",
    "sorting"
  ],
  ...
  "sorting_metadata": {
     "currentSort": "name",
     "availableSorts": [
       {
       "property": "registrationDate",
       "jsonPath": "$.domainSearchResults[*]
          .events[?(@.eventAction==\"registration\")].eventDate",
       "default": false,
       "links": [
         {
         "value": "https://example.com/rdap/domains?name=example*.com
                   &sort=name",
         "rel": "alternate",
         "href": "https://example.com/rdap/domains?name=example*.com
                  &sort=registrationDate",
         "title": "Result Ascending Sort Link",
         "type": "application/rdap+json"
         },
         {
         "value": "https://example.com/rdap/domains?name=example*.com
                   &sort=name",
         "rel": "alternate",
         "href": "https://example.com/rdap/domains?name=example*.com
                  &sort=registrationDate:d",
         "title": "Result Descending Sort Link",
         "type": "application/rdap+json"
         }
       ]
       },
       ...
     ]
  },
  "domainSearchResults": [
    ...
  ]
}
```

Figure 2: Example of a "sorting_metadata" Instance to Implement Result Sorting

2.4. "cursor" Parameter

The "cursor" parameter defined in this specification can be used to encode information about any pagination method. For example, in the case of a simple implementation of the "cursor" parameter to represent offset pagination information, the cursor value "b2Zmc2V0PTEwMCxsaW1pdD01MA==" is the base64 encoding of "offset=100,limit=50". Likewise, in a simple implementation to represent keyset pagination information, the cursor value "ZXhhbXBsZS1OLmNvbQ==" represents the base64 encoding of "key=example-N.com" whereby the key value identifies the last row of the current page.

Note that this specification uses a base64 encoding for cursor obfuscation just for example. RDAP servers are **NOT RECOMMENDED** to obfuscate a cursor value through a mere base64 encoding.

This solution lets RDAP providers implement a pagination method according to their needs, a user's access level, and the submitted query. Besides, servers can change the method over time without announcing anything to clients. The considerations that have led to this solution are described in more detail in Appendix B.

The ABNF syntax of the "cursor" parameter is the following:

```
   cursor = "cursor=" 1*( ALPHA / DIGIT / "/" / "=" / "-" / "_" )
```

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

```
https://example.com/rdap/domains?name=example*.com
&cursor=wJlCDLIl6KTWypN7T6vc6nWEmEYe99Hjf1XY1xmqV-M=
```

2.4.1. Representing Paging Links

An RDAP server **SHOULD** use the "links" array of the "paging_metadata" element to provide a ready-made reference [RFC8288] to the next page of the result set (Figure 3). Examples of additional "rel" values a server **MAY** implement are "first", "last", and "prev".

```
{
  "rdapConformance": [
    "rdap_level_0",
    "paging"
  ],
  ...
  "notices": [
    {
      "title": "Search query limits",
      "type": "result set truncated due to excessive load",
      "description": [
      "search results for domains are limited to 50"
      ]
    }
  ],
  "paging_metadata": {
    "totalCount": 73,
    "pageSize": 50,
    "pageNumber": 1,
    "links": [
      {
      "value": "https://example.com/rdap/domains?name=example*.com",
      "rel": "next",
      "href": "https://example.com/rdap/domains?name=example*.com
              &cursor=wJlCDLIl6KTWypN7T6vc6nWEmEYe99Hjf1XY1xmqV-M=",
      "title": "Result Pagination Link",
      "type": "application/rdap+json"
      }
    ]
  },
  "domainSearchResults": [
    ...
  ]
}
```

Figure 3: Example of a "paging_metadata" Instance to Implement Cursor Pagination

3. Negative Answers

The constraints for the parameters values are defined by their ABNF syntax. Therefore, each request that includes an invalid value for a parameter **SHOULD** produce an HTTP 400 (Bad Request) response code. The same response **SHOULD** be returned in the following cases:

- if in both single and multi sort the client provides an unsupported value for the "sort" parameter, as well as a value related to an object property not included in the response
- if the client submits an invalid value for the "cursor" parameter

Optionally, the response **MAY** include additional information regarding either the supported sorting properties or the correct cursor values in the HTTP entity body (Figure 4).

