Registration Protocols Extensions (regext)
Internet-Draft
Intended status: Standards Track
Expires: 3 January 2026

A. Newton
ICANN
J. Singh
ARIN

2 July 2025

Extensions Parameter for the RDAP Media Type
draft-ietf-regext-rdap-x-media-type-04

Abstract

This document defines a new parameter for the RDAP media type that can be used to describe RDAP content with RDAP extensions. Additionally, this document describes the usage of this parameter with RDAP for the purposes of signalling RDAP extensions during content negotiation.

Status of This Memo

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

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

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

This Internet-Draft will expire on 3 January 2026.

Table of Contents

1. Background 2
1.1. Document Terms 3
2. The RDAP Media Type With Extensions Parameter 3
3. Using The Extensions Parameter 3
3.1. Extension Identifier 5
3.2. Examples 5
3.2.1. Classic Negotiation 5
3.2.2. Negotiation of an RDAP Extension 6
3.2.3. Negotiation of an RDAP Extension Without "Content-Type" 6
3.2.4. No Server Support for exts_list Parameter 7
3.2.5. Differing Extension Negotiation 7
3.2.6. Extension Versioning and Meta-data 7
4. Usage in RDAP Links 8
5. Security Considerations 9
6. IANA Considerations 9
6.1. RDAP Extension Registry 9
6.2. Addition of Parameter to RDAP Media Type 9
7. Acknowledgements 10
8. References 10
8.1. Normative References 10
8.2. Informative References 10
Appendix A. Using the Vary Header 11
Appendix B. Design Considerations 12
B.1. Reusing the Existing Media Type 12
B.2. Inappropriate Use of Query Parameters 13
B.2.1. Copy and Paste 13
B.2.2. Redirects 13
B.2.3. Referral Compatibility 14
B.2.4. Architectural Violations 14
Authors' Addresses 15

1. Background

[RFC7480] defines the "application/rdap+json" media type to be used with RDAP. This document defines a new parameter for this media type when an RDAP extension needs to be described during HTTP content negotiation.

This parameter enables an RDAP client to signal to an RDAP server the list of RDAP extensions supported by that client. For example, an RDAP client that supports the "foo" extension may use this mechanism as a signal to an RDAP server, thus allowing the server to respond with data using the "foo" extension inside an RDAP response only when it can be assured the client can understand the extension.

By using this method, there is no need for every RDAP extension to define their own unique signaling mechanism. Additionally, this method is designed to be backwards-compatible with the deployed RDAP ecosystem (see Appendix B for further information).

1.1. Document Terms

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. The RDAP Media Type With Extensions Parameter

The RDAP media type, "application/rdap+json", may have an optional parameter named "exts_list". This parameter is a whitespace-separated list of RDAP extension identifiers (as would be found in the "rdapConformance" array).

Here is an example:

```
application/rdap+json;exts_list="rdap_level_0 exts fred"
```

3. Using The Extensions Parameter

[RFC7480] specifies the usage of "application/json", "application/rdap+json" or both with HTTP "accept" header. The "exts_list" parameter may only be used with the "application/rdap+json" media type.

This is an example of the "accept" header using the RDAP media type with an "extensions" parameter:

```
accept: application/json;q=0.9,
    application/rdap+json;exts_list="rdap_level_0 exts fred";q=1
```

If both a client and a server support the "exts_list" parameter, and the client requests an extension that is unimplemented by the server, the server MUST respond with only extensions included in the response by the server. This behavior is backwards-compatible as RDAP clients must ignore unknown RDAP extensions as specified by [RFC9083]. Responding with an HTTP 406 Not Acceptable status code is NOT RECOMMENDED because an RDAP client could interpret this status code to mean that the server does not understand RDAP in its entirety.

Likewise, if a server is required to use an extension in a response that was not requested by the client, the server MUST respond as if the client had requested the extension. This behavior is backwards-compatible as RDAP clients must ignore unknown extensions as specified by [RFC9083]. Responding with an HTTP 406 Not Acceptable status code is NOT RECOMMENDED for the reason stated above.

