Registration Protocols Extensions (regext)
Internet-Draft
Updates: 7480, 9082, 9083 (if approved)
Intended status: Standards Track
Expires: 7 June 2026

A. Newton
ICANN
J. Singh
ARIN
T. Harrison
APNIC

4 December 2025

RDAP Extensions
draft-ietf-regext-rdap-extensions-09

Abstract

This document describes and clarifies the usage of extensions in RDAP.

Status of This Memo

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

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

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

This Internet-Draft will expire on 7 June 2026.

Table of Contents

1. Background

The Registration Data Access Protocol (RDAP) defines a uniform protocol for accessing data from Internet operations registries, specifically Domain Name Registries (DNRs), Regional Internet Registries (RIRs), and other registries in the Internet Number Registry System (INRS). RDAP queries are defined in [RFC9082] and RDAP responses are defined in [RFC9083].

RDAP contains a means to define extensions for queries not found in [RFC9082] and responses not found in [RFC9083]. RDAP extensions are also described in [RFC7480]. This document describes the requirements for RDAP extension definition and use, clarifying ambiguities and defining additional semantics and options that were previously implicit or under-specified, and places some constraints on the definition of RDAP extensions to prevent collisions with various extension mechanisms.

1.1. Summary of Updates

This document updates [RFC7480], [RFC9082], and [RFC9083] for the purposes of constraining how extensions are defined. This document does not update any core RDAP requests or responses nor does it update or obsolete any existing RDAP extensions. The updates in this document should require no changes to either client or server implementations.

This document describes the following methods for extending RDAP by registered extensions:

1. JSON Names - The most common extension point for RDAP is the definition of new JSON Names. Guidance for JSON Names is provided in this document with regard to [RFC7480] and [RFC9083].
2. Query Paths - New lookups and searches are defined using URL paths. This document clarifies the practice as described in [RFC9082].
3. Query Parameters - Many queries use URL query parameters to scope and/or enhance RDAP results. This document clarifies the practice as described in [RFC9082].
4. HTTP Headers - Some extensions may use HTTP headers or header parameters not explicitly enumerated by [RFC7480].
5. Object Classes - Extensions may define new types of objects to be queried. This document clarifies this method as described in [RFC9082] and [RFC9083].

Additionally, this document updates the IANA registry practices for RDAP. See Section 7.

1.2. 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. Extension Identifiers

2.1. Purpose

Section 6 of [RFC7480] describes the identifier used to signify RDAP extensions and the IANA registry into which RDAP extensions are to be registered.

When in use in RDAP, extension identifiers are prepended to URL path segments, URL query parameters, and JSON object member names. They are also included in the "rdapConformance" array member of each response that relies on the extension, so that clients can determine the extensions being used by the server for that response. The "/help" query returns a response with an "rdapConformance" member containing the identifiers for all extensions used by the server.

The main purpose of the extension identifier is to act as a namespace, preventing collisions between elements from different extensions. Additionally, implementers and operators can use the extension identifiers to find extension definitions via an IANA registry.

2.1.1. Profile Extensions

While the RDAP extension mechanism was created to extend RDAP queries and/or responses, extensions can also be used to signal server policy (for example, specifying the conditions of use for existing response structures). Extensions that are primarily about signaling server policy are called "profiles". Profile extensions are often used by a class of RDAP server operators, such as the [icann-profile] used by gTLD registries and registrars and the [nro-profile] used by RIRs.

Profile extensions may do the following:

* Mark some specific extensions (and versions thereof) as required.
* Mark some specific optional queries, object classes, or JSON structures as required.

* Limit or restrict the values of specific JSON structures.

Some profile extensions exist to denote the usage of values placed into an IANA registry, such as the IANA RDAP registries, or the usage of extensions for specifications used in RDAP responses, such as extended vCard/jCard properties.

For example, an extension may be used to signal desired processing of a "rel" attribute in a "links" array, where the "rel" value is registered in the [link-relations]:

```
{
  "rdapConformance": [
    "rdap_level_0",
    "lunarNIC"
  ],
  "objectClassName": "domain",
  "ldhName": "example.com",
  "links": [
    {
      "value": "https://example.com/domain/example.com",
      "href": "https://example.com/sideways_href",
      "rel": "sideways",
      "type": "application/rdap+json"
    }
  ]
}
```

When defining the usage of link relations, extensions MUST specify the media types expected to be used with those link relations.