```
{
    "errorCode": 400,
    "title": "Domain sorting property 'unknownproperty' is not valid",
    "description": [
        "Supported domain sorting properties are: 'aproperty',
'anotherproperty'."
    ]

}
```

Figure 4: Example of RDAP Error Response Due to an Invalid Domain Sorting Property Included in the Request

4. Implementation Considerations

Implementation of the new parameters is technically feasible, as operators for counting, sorting, and paging are currently supported by the major relational database management systems. Similar operators are completely or partially supported by the most well-known NoSQL databases (e.g., MongoDB, CouchDB, HBase, Cassandra, Hadoop, etc.). Additional implementation notes are included in Appendix C.

5. IANA Considerations

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

Extension identifier: paging
Registry operator: Any
Published specification: This document
Contact: IETF <iesg@ietf.org>
Intended usage: This extension describes best practice for result set paging.

Extension identifier: sorting
Registry operator: Any
Published specification: This document.
Contact: IETF <iesg@ietf.org>
Intended usage: This extension describes best practice for result set sorting.

6. Security Considerations

Security services for the operations specified in this document are described in [RFC7481].

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 either restricting search functionality or limiting the rate of search requests. Servers can also reduce their load by truncating the results in a response. However, this last security policy can result in a higher inefficiency or risk due to acting on incomplete information if the RDAP server does not provide any functionality to return the truncated results.

The new parameters presented in this document provide RDAP operators with a way to implement a server that reduces inefficiency risks. The "count" parameter gives the client the ability to evaluate the completeness of a response. The "sort" parameter allows the client to obtain the most relevant information at the beginning of the result set. This can reduce the number of unnecessary search requests. Finally, the "cursor" parameter enables the user to scroll the result set by submitting a sequence of sustainable queries within server-acceptable limits.

7. References

7.1. 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>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/info/rfc5234>.

[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>.

[RFC6350] Perreault, S., "vCard Format Specification", RFC 6350, DOI 10.17487/RFC6350, August 2011, <https://www.rfc-editor.org/info/rfc6350>.

[RFC7095] Kewisch, P., "jCard: The JSON Format for vCard", RFC 7095, DOI 10.17487/RFC7095, January 2014, <https://www.rfc-editor.org/info/rfc7095>.

[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>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <https://www.rfc-editor.org/info/rfc7231>.

[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>.

[RFC8259] 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/info/rfc8259>.

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

[RFC8605] Hollenbeck, S. and R. Carney, "vCard Format Extensions: ICANN Extensions for the Registration Data Access Protocol (RDAP)", RFC 8605, DOI 10.17487/RFC8605, May 2019, <https://www.rfc-editor.org/info/rfc8605>.

7.2. Informative References

[CURSOR] Nimesh, R., "Paginating Real-Time Data with Cursor Based Pagination", July 2014, <https://www.sitepoint.com/paginating-real-time-data-cursor-based-pagination/>.

[CURSOR-API1] Facebook, "Facebook for Developers -- Using the Graph API", <https://developers.facebook.com/docs/graph-api/using-graph-api>.

[CURSOR-API2] Twitter, "Twitter Ads API", <https://developer.twitter.com/en/docs/twitter-ads-api>.

[GOESSNER-JSON-PATH] Goessner, S., "JSONPath - XPath for JSON", February 2007, <https://goessner.net/articles/JsonPath/>.

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

[JSONPATH-COMPARISON] "JSONPath Comparison", <https://cburgmer.github.io/json-path-comparison/>.

[JSONPATH-WG] IETF, "JSON Path (jsonpath)", <https://datatracker.ietf.org/wg/jsonpath/about/>.