When the "exts_list" parameter is used in the RDAP media type in the "content-type" header, the values in the media type's "exts_list" parameter MUST match the values in the "rdapConformance" array in the returned JSON. However, implementation experience has shown that some HTTP server libraries do not support modification of the "content-type" header per query type. Therefore use of the "exts_list" parameter with the media type of the "content-type" header is NOT REQUIRED. That is, when used in the "content-type" header, the values of the "exts_list" parameter must match that of the "rdapConformance" array but server may opt to omit the "exts_list" parameter from the media type in the "content-type" header.

The contents of the "exts_list" parameter mirrors the content of the "rdapConformance" array in server responses. This includes the identifier "rdap_level_0", which is not an extension identifier but an identifier for the base RDAP specifications. Servers MUST follow the same rules for placing "rdap_level_0" in the content of the "exts_list" parameter and the "rdapConformance" array. Clients MUST interpret an "exts_list" parameter without "rdap_level_0" or one of its successor identifiers (e.g. "rdap_level_1") in the same manner as the interpretation of the "rdapConformance" array without "rdap_level_0" or one of its successors.

Nothing in this specification sidesteps or obviates the HTTP content negotiation defined in [RFC9110] for RDAP.

Likewise, nothing in this specification sidesteps or obviates the HTTP caching mechanisms defined in [RFC9110]. Further advice on the "vary" header can be found in Appendix A.

Some RDAP extensions, such as [RFC9560], have other protocol elements (e.g. extension-specific query parameters) passed from the client to the server, and the presence of these protocol elements may be used by servers to determine a client's capability to handle the related RDAP extension(s). This specification does not require the usage of those extension identifiers in the "exts_list" parameter, though clients SHOULD list the extension identifier in the "exts_list" parameter when using other protocol elements of those extensions for better compatibility with servers recognizing the "exts_list" parameter. Servers SHOULD NOT require the usage of extension identifiers in the "exts_list" parameter when other extension protocol elements are used for backwards-compatibility purposes.

3.1. Extension Identifier

This document defines an RDAP extension using the identifier "exts". This RDAP extension defines no additional RDAP queries or response structures.

The purpose of this RDAP extension is to allow servers to signal support for the "exts_list" parameter in "rdapConformance" arrays of responses to "/help" (aka "service discovery").

3.2. Examples

The following examples use the HTTP/1.1 message exchange syntax as seen in [RFC9110].

3.2.1. Classic Negotiation

This example demonstrates the negotiation of the "application/rdap+json" media type as defined in [RFC7480] using an RDAP "/help" query. This example also demonstrates the negotiation in which a client does not support the "exts_list" parameter but a server does support the "exts_list" parameter.

Client Request:

```
GET /help HTTP/1.1
accept: application/rdap+json
```

Server Response:

```
HTTP/1.1 200 OK
content-type: application/rdap+json;exts_list="rdap_level_0 exts"

{ "rdapConformance" : [ "rdap_level_0", "exts" ],
  "notices" : [
    { "description" : [ "my content includes a trailing CRLF" ] } ] }
```

3.2.2. Negotiation of an RDAP Extension

In this example, both the client and server support the "exts_list" parameter and a fictional extension of "foo".

Client Request:

```
GET /help HTTP/1.1
accept: application/rdap+json;exts_list="rdap_level_0 exts foo"
```

Server Response:

```
HTTP/1.1 200 OK
content-type: application/rdap+json;exts_list="rdap_level_0 exts foo"

{ "rdapConformance" : [ "rdap_level_0", "exts", "foo" ],
  "notices" : [
    { "description" : [ "my content includes a trailing CRLF" ] } ] }
```

3.2.3. Negotiation of an RDAP Extension Without "Content-Type"

In this example, both the client and server support the "exts_list" parameter and a fictional extension of "foo". However, the server does not support the "exts_list" parameter in the "content-type" header.

Client Request:

```
GET /help HTTP/1.1
accept: application/rdap+json;exts_list="rdap_level_0 exts foo"
```

Server Response:

```
HTTP/1.1 200 OK
content-type: application/rdap+json

{ "rdapConformance" : [ "rdap_level_0", "exts", "foo" ],
  "notices" : [
    { "description" : [ "my content includes a trailing CRLF" ] } ] }
```

3.2.4. No Server Support for exts_list Parameter

In this example, only the client supports the "exts_list" parameter, along with a fictional extension of "foo" by both.