Profile extensions may also leverage the appearance of their identifier in the "rdapConformance" array (i.e., clients are signaled that a profile is in use). Profile extensions that mandate the implementation of another extension MUST require that the implementer include the extension identifier for that other extension in the "rdapConformance" array.

[RFC7480] mandates the implementation of HTTPS but does not mandate its use. Some profile extensions, especially those used by classes of server operators, specify the required use of HTTPS and disallow the use of unencrypted HTTP. Similarly, some profile extensions specify the availability of service over IPv6.

As described above, these characteristics are not exclusive to profile extensions and may be found in extensions defining new queries, JSON, and other RDAP extension points (see Section 1.1).

2.1.2. Multiple Identifiers in Single Extension

Extension specifications MAY define more than one extension identifier. The servers MUST list all extension identifiers used to generate a response in the "rdapConformance" array. The server MUST list all supported extension identifiers in the "rdapConformance" array of a response to a "/help" request.

2.2. Syntax

In brief, RDAP extension identifiers start with an alphabetic character and may contain alphanumeric characters and "_" (underscore) characters. This formulation was explicitly chosen to allow compatibility with variable names in programming languages and transliteration with XML. See Section 6 and Section 2.1.

RDAP extension identifiers have no explicit structure, and are opaque insofar as no inner-meaning can be "seen" in them.

RDAP extensions MUST NOT define an extension identifier that may collide with an existing extension identifier. For example, if there were a pre-existing identifier of "foo_bar", another extension could not define the identifier "foo". Likewise, if there were a pre-existing identifier of "foo_bar", another extension could not define the identifier "foo_bar_buzz". However, an extension could define "foo" if there were a pre-existing definition of "foobar", and vice versa.

For this reason, this document updates the guidance of [RFC7480] regarding underscore characters: RDAP extensions MUST NOT use an underscore character in their RDAP extension identifier. Implementers should be aware that many existing extension identifiers do contain underscore characters.

[RFC7480] does not explicitly state that extension identifiers are case-sensitive. This document clarifies the formulation in [RFC7480] to explicitly note that extension identifiers are case-sensitive, and extension identifiers MUST NOT be registered where a new identifier is a mixed-case version of an existing identifier (see Section 7.1). For example, given "lunarNIC" is already registered as an identifier, then a new registration with "lunarNic" (note the lowercase "ic" in "Nic") would not be allowed.

2.3. Bare Extension Identifiers

Section 2.1 of [RFC9083] states the following when using the names of JSON members:

> | Clients of these JSON responses SHOULD ignore unrecognized JSON
> | members in responses. Servers can insert members into the JSON
> | responses, which are not specified in this document, but that does
> | not constitute an error in the response. Servers that insert such
> | unspecified members into JSON responses SHOULD have member names
> | prefixed with a short identifier followed by an underscore
> | followed by a meaningful name. It has been observed that these
> | short identifiers aid software implementers with identifying the
> | specification of the JSON member, and failure to use one could
> | cause an implementer to assume the server is erroneously using a
> | name from this specification. This allowance does not apply to
> | jCard [RFC7095] objects. The full JSON name (the prefix plus the
> | underscore plus the meaningful name) SHOULD adhere to the
> | character and name limitations of the prefix registry described in
> | [RFC7480]. Failure to use these limitations could result in
> | slower adoption as these limitations have been observed to aid
> | some client programming models.

Despite this, some RDAP extensions define only one JSON value and do not prefix it with their RDAP extension identifier, instead using the extension identifier as the JSON name for that JSON value. That is, the extension identifier is used "bare" and not appended with an underscore character and subsequent names.

Consider the example in Section 2.5.2. Using the bare extension identifier pattern, that example could be written as:

```
{
  "rdapConformance": [
    "rdap_level_0",
    "lunarNIC"
  ],
  "objectClassName": "domain",
  "ldhName": "example.com",
  "remarks":
  [
    {
      "description":
      [
        "She sells sea shells down by the sea shore.",
        "Originally written by Terry Sullivan."
      ]
    }
  ],
  "lunarNIC":
  {
    "firstInitial": "R",
    "lastName": "Heinlein"
  }
}
```

While [RFC9083] is specific to JSON, the use of a bare extension identifier also applies to other identifiers of RDAP extensions, such as query parameters and object class names. Identifiers of an RDAP extension which need a prefix to avoid name collision with identifiers of other RDAP extensions or RDAP as specified in [RFC7480], [RFC9082], and [RFC9083] are referred to as namespaced identifiers.