[ODATA-PART1] Pizzo, M., Handl, R., and M. Zurmuehl, "OData Version 4.0. Part 1: Protocol Plus Errata 03", June 2016, <https://docs.oasis-open.org/odata/odata/v4.0/errata03/os/complete/part1-protocol/odata-v4.0-errata03-os-part1-protocol-complete.pdf>.

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

[RFC6901] 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/info/rfc6901>.

[SEEK] EverSQL.com, "Faster Pagination in Mysql - Why Order By With Limit and Offset is Slow?", July 2017, <https://www.eversql.com/faster-pagination-in-mysql-why-order-by-with-limit-and-offset-is-slow/>.

[W3C.CR-xpath-31-20161213] Robie, J., Dyck, M., and J. Spiegel, "XML Path Language (XPath) 3.1", World Wide Web Consortium CR CR-xpath-31-20161213, December 2016, <https://www.w3.org/TR/2016/CR-xpath-31-20161213>.

Appendix A. JSONPath Operators

The jsonpaths used in this document are provided according to the Goessner v.0.8.0 proposal [GOESSNER-JSON-PATH].

Such specification requires that implementations support a set of "basic operators". These operators are used to access the elements of a JSON structure like objects and arrays as well as their subelements, object members and array items, respectively. No operations are defined for retrieving parent or sibling elements of a given element. The root element is always referred to as $ regardless of it being an object or array.

Additionally, the specification permits implementations to support arbitrary script expressions. These can be used to index into an object or array, or to filter elements from an array. While script expression behavior is implementation defined, most implementations support the basic relational and logical operators as well as both object member and array item access, sufficiently similar for the purpose of this document. Commonly supported operators/functions divided into "top-level operators" and "filter operators" are documented in Table 2 and Table 3, respectively.

For more information on implementation interoperability issues, see [JSONPATH-COMPARISON]. At the time of writing, work is beginning on a standardization effort too (see [JSONPATH-WG]).

Operator	Description
$	Root element
.<name>	Object member access (dot-notation)
['<name>']	Object member access (bracket-notation)
[<number>]	Array item access
*	All elements within the specified scope

Operator	Description
[?(<expression>)]	Filter expression

Table 2: JSONPath Top-Level Operators

Operator	Description
@	Current element being processed
.<name>	Object member access
.[<name1>,<name2>]	Union of object members
[<number>]	Array item access
==	Left is equal to right
!=	Left is not equal to right
<	Left is less than right
<=	Left is less than or equal to right
>	Left is greater than right
>=	Left is greater than or equal to right
&&	Logical conjunction
||	Logical disjunction

Table 3: JSONPath Filter Operators

Appendix B. Approaches to Result Pagination

An RDAP query could return a response with hundreds, even thousands, of objects, especially when partial matching is used. For this reason, the "cursor" parameter addressing result pagination is defined to make responses easier to handle.

Presently, the most popular methods to implement pagination in a REST API include offset pagination and keyset pagination. Neither pagination method requires the server to handle the result set in a storage area across multiple requests since a new result set is generated each time a request is submitted. Therefore, they are preferred to any other method requiring the management of a REST session.

Using limit and offset operators represents the traditionally used method to implement result pagination. Both of them can be used individually:

"limit=N": means that the server returns the first N objects of the result set;

"offset=N": means that the server skips the first N objects and returns objects starting from position N+1.

When limit and offset are used together, they provide the ability to identify a specific portion of the result set. For example, the pair "offset=100,limit=50" returns the first 50 objects starting from position 101 of the result set.

Though easy to implement, offset pagination also includes drawbacks:

- when offset has a very high value, scrolling the result set could take some time;
- it always requires fetching all rows before dropping as many rows as specified by offset;
- it may return inconsistent pages when data is frequently updated (i.e., real-time data).

Keyset pagination [SEEK] adds a query condition that enables the selection of the only data not yet returned. This method has been taken as the basis for the implementation of a "cursor" parameter [CURSOR] by some REST API providers [CURSOR-API1] [CURSOR-API2]. The cursor is an opaque to client URL-safe string representing a logical pointer to the first result of the next page.