Client Request:

```
GET /help HTTP/1.1
accept: application/rdap+json;exts_list="rdap_level_0 exts foo"
```

Server Response:

```
HTTP/1.1 200 OK
content-type: application/rdap+json

{ "rdapConformance" : [ "rdap_level_0", "foo" ],
  "notices" : [
    { "description" : [ "my content includes a trailing CRLF" ] } ] }
```

3.2.5. Differing Extension Negotiation

In this example, both the client and server support the "exts_list" parameter. The client supports the extensions "foo" and "bar" while the server only support "foo".

Client Request:

```
GET /help HTTP/1.1
accept: application/rdap+json;exts_list="rdap_level_0 exts foo bar"
```

Server Response:

```
HTTP/1.1 200 OK
content-type: application/rdap+json;exts_list="rdap_level_0 exts foo"

{ "rdapConformance" : [ "rdap_level_0", "exts", "foo" ],
  "notices" : [
    { "description" : [ "my content includes a trailing CRLF" ] } ] }
```

3.2.6. Extension Versioning and Meta-data

For scenarios where the "versioning" extension, as defined by [I-D.ietf-regext-rdap-versioning], is used, the extension identifiers in the client request may not be exact or case-insensitive matches for the extension identifiers in the server response (unlike scenarios where the "versioning" extension is not used). That is, the extension identifiers used by the client have appended versioning information, but the extension identifiers returned by the server do

not have appended versioning information (such information is in the "versioning" JSON).

Client Request:

```
GET /domain/example.com HTTP/1.1
accept: application/rdap+json;exts_list="rdap_level_0 exts versioning_0_2"
```

Server Response:

```
HTTP/1.1 200 OK
content-type: application/rdap+json;exts_list="rdap_level_0 exts versioning"

{ "rdapConformance" : [ "rdap_level_0", "exts", "versioning" ],
  "objectClassName": "domain",
  "ldhName": "example.com",
  "versioning": [ {
    "extension": "versioning",
    "type": "semantic",
    "version": "versioning_0_2" } ]
}
```

Servers might also use the "versioning" extension to describe meta-data about supported extensions even if the servers do not explicitly support extension versioning.

4. Usage in RDAP Links

Section 4.2 of [RFC9083] defines a link structure used in RDAP.

```
{
  "value": "https://example.com/context_uri",
  "rel": "self",
  "href": "https://example.com/target_uri",
  "hreflang": [ "en", "ch" ],
  "title": "title",
  "media": "screen",
  "type": "application/json"
}
```

The type attribute signals to a client the expected media type of the resource referenced in the href attribute, and some clients use this information to determine if the URI in the href attribute should be de-referenced.

Usage of the "exts_list" parameter in the media type of the "type" attribute is allowed but the "type" attribute as a whole is only a hint, as noted by [RFC8288]:

| The "type" attribute, when present, is a hint indicating what the
| media type of the result of dereferencing the link should be.
| Note that this is only a hint; for example, it does not override
| the Content-Type header field of a HTTP response obtained by
| actually following the link.

5. Security Considerations

As stated in Section 3, this specification does not override the protocol elements of RDAP security extensions, such as [RFC9560], nor does it override the protocol elements of other security features of HTTP.

This specification does contrast with solutions using query parameters in that those solutions require servers to blindly copy query parameters into redirect URLs in situations where such copying could cause harm, such as copying an API key intended for one server into the redirect URL of another server.

6. IANA Considerations

6.1. RDAP Extension Registry

The IETF requests the IANA to register the following extension in the RDAP Extensions Registry at [RDAP-EXTENSIONS]:

Extension identifier: exts

Registry operator: ALL

Published specification: [RFC Reference Once Published]

Person & email address to contact for further information:
The Internet Engineering Steering Group <iesg@ietf.org>

Intended usage: COMMON

6.2. Addition of Parameter to RDAP Media Type

This document defines the optional parameter "exts_list" for the media type "application/rdap+json" as described in Section 2.