Usage of a bare extension identifier conflicts with the guidance in Section 2.1 of [RFC9083]. Previously, extension authors have used this pattern when only one query path, JSON name, and/or object class is being defined by the extension.

Implementation experience has shown that an extension using a bare identifier can be interoperable, though more difficult to process and parse in some instances. Furthermore, prefixed identifiers are clearly syntactically distinguishable from identifiers defined by the core RDAP specifications, which provides more flexibility to implementers and helps with debugging and similar. Due to these considerations, the bare extension identifier pattern MUST NOT be used for any namespaced identifier.

2.4. Usage in Requests

2.4.1. Usage in Paths

Section 5 of [RFC9082] describes the use of extension identifiers in formulating URLs for RDAP queries. The extension identifiers are to be prepended to the path segments they use. For example, if an extension uses the identifier "foobar", then the path segments used in that extension are prepended with "foobar_". If the "foobar" extension defines paths "fizz" and "fazz", the URLs for this extension would be like so:

```
https://base.example/foobar_fizz
https://base.example/foobar_fazz
```

While [RFC9082] describes the extension identifier as a prepended string to a path segment, it does not describe the usage of the extension identifier as a path segment.

Note that "bare" identifiers are now explicitly forbidden (see Section 2.3).

Extensions defining new URL paths MUST explicitly define the expected responses for each new URL path. New URL paths may return existing object classes or search results as defined in [RFC9083], object classes or search results defined by the extension (see Section 2.5.3 and Section 2.5.4 below), or object classes or search results from other extensions.

Additionally, RDAP extensions MUST NOT append a path segment to an existing path segment as this increases the likelihood of collisions with the queries defined by an extension.

2.4.2. Usage in Query Parameters

Although [RFC9082] describes the use of URL query strings, it does not define their use with extensions. [RFC7480] instructs servers to ignore unknown query parameters, where a query parameter is defined as an explicitly named value in a query string. Therefore, the use of query parameters, whether prefixed with an extension identifier or not, is not supported by [RFC9082] and [RFC7480].

Despite this, there are several extensions that do specify query parameters. This document updates [RFC9082] with regard to the use of RDAP extension identifiers in URL query parameters.

When an RDAP extension defines query parameters to be used with a URL path that is not defined by that RDAP extension, those query parameter names MUST be constructed in the same manner as URL path segments (that is, extension identifier + '_' + parameter name).

Note that "bare" identifiers are now explicitly forbidden (see Section 2.3).

See Section 5.1 and Section 5.2 for other guidance on the use of query parameters, and see Section 8 and Section 9 regarding constraints on the usage of query parameters.

[RFC3986] does not exclusively define a query string as being a list of name=value pairs, however that is the convention used in RDAP. RDAP extensions MUST NOT define query strings in other forms.

2.5. Usage in Responses

2.5.1. Basic Requirements

Section 2 of [RFC9083] describes the use of extension identifiers in the JSON returned by RDAP servers. Just as in URLs, the extension identifier is prepended to JSON names to create a namespace so that the JSON name from one extension will not collide with the JSON name from another extension. Just as with unknown query parameters in URLs, clients are to ignore unknown JSON names.

The example given in [RFC9083] is as follows:

```
{
  "handle": "ABC123",
  "lunarNIC_beforeOneSmallStep": "TRUE THAT!",
  "remarks":
  [
    {
      "description":
      [
        "She sells sea shells down by the sea shore.",
        "Originally written by Terry Sullivan."
      ]
    }
  ],
  "lunarNIC_harshMistressNotes":
  [
    "In space,",
    "nobody can hear you scream."
  ]
}
```

In this example, the extension identified by "lunarNIC" is prepended to the member names of both a JSON string and a JSON array.

Note that "bare" identifiers are now explicitly forbidden (see Section 2.3).

As Section 4.1 of [RFC9083] requires the use of the "rdapConformance" data structure, and the "objectClassName" string is required of all object class instances, the complete example from above would be:

```
{
  "rdapConformance": [
    "rdap_level_0",
    "lunarNIC"
  ],
  "objectClassName": "domain",
  "handle": "ABC123",
  "ldhName": "example.com",
  "lunarNIC_beforeOneSmallStep": "TRUE THAT!",
  "remarks":
  [
    {
      "description":
      [
        "She sells sea shells down by the sea shore.",
        "Originally written by Terry Sullivan."
      ]
    }
  ],
  "lunarNIC_harshMistressNotes":
  [
    "In space,",
    "nobody can hear you scream."
  ]
}
```

2.5.2. Child JSON Values

Prefixing with the extension identifier is not required for children of a prefixed JSON object defined by an RDAP extension.

