.it RDAP public test server

This experimental RDAP service is based on *.it Registry* public test environment registration data. The response data generated by this server are provided by *.it Registry* for experimental testing purposes only, and are not, and should not be considered to be, official WHOIS or RDAP query responses or data. This service allows the user to perform all the queries described in RFC 7482. Furthermore, it provides the user with additional capabilities about domain availability, sorting and paging, partial response, reverse search, advanced query and filtering and OpenAPI 3.0 specification.

Table of contents

1. Basic lookup queries

- https://rdap.pubtest.nic.it/domain/1nce.it
- https://rdap.pubtest.nic.it/nameserver/ns3.ix-glue-test-1353440617.it
- https://rdap.pubtest.nic.it/entity/C-CFL8306-LYFBBM
- https://rdap.pubtest.nic.it/help

2. Lookup queries including domain availability extensions

This extension is described in draft-newton-regext-rdap-domain-availability-00

With availabilityCheck extension

- https://rdap.pubtest.nic.it/domain/toscana.it?availabilityCheck=1 (geographic)
- https://rdap.pubtest.nic.it/domain/www.it?availabilityCheck=1 (unassignable)
- https://rdap.pubtest.nic.it/domain/comune-pisa.it?availabilityCheck=1 (reserved)

- https://rdap.pubtest.nic.it/domain/xn--.it?availabilityCheck=1 (invalid)
- https://rdap.pubtest.nic.it/domain/marioloffredo.it?availabilityCheck=1 (available)
- https://rdap.pubtest.nic.it/domain/1nce.it?availabilityCheck=1 (registered)

With availabilityInformation extension

If domain is available the response is the same as availabilityCheck=1, otherwise the response is the same as a basic lookup query.

- https://rdap.pubtest.nic.it/domain/marioloffredo.it?availabilityInformation=1 (available)
- https://rdap.pubtest.nic.it/domain/1nce.it?availabilityInformation=1 (registered)

3. Basic search queries

It is possible to use wildcard character to ask for partial string matching.

- https://rdap.pubtest.nic.it/domains?name=*nr.it
- https://rdap.pubtest.nic.it/domains?nsLdhName=ns3.ix-glue-test-*.it
- https://rdap.pubtest.nic.it/domains?nsIp=212.97.32.3
- https://rdap.pubtest.nic.it/nameservers?name=ns3.ix-glue-test-*.it
- https://rdap.pubtest.nic.it/nameservers?ip=212.*
- https://rdap.pubtest.nic.it/entities?fn=Cons*
- https://rdap.pubtest.nic.it/entities?handle=TUY*

4. Search queries including searchtype=regex extension

This extension is described in draft-fregly-regext-rdap-search-regex-03

To apply Base64 encoding, the user can rely on the online tool at https://www.base64encode.org/

- https://rdap.pubtest.nic.it/domains?name=LipuclwuaXQk&searchtype=regex (base64Encoded for ".*nr\.it$")
- https://rdap.pubtest.nic.it/nameservers?ip=XjIxMlwuLio=&searchtype=regex (base64Encoded for "^212\..*")
- https://rdap.pubtest.nic.it/entities?handle=XlRVWS4q&searchtype=regex (base64Encoded for "^TUY.*")

5. Search queries enabling domain suggestion

This extension is described in Technical Report IIT-06-2018

This server implements the new value *suggestion* for the query parameter *searchtype* to enable domain suggestion. The new search is allowed only for the *domains?name* search path. The search pattern must be a domain name in LDH or U-label formats. Partial matching is not allowed. Some

additional parameters could be used:

- **language**=(de|en|fr|it) Default:it
- **maxLength**=[3-63]
- **showRegistered**=(true|false) Default:false
- **showCensurable**=(true|false) Default:false
- **useNumbers**=(true|false) Default:false
- **useHypens**=(true|false) Default:false
- **useIdns**=(true|false) Default:false

Here in the following some examples:

- https://rdap.pubtest.nic.it/domains?name=auto-lavaggio.it&searchtype=suggestion&showRegistered=true
- https://rdap.pubtest.nic.it/domains?name=auto-wäsche.it&searchtype=suggestion&language=de&useIdns=true
- https://rdap.pubtest.nic.it/domains?name=lave-auto.it&searchtype=suggestion&language=fr&maxLength=20

6. Search queries including operators for counting, sorting and paging results

This extension is described in draft-loffredo-regext-rdap-sorting-and-paging-05

Count parameter

The count parameter reports the total number of results in the *totalCount* field of the *paging_metadata* section.

- https://rdap.pubtest.nic.it/entities?handle=R*-REG&count=true

Sort parameter

The sort parameter allows a client to request a specific sort order for the result set.

- Sort parameter properties:
  - **Domain**: *registrationDate*, *lastChangedDate*, *expirationDate*, *ldhName*
  - **Nameserver**: *ldhName*, *ipV4*, *ipV6*
  - **Entity**: *registrationDate*, *lastChangedDate*, *org*, *city*, *country*, *email*, *voice*

Current sort and other available sorts are reported in the response section *sorting_metadata*.