The IETF requests the IANA to add this document as an additional reference to the IANA Media Type registry at [MEDIA-TYPES] for the media type "application/rdap+json".

7. Acknowledgements

Pawel Kowalik provided extensive review of this document and conducted a study that forms the basis of re-using the existing RDAP media type. Mario Loffredo and James Mitchell have provided ideas and feedbacks that have contributed to the content of this document based on their implementation experience. Murray Kucherawy and Alexey Melnikov provided guidance on the use of media types and media type parameters. Jame Gould provided feedback that contributed to the content of this document.

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>.

[RFC7480] Newton, A., Ellacott, B., and N. Kong, "HTTP Usage in the Registration Data Access Protocol (RDAP)", STD 95, RFC 7480, DOI 10.17487/RFC7480, March 2015, <https://www.rfc-editor.org/info/rfc7480>.

[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

[I-D.ietf-regext-rdap-extensions] Newton, A., Singh, J., and T. Harrison, "RDAP Extensions", Work in Progress, Internet-Draft, draft-ietf-regext-rdap-extensions-06, 29 April 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-regext-rdap-extensions-06>.

[I-D.ietf-regext-rdap-versioning]
Gould, J., Keathley, D., and M. Loffredo, "Versioning in the Registration Data Access Protocol (RDAP)", Work in Progress, Internet-Draft, draft-ietf-regext-rdap-versioning-03, 23 June 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-regext-rdap-versioning-03>.

[MEDIA-TYPES]
IANA, "Media Types", <https://www.iana.org/assignments/media-types/>.

[RDAP-EXTENSIONS]
IANA, "RDAP Extensions", <https://www.iana.org/assignments/rdap-extensions/>.

[RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, DOI 10.17487/RFC2045, November 1996, <https://www.rfc-editor.org/info/rfc2045>.

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

[RFC9083] Hollenbeck, S. and A. Newton, "JSON Responses for the Registration Data Access Protocol (RDAP)", STD 95, RFC 9083, DOI 10.17487/RFC9083, June 2021, <https://www.rfc-editor.org/info/rfc9083>.

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

[RFC9560] Hollenbeck, S., "Federated Authentication for the Registration Data Access Protocol (RDAP) Using OpenID Connect", RFC 9560, DOI 10.17487/RFC9560, April 2024, <https://www.rfc-editor.org/info/rfc9560>.

Appendix A. Using the Vary Header

Server implementers may want to consider using the "vary" header depending on the caching behavior desired of shared caches (i.e. middleboxes, not client caches).

Consider the following scenario where user Bob and user Alice send queries to the same RDAP server that is routed through a middlebox network element implementing a shared HTTP cache.

User Bob sends a query for the domain "example.com" (http://regy.example/domain/example.com (http://regy.example/domain/example.com)) without the "exts_list" parameter. The "accept" header sent for Bob's query would be accept: application/rdap+json or accept: application/json.

User Alice later sends a query for the same domain, however her client uses the "exts_list" parameter. The "accept" header sent for Alice's query might be accept: application/rdap+json;exts_list="rdap_level_0 exts foo".

If no "vary" header is set in the response for these queries, the shared cache will compare only the URL of the query when processing cache items and therefore user Bob and user Alice would receive the same answer. In other words, since both queried "http://regy.example/domain/example.com" (http://regy.example/domain/example.com") the shared cache would return the answer of the first query to the second query and all other subsequent queries until the item expired out of the cache.

If server implementers do not desire this behavior and would signal that caches consider each query separately, servers should also return a "vary: accept" header to inform the cache that the "accept" header should also be considered when processing cache items. Server implementers should also consult [RFC9110] regarding caching and other uses of the "vary" header.

Appendix B. Design Considerations

B.1. Reusing the Existing Media Type

Earliest versions of this document specified a new media type because the authors believed the addition of new parameter on the existing RDAP media type may be backwards-incompatible with many RDAP servers. However, a study conducted by Pawel Kowalik concluded that 99.65% of RDAP servers are compatible with a new parameter on the existing RDAP media type.

Additionally, [RFC2045] requires that the server ignore unknown parameters.