The following example shows this use with a JSON object:

```
{
  "rdapConformance": [
    "rdap_level_0",
    "lunarNIC"
  ],
  "objectClassName": "domain",
  "ldhName": "example.com",
  "remarks":
  [
    {
      "description":
      [
        "She sells sea shells down by the sea shore.",
        "Originally written by Terry Sullivan."
      ]
    }
  ],
  "lunarNIC_author":
  {
    "firstInitial": "R",
    "lastName": "Heinlein"
  }
}
```

Here the JSON name "lunarNIC_author" will separate the JSON from other extensions that may have an "author" structure. But the JSON contained within "lunarNIC_author" need not be prepended, as collision is avoided by the use of "lunarNIC_author".

2.5.3. Object Classes in Extensions

As described in [RFC9082] and Section 2.4, an extension may define new paths in URLs. If the extension describes the behavior of an RDAP query using that path to return an instance of a new class of RDAP object, the JSON names are not required to be prepended with the extension identifier as described in Section 2.5.2. However, the extension MUST define the value for the "objectClassName" string which is used by clients to evaluate the type of the response. To avoid collisions with object classes defined in other extensions, the value for the "objectClassName" MUST be prepended with the extension identifier, in the same way as for URL paths, query parameters, and JSON names:

```
{
  "rdapConformance": [
    "rdap_level_0",
    "lunarNIC"
  ],
  "objectClassName": "lunarNIC_author",
  "author":
  {
    "firstInitial": "R",
    "lastName": "Heinlein"
  }
}
```

Note that "bare" identifiers are now explicitly forbidden (see Section 2.3).

Extension authors are encouraged to use the "camel case" style described in Section 2.5.6.

Though "objectClassName" is a string and [RFC9083] does define one object class name with a space separator (i.e., "ip network"), this document disallows further use of a space character in object class names. Extensions MUST NOT define object class names using the space character or any other character that requires URL-encoding.

2.5.4. Search Results in Extensions

As described in [RFC9082] and Section 2.4, an extension may define new paths in URLs. If the extension describes the behavior of an RDAP query using the path to return an RDAP search result for a new object class, the JSON name of the search result MUST be prepended with the extension identifier to avoid collision with search results defined in other extensions.

If the search result contains object class instances defined by the extension, each instance MUST have an "objectClassName" string as defined in Section 2.5.3. For example:

```
{
  "rdapConformance": [
    "rdap_level_0",
    "lunarNIC"
  ],
  "lunarNIC_authorSearchResult": [
    {
      "objectClassName": "lunarNIC_author",
      "author":
      {
        "firstInitial": "R",
        "lastName": "Heinlein"
      }
    },
    {
      "objectClassName": "lunarNIC_author",
      "author":
      {
        "firstInitial": "J",
        "lastName": "Pournelle"
      }
    },
  ]
}
```

2.5.5. rdapConformance Population

Section 4.1 of [RFC9083] offers the following guidance on including extension identifiers in the "rdapConformance" member of an RDAP response:

A response to a "help" request will include identifiers for all of the specifications supported by the server. A response to any other request will include only identifiers for the specifications used in the construction of the response.

A strict interpretation of this wording where "construction of the response" refers only to the JSON structure would rule out the use of Section 2.1.1 extension identifiers, which are in common use in RDAP. This document clarifies the guidance. For responses to queries other than "/help", a response MUST include in the "rdapConformance" array only those extension identifiers necessary for a client to deserialize the JSON and understand the semantic meaning of the content within the JSON, and each extension identifier MUST be free from conflict with the other identifiers with respect to their syntax and semantics.

Note that this document does not update the guidance from Section 4.1 of [RFC9083] regarding "/help" responses and the "rdapConformance" array.

2.5.6. Camel Casing

The styling convention used in [RFC9083] for JSON names is often called "camel casing", in reference to the hump of a camel. In this style, the first letter of every word, except the first word, composing a name is capitalized. This convention was adopted to visually separate the namespace from the name, with an underscore between them. Extension authors are encouraged to use camel casing for JSON names defined in extensions.

3. Usage with HTTP

Extensions MUST NOT redefine the meaning of HTTP status codes or other HTTP semantics. Extensions MAY require the use of specific HTTP headers but MUST NOT redefine their meanings. Extensions defining new HTTP headers MUST have IETF consensus.