Nevertheless, even keyset pagination can be troublesome for the following reasons:

- It needs at least one key field.
- It does not allow sorting simply by any field because the sorting criterion must contain a key.
- It works best with full composite value support by database management systems (i.e., [x,y]>[a,b]); emulation is possible but inelegant and less efficient.
- It does not allow direct navigation to arbitrary pages because the result set must be scrolled in sequential order starting from the initial page.
- Implementing bidirectional navigation is tedious because all comparison and sort operations have to be reversed.

B.1. Specific Issues Raised by RDAP

Some additional considerations can be made in the RDAP context:

- An RDAP object is a conceptual aggregation of information generally collected from more than one data structure (e.g., table), and this makes it even harder to implement keyset pagination, a task that is already quite difficult. For example, the entity object can include information from different data structures (registrars, registrants, contacts, resellers), each one with its key field mapping the RDAP entity handle.

- Depending on the number of page results as well as the number and the complexity of the properties of each RDAP object in the response, the time required by offset pagination to skip the previous pages could be much faster than the processing time needed to build the current page. In fact, RDAP objects are usually formed by information belonging to multiple data structures and containing multivalued properties (i.e., arrays); therefore, data selection might be a time-consuming process. This situation occurs even though the selection is supported by indexes.
- Depending on the access levels defined by each RDAP operator, the increase in complexity and the decrease in flexibility of keyset pagination in comparison to offset pagination could be considered impractical.

Ultimately, both pagination methods have benefits and drawbacks.

Appendix C. Additional Implementation Notes

This section contains an overview of the main choices made during the implementation of the capabilities defined above in the RDAP public test server of Registro.it at the Institute of Informatics and Telematics of the National Research Council (IIT-CNR). The content of this section can represent a guidance for those implementers who plan to provide RDAP users with those capabilities. The RDAP public test server can be accessed at <https://rdap.pubtest.nic.it/>. Further documentation about the server features is available at <https://rdap.pubtest.nic.it/doc/README.html>.

C.1. Sorting

If no sort criterion is specified in the query string, the results are sorted by a default property: "name" for domains and nameservers, and "handle" for entities. The server supports multiple property sorting but the "sorting_metadata" object includes only the links to alternative result set views sorted by a single property just to show the list of sorting properties allowed for each searchable object. The server supports all the object-specific sorting properties described in the specification except for nameserver sorting based on unicodeName, that is, the "name" sorting property is mapped onto the "ldhName" response field. Regarding the object common properties, the sorting by registrationDate, expirationDate, lastChangedDate, and transferDate is supported.

C.2. Counting

The counting operation is implemented through a separate query. Some relational database management systems support custom operators to get the total count together with the rows, but the resulting query can be considerably more expensive than that performed without the total count. Therefore, as "totalCount" is an optional response information, fetching always the total number of rows has been considered an inefficient solution. Furthermore, to avoid the processing of unnecessary queries, when the "count" parameter is included in the submitted query, it is not also repeated in the query strings of the "links" array provided in both "paging_metadata" and "sorting_metadata" objects.

C.3. Paging

The server implements the cursor pagination through the keyset pagination when sorting by a unique property is requested or the default sort is applied, through offset pagination otherwise. As most of the relational database management systems don't support the comparison of full composite values natively, the implementation of full keyset pagination seem to be troublesome so, at least initially, a selective applicability of keyset pagination is advisable. Moreover, the "cursor" value encodes not only information about pagination but also about the search pattern and the other query parameters in order to check the consistency of the entire query string. If the "cursor" value is inconsistent with the rest of the query string, the server returns an error response.

Acknowledgements

The authors would like to acknowledge Brian Mountford, Tom Harrison, Karl Heinz Wolf, Jasdip Singh, Erik Kline, Éric Vyncke, Benjamin Kaduk, and Roman Danyliw for their contribution to the development of 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

Scott Hollenbeck
Verisign Labs
12061 Bluemont Way
Reston, VA 20190
United States of America
Email: shollenbeck@verisign.com
URI: https://www.verisignlabs.com/