B.2. Inappropriate Use of Query Parameters

Another design approach to communicating RDAP extensions from the client to the server would be the use of URI query parameters:

https://rdap.example/domain/foo.example?extensions=fizzbuzz

However, there are a few problems with using query parameters for this scenario. Some of these problems are specific to RDAP and are also documented in [I-D.ietf-regext-rdap-extensions]. The following sections also describe the problems.

B.2.1. Copy and Paste

Consider two RDAP users, Alice and Bob. Alice has an RDAP client that supports the extension "fizzbuzz", and Bob has an RDAP client that does not support this extension.

Now consider the scenario where Alice copies and pastes the RDAP URL from above into an email and sends it to Bob. When Bob uses that URL with his RDAP client, it will be communicating to the server that the extension "fizzbuzz" is understood by Bob’s client when it is not.

In this scenario, Bob’s client will be unable to render the RDAP extension regardless of the usage or not of the query parameter. However, if the server is using the query parameter for secondary purposes, such as gathering metrics and statistics, then the capabilities of Bob’s client will have been incorrectly signalled to the server.

B.2.2. Redirects

The RDAP ecosystem uses redirects in many situations. [RFC7480] discusses "aggregators", which are RDAP servers used to help clients find authoritative RDAP servers using the RDAP bootstrap registries. Redirects are also heavily used by the RIRs when IP addresses or autonomous system numbers are transferred from one RIR to another.

Within HTTP, URI query parameters are not explicitly preserved during a redirect (probably due to architecture considerations, see the section below). Specific to RDAP, [RFC7480] instructs RDAP servers to ignore unknown query parameters and instructs clients not to transform the URL of a redirect.

Therefore, query parameters denoting RDAP extensions should not survive redirects in RDAP. This can be readily observed in currently deployed RDAP servers:

```
curl -v https://rdap-bootstrap.arin.net/bootstrap/autnum/2830?extension=fizzbuzz
```

To further demonstrate that query parameters do not automatically survive redirects but that media types do, consider the code found here (https://github.com/anewton1998/draft-regext-ext-json-media-type). This code consists of a simple client and a simple server. The client sets both a new media type and query parameters. The servers listen on two ports, redirecting the client from a URL on the first port to a URL on the second port.

Preservation of query parameters is not a guaranteed feature of HTTP client and server libraries, whereas preservation of media types is much more likely to occur.

B.2.3. Referral Compatibility

It is common in the RDAP ecosystem to link from one RDAP resource to another. These are typically conveyed in the link structure defined in Section 4.2 of [RFC9083] and use the "application/rdap+json" media type. One common usage is to link to a domain registration in a domain registrar from a domain registration in a domain registry.

```
{
  "value" : "https://regy.example/domain/foo.example",
  "rel" : "related",
  "href" : "https://regr.example/domain/foo.example",
  "type" : "application/rdap+json"
}
```

Usage of the "exts_list" parameter does not require clients to conduct further processing of these referrals, whereas a query parameter approach would require clients to process and de-conflict any other query parameters if present.

B.2.4. Architectural Violations

Section 3.4 of [RFC3986] states the following:

> The query component contains non-hierarchical data that, along with data in the path component (Section 3.3), serves to identify a resource within the scope of the URI’s scheme and naming authority (if any).

Therefore, URI query parameters are meant to be part of the identity of the resource being identified by a URI and pointed to by the location of a URL. RDAP extensions change the portions of JSON returned by the server but are not intended to change the resource being identified. That is, a domain registration is the same domain

registration regardless of whether the postal address in that domain registration is communicated via JCard or a new RDAP extension for JSContact.

Changing how the content of a resource is conveyed is called content negotiation and is discussed in detail in [RFC9110] using media types.

Readers should note that protocol design is not a "priestly affair" in which architectural violations are strictly forbidden. Every design decision is a trade-off. However, following the architecture of an ecosystem generally makes re-use of software and systems easier, and often eases the adoption of newer features in the future. When given the choice between two designs, the design that does not violate architecture should be preferred when all other considerations are equal.

Authors' Addresses

Andy Newton
ICANN
Email: andy@hxr.us

Jasdip Singh
ARIN
Email: jasdips@arin.net