4. Extension Implementer Considerations

4.1. Redirects

[RFC7480] describes the use of redirects in RDAP. Redirects are prominent in the discovery of authoritative RIR servers, as the process outlined in [RFC9224], which uses IANA allocations, does not account for transfers of resources between RIRs. Section 4.3 of [RFC7480] instructs servers to ignore unknown query parameters (where "unknown" generally means no defined implementation behavior). As it relates to issuing URLs for redirects, servers MUST NOT blindly copy query parameters from a request to a redirect URL as query parameters may contain sensitive information, such as security credentials, not relevant to the target server of the URL. Following the advice in [RFC7480], servers MUST only place query parameters in redirect URLs when it is known by the origin server (the server issuing the redirect) that the target server (the server referenced by the redirect) can process the query parameter and is a proper target for the contents of the query parameter.

5. Extension Author Considerations

5.1. Redirects

As it is unlikely that every server in a cross-authority, redirect scenario will be upgraded to process every new extension, extensions should not rely on query parameters alone to convey information about a resource, as query parameters are not guaranteed to survive a redirect.

This does not mean extensions are prohibited from using query parameters, but rather that the use of query parameters must be applied for the scenarios appropriate for the use of the extension. Therefore, extensions MUST NOT rely on query parameters when the extension is to be used in scenarios requiring clients to find authoritative servers, or other scenarios using redirects among servers of differing authorities.

Extensions MAY use query parameters in scenarios where the client has a priori knowledge of the authoritative server to which queries are to be sent, and will be sending queries to that server directly. Searches (Section 8 of [RFC9083]) are an example scenario where a client will be operating in this way.

In general, extension authors should be mindful of situations requiring clients to directly handle redirects at the RDAP layer. Some clients may not be utilizing HTTP libraries that provide such an option, and some HTTP client libraries that do provide the option do not provide it as a default behavior. Additionally, requiring clients to handle redirects at the RDAP layer adds complexity to the client in that additional logic must be implemented to handle redirect loops, parameter deconfliction, and URL encoding. The guidance given in Section 5.2 of [RFC7480] exists to simplify clients, especially those constructed with shell scripts and HTTP command-line utilities.

5.2. Referrals

It is common in the RDAP ecosystem to link from one RDAP resource to another, such as can be found in domain registrations in gTLD DNRs. These are typically conveyed in the link structure defined in Section 4.2 of [RFC9083] and use the "application/rdap+json" media type. For example:

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

Extensions MUST explicitly define any required behavioral changes to the processing of referrals. If an extension does not make any provision in this respect, clients MUST assume the information provided by referrals requires no additional processing or modification to use in the dereferencing of the referral.

Extensions MAY define referral processing behaviors of referrals defined in other extensions or in [RFC9083].

Servers MUST NOT use multiple extensions in a response with processing requirements over the same referrals where clients would not be able to process the referrals in a deterministic way.

5.3. Extensions Referencing Other Extensions

As stated in Section 2.1.1, extensions may rely on other extensions by stipulating the usage of those other extensions.

For example, the extensions "bazz" may require the usage of structures defined in "fuzz" instead of redefining new, equivalent structures:

```
{
  "rdapConformance": [
    "rdap_level_0",
    "bazz",
    "fuzz"
  ],
  "objectClassName": "autnum",
  "startAutnum": 64496,
  "endAutnum": 64497,
  "bazz_cones": [ 64498, 64499],
  "fuzz_adjacents": [ 64500, 64501 ]
}
```

5.4. Extension Versioning

As stated in Section 2.1, RDAP extension identifiers and RDAP conformance strings are opaque, and they possess no explicit version despite the fact that some extension identifiers include trailing numbers. That is, RDAP extensions without an explicitly-defined versioning scheme are opaquely versioned.

For example, "fizzbuzz1" may be the successor to "fizzbuzz0", but it may also be an extension for a completely separate purpose. Only consultation of the definition of "fizzbuzz1" will determine its relationship with "fizzbuzz0". Additionally, "fizzbuzz99" may be the predecessor of "fizzbuzz0".

An RDAP extension definition MUST explicitly denote its compliance with any versioning scheme, such as [I-D.ietf-regext-rdap-versioning].

5.4.1. Non-overlapping Successors

Should an extension author desire to create a successor extension, the simplest method is to create a new extension (with a new extension identifier, as required) that replicates all the functionality of the previous extension.

Take for example this RDAP response for "fizzbuzz0":

```
{
  "rdapConformance": [
    "rdap_level_0",
    "fizzbuzz0"
  ],
  "objectClassName": "domain",
  "ldhName": "example.com",
  "fizzbuzz0_malwareReputationId": 1234
}
```

A successor extension may define the same functionality with equivalent structures.

```
{
  "rdapConformance": [
    "rdap_level_0",
    "fizzbuzz1"
  ],
  "objectClassName": "domain",
  "ldhName": "example.com",
  "fizzbuzz1_malwareReputationId": 1234,
  "fizzbuzz1_spamReputationId": 7890
}
```

During a transition period, both extensions could be in use.

```
{
  "rdapConformance": [
    "rdap_level_0",
    "fizzbuzz0",
    "fizzbuzz1"
  ],
  "objectClassName": "domain",
  "ldhName": "example.com",
  "fizzbuzz0_malwareReputationId": 1234,
  "fizzbuzz1_malwareReputationId": 1234,
  "fizzbuzz1_spamReputationId": 7890
}
```

5.4.2. Overlapping Successors

If extension authors are concerned about the size of responses for successor extensions using non-overlapping structures (see Section 5.4.1), they may overlap the functionality by requiring the use of the previous extension. For example:

```
{
  "rdapConformance": [
    "rdap_level_0",
    "fizzbuzz0",
    "fizzbuzz1"
  ],
  "objectClassName": "domain",
  "ldhName": "example.com",
  "fizzbuzz0_malwareReputationId": 1234,
  "fizzbuzz1_spamReputationId": 7890
}
```

And at some future time, a successor such as "fizzbuzz9" may no longer need the function provided by "fizzbuzz0" and may cease to reference it.

5.4.3. Breaking Changes in Successors

With the current extension model, an extension with a successor with breaking changes is indistinguishable from a new, unrelated extension. Additionally, there is no signaling mechanism in RDAP to specify successors with breaking changes. Implementers of such changes should consider the following:

* whether the new version of the extension can be provided alongside the old version of the extension, so that a service can simply support both during a transition period;

* whether some sort of client signaling should be supported, so that clients can opt for the old or new version of the extension in responses that they receive (see [I-D.ietf-regext-rdap-x-media-type] for an example of how this might work); and
* whether the extension itself should define how versioning is handled within the extension documentation.

When using a transition period between two versions of an extension by using both versions, the successor must not conflict with the predecessor. Typically, this is not an issue when the rules of RDAP namespaced identifiers are followed (see #(bare_extensions)), but care should be taken if the extensions specify other behaviors not protected by namespaces, particularly referrals (see Section 5.2).

Breaking changes may also occur in requirements for processing of data in protocol elements that appear in both a successor and predecessor. For example, a profile extension (see #(profiles)) may require domain names always end with a dot ("."). Should its successor remove this requirement this could be considered a breaking change.

5.4.4. Evolving Extensions without Signaled Changes

Because RDAP clients ignore unrecognized JSON names and query parameters, it is possible to extend an RDAP extension by adding new JSON names or query parameters within the same namespace of an existing RDAP extension without changing the extension identifier or other signaling methods (see [I-D.ietf-regext-rdap-versioning]).

In this scenario, clients that are not updated to recognize the new elements should simply ignore them. The same is also true for referrals (see Section 5.2).

However, the introduction of new object classes into an existing extension will cause most clients to process no information and will cause some clients to produce errors.

Extensions MUST NOT be evolved as described in this section because there is no explicit signal to clients regarding these extensions. This lack of signal will lead to difficulty in troubleshooting issues and could mislead client implementers to believe their software is fully conforming with the extension specification when it is not.

5.5. Extension Specification Content

The primary purpose of an RDAP extension specification is to aid in the implementation of RDAP clients. Extension authors should consider the following content guidelines:

1. Examples of RDAP JSON should be generously given, especially in areas of the specification which may be complex or difficult to describe with prose.
2. Normative references, i.e., references to materials that are required for the interoperability of the extension, MUST be stable and non-changing and MUST NOT be denoted as a "work in progress" or similar description.
3. Extension specifications MUST NOT define requests and responses exchanges over an unencrypted HTTP connection. Extensions should also be compliant with the security considerations of [RFC7481].
4. Extension specifications MUST NOT forbid the use of RDAP services over IPv6.
5. The use of the various RDAP extension points, as described in Section 1.1, should be clearly delineated.

5.6. Extension Definitions

Extensions must be documented in an RFC or in some other permanent, stable, and readily available reference, in sufficient detail that interoperability between independent implementations is possible.

Though RDAP gives each extension its own namespace, the definition of an extension may reuse definitions found in the base RDAP specification or in any other registered extension.

[RFC9083] notes that the extension identifiers provide a "hint" to the client as to how to interpret the response. This wording does not intentionally restrict the extension to defining only JSON values within the extension’s namespace. Therefore, an extension may define the use of its own JSON values together with the use of JSON values from other extensions or RDAP specifications. As with the [icann-profile] and [nro-profile] extensions, the extension may simply signal policy applied to previously-defined RDAP structures (see Section 2.1.1).

6. Existing Extension Registrations

The following extensions have been registered with IANA, but do not comply with the requirements set out in the base specifications, as clarified by this document:

* Extension identifier: fred

- RDAP conformance value: fred_version_0
- Field/path prefix: fred

* Extension identifier: artRecord

- RDAP conformance value: artRecord_level_0
- Field/path prefix: artRecord

* Extension identifier: platformNS

- RDAP conformance value: platformNS_level_0
- Field/path prefix: platformNS

* Extension identifier: regType

- RDAP conformance value: regType_level_0
- Field/path prefix: regType

Client authors should be aware that responses that make use of these extensions may require special handling on the part of the client. Also, while these extensions will be retained in the registry, future extensions that are similarly non-compliant will not be registered.

7. IANA Considerations

7.1. RDAP Extensions Registry

[RFC7480] defines the [rdap-extensions] registry. This document does not change the purpose of this registry but does update the structure and procedures to be used by its expert reviewers.

7.1.1. Deprecation Date

IANA is instructed to add a new "Deprecation Date" field to each registration in the registry. This field is to remain empty unless IANA is given a date to place in the field. A registrant, as denoted by the contact field of the registry, may request of IANA to deprecate an RDAP extension. The IETF may request of the IANA to deprecate any RDAP extension in the registry. When deprecating an entry in this registry, IANA is to record the date of the request in the "Deprecation Date" field. The "Deprecation Date" field should use the date format specified in [RFC3339].

7.1.2. Registration Procedures

Extension authors are encouraged but not required to seek an informal review of their extension by sending a request for review to regext@ietf.org or its successor.

The registration template of this registry is found in [RFC7480] and is unchanged. It is requested of the IANA that all registrations be forwarded to regext@ietf.org or its successor.

Extensions MUST be documented in a stable, non-changing, and readily available reference, in sufficient detail that interoperability between independent implementations is possible, and MUST NOT be denoted as a "work in progress" or similar description.

7.1.3. Expert Review

The RDAP Extensions Registry should have as a minimum three expert reviewers and ideally four or five. An expert reviewer assigned to the review of an RDAP extension registration must have another expert reviewer double-check any submitted registration.

Expert reviewers are to use the following criteria for extensions defined in this document, [RFC7480], [RFC9082], and [RFC9083]. The following is a non-exhaustive checklist:

1. Does the extension define an extension identifier following the naming conventions described in Section 2.2 and Section 2.5.6?
2. If the extension defines new queries, does it clearly describe the expected results of each new query?
3. Does the extension follow the JSON naming requirements as described in Section 2.5?
4. If the extension is a newer version of an older extension, does the extension specification clearly describe if it is backwards-compatible (see Section 5.4)?
5. If the extension registers new values in an IANA registry used by RDAP, does it describe how a client is to use those values?
6. If the extension is a new registration, is it a case-variant of an existing registration (see Section 2.2)?

As noted in Section 2.2, any new registration that is a case-variant of an existing registration MUST be rejected.

RDAP clients SHOULD match values in this registry using case-insensitive matching to handle server implementations incorrectly using the wrong case.

7.2. RDAP JSON Values Registry

Section 10.2 of [RFC9083] defines the [rdap-json-values]. This registry contains values to be used in the JSON values of RDAP responses. Registrations into this registry may occur in IETF-defined RDAP extensions or via requests to the IANA. Authors of RDAP extensions not defined by the IETF MAY register values in this registry via requests to the IANA. IANA is requested to send a copy of any request not originating from the IETF to regext@ietf.org or its successor.

This document does not change the [rdap-json-values] nor its purpose. However, this document does update the procedures for registrations and the processes to be used by its expert reviewers.

In addition to the registration of values, RDAP extensions defined by the IETF and other IETF specifications MAY define additional value types (the "type" field). These specifications MUST describe the specific JSON field to be used for each new value type.

Section 10.2 of [RFC9083] defines the criteria for the values. Of these, criteria two states:

> Values must be strings. They should be multiple words separated by single space characters. Every character should be lowercased. If possible, every word should be given in English and each character should be US-ASCII.

All registrations SHOULD meet these requirements. However, there may be scenarios in which it is more appropriate for the values to follow other requirements, such as for values also used in other specifications or documents. In all cases, it should be understood that additional registrations of RDAP JSON values occurring after the specification of the value’s type in the registry may not be recognized by clients, and therefore either ignored or passed on to users without processing.

Designated experts MUST reject any registration that is a duplicate of an existing registration, and all registrations are to be considered case-insensitive. That is, any new registration that is a case-variant of an existing registration should be rejected.

RDAP clients SHOULD match values in this registry using case-insensitive matching to handle scenarios in which servers incorrectly use the wrong case.

Definitions of new types (see above) MAY additionally constrain the format of values for those new types beyond the specification of this document and [RFC9083]. Designated experts MUST evaluate registrations with those criteria.

The [rdap-json-values] registry should have as a minimum three expert reviewers and ideally four or five. An expert reviewer assigned to the review of an RDAP JSON values registration must have another expert reviewer double-check any submitted registration.

Expert reviewers are to use the criteria defined in Section 10.2 of [RFC9083].

8. Security Considerations

Section 2.4.2 describes the usage of query parameters and Section 5.1 describes the restrictions extensions must follow to use them. Section 4.3 of [RFC7480] instructs servers to ignore unknown query parameters. As it relates to issuing URLs for redirects, servers MUST NOT blindly copy query parameters from a request to a redirect URL as query parameters may contain sensitive information, such as security credentials or tracking information, not relevant to the target server of the URL. Following the advice in [RFC7480], servers MUST only place query parameters in redirect URLs when it is known by the origin server (the server issuing the redirect) that the target server (the server referenced by the redirect) can process the query parameter and the contents of the query parameter are appropriate to be received by the target.

9. Privacy Considerations

Section 2.4.2 describes the usage of query parameters and Section 5.1 describes the restrictions extensions must follow to use them. As query parameters have been known to be used to subvert the privacy preferences of users in HTTP-based protocols, server MUST NOT blindly copy query parameters from a request to a redirect URL as described in Section 8 and extensions MUST follow the constraints of query parameter usage as defined in Section 5.1.

10. Acknowledgments

The following individuals have provided feedback and contributions to the content and direction of this document: James Gould, Scott Hollenbeck, Ties de Kock, Pawel Kowalik, Daniel Keathley, and Mario Loffredo.

11. References

11.1. 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>.

[RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <https://www.rfc-editor.org/info/rfc3339>.

[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>.

[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>.

[RFC7481] Hollenbeck, S. and N. Kong, "Security Services for the Registration Data Access Protocol (RDAP)", STD 95, RFC 7481, DOI 10.17487/RFC7481, March 2015, <https://www.rfc-editor.org/info/rfc7481>.

[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>.

[RFC9082] Hollenbeck, S. and A. Newton, "Registration Data Access Protocol (RDAP) Query Format", STD 95, RFC 9082, DOI 10.17487/RFC9082, June 2021, <https://www.rfc-editor.org/info/rfc9082>.

[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>.

[RFC9224] Blanchet, M., "Finding the Authoritative Registration Data Access Protocol (RDAP) Service", STD 95, RFC 9224, DOI 10.17487/RFC9224, March 2022, <https://www.rfc-editor.org/info/rfc9224>.

11.2. Informative References

[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>.

[I-D.ietf-regext-rdap-x-media-type]
Newton, A. and J. Singh, "Extensions Parameter for the RDAP Media Type", Work in Progress, Internet-Draft, draft-ietf-regext-rdap-x-media-type-04, 2 July 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-regext-rdap-x-media-type-04>.

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

[icann-profile]
ICANN, "gTLD RDAP Profile", 2024, <https://www.icann.org/gtld-rdap-profile>.

[link-relations]
IANA, "Link Relations", <https://www.iana.org/assignments/link-relations/link-relations.xhtml>.

[nro-profile]
NRO, "NRO RDAP Profile", 2021, <https://bitbucket.org/nroecg/nro-rdap-profile/raw/v1/nro-rdap-profile.txt>.

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

[rdap-json-values]
IANA, "RDAP JSON Values", <https://www.iana.org/assignments/rdap-json-values/rdap-json-values.xhtml>.

Authors' Addresses

Andy Newton
ICANN
Email: andy@hxr.us

Jasdip Singh
ARIN
Email: jasdips@arin.net

Tom Harrison
APNIC
Email: tomh@apnic.net