Here in the following some examples:

- https://rdap.pubtest.nic.it/entities?handle=R*-REG&sort=handle
- https://rdap.pubtest.nic.it/entities?handle=R*-REG&sort=registrationDate

- https://rdap.pubtest.nic.it/entities?handle=R*-REG&sort=email,registrationDate:d

Limit & offset parameters

The limit and offset parameters allow a client to request a specific portion of the entire result set.

If used together, limit and offset implement result set pagination. This RDAP server implements cursor-based pagination, too. Such a method occurs when the query string doesn't contain the sort parameter. The server uses the *links* array of the *paging_metadata* section to provide the user with a ready-made reference to the next page of the result set.

Here in the following some examples:

- https://rdap.pubtest.nic.it/entities?handle=R*-REG&limit=10 (it returns top 10 results)
- https://rdap.pubtest.nic.it/entities?handle=R*-REG&offset=10 (it skips the top 10 results and returns the results starting from position 11 - the result set can be truncated due to server limits)
- https://rdap.pubtest.nic.it/entities?handle=R*-REG&limit=10&offset=10 (it skips the top 10 results and returns the next 10 results starting from position 11)

7. Search queries asking for partial response

This extension is described in draft-loffredo-regext-rdap-partial-response-02

The **fieldSet** parameter allows the user to ask for a server pre-defined set of fields in the response.

Three sets have been defined:

- *id*: it contains only the *objectClassName* field and the key field (e.g. *handle* for entities). This field set can be used when the client wants to obtain a collection of object identifiers;
- *brief*: it contains the fields that can be included in a *short* response. This field set can be used when the client is asking for a subset of the full response which gives a basic knowledge of each object;
- *full*: it contains all the information the server can provide for a particular object.

Current field set and other available field sets are reported in the response section *subsetting_metadata*.

Here in the following an example:

- https://rdap.pubtest.nic.it/entities?handle=R*-REG&fieldSet=id

8. Search queries enabling reverse search

This extension is described in draft-loffredo-regext-rdap-reverse-search-03

This RDAP server implements four query parameters enabling reverse search based on the domains-entities relationship (the classic Reverse Whois scenario):

- **entityHandle**: it returns the domains related to an entity whose handle is given;
- **entityFn**: it returns the domains related to an entity whose fn is given;
- **entityEmail**: it returns the domains related to an entity whose email is given;
- **entityAddr**: it returns the domains related to an entity whose postal address is given.

In this case, the search pattern is a JSON object including two members:

- *value*: it represents the search pattern to be applied to the corresponding entity field and can be a JSON type primitive or object;
- *role*: it is a string whose possible values are those detailed in Section 10.2.4 of RFC 7483. In this implementation the allowed values are: registrar, registrant, administrative, technical.

In the case of the entityAddr parameter, the search pattern is a JSON object containing the information described in Section 2.4 of RFC 5733 respectively: *street*, *city*, *sp*, *pc* and *cc*. All the members of the postal address object are optional but at least one is required. The constraints on the members are implicitly joined by *and*.

Here in the following some examples:

- <https://rdap.pubtest.nic.it/domains?entityHandle={"value":"REGISTRY-REG","role":"registrar"}>
- <https://rdap.pubtest.nic.it/domains?entityFn={"value":"*Loffredo","role":"registrant"}>
- <https://rdap.pubtest.nic.it/domains?entityEmail={"value":"mario.loffredo@iit.cnr.it","role":"technical"}>
- <https://rdap.pubtest.nic.it/domains?entityAddr={"value":{"cc":"IT","sp":"PI"},"role":"registrant"}>

9. Search queries including query and/or filter parameters

This extension is described in Technical Report IIT-07-2018

The query and filter parameters allow the user to submit complex search conditions. In detail:

- **query**: it allows the user to submit a complex search. It must be used in place of a search path segment (e.g. name for domains).
- **filter**: it allows the user to filter the results according to the values of those properties that are not used as search path segments (e.g. status for domains).

In both cases, the value is a JSON object representing predicates whose complexity ranges from very simple to extremely complicated. In fact, traditionally, a search condition includes a set of predicates joined by the logical operators *and*, *or* and *not*. A predicate contains three components:

- a property name;
- an allowed operator for the property;
- a filter value (or a list of values) whose type is allowed for the property.

Therefore, the structure of a predicate is described by the following JCR (JSON Content Rules) syntax:

```
@{root} $expression = {
 (
   $or_expression   |
   $and_expression  |
   $not_expression  |
   [ $predicate + ] |
   $predicate
  )
}

$or_expression = {
    "or" : [ $expression, $expression + ]
}

$and_expression = {
    "and" : [ $expression, $expression + ]
}

$not_expression = {
    "not" : $expression
}

$predicate = [
/^[A-Za-z]+$/,
    (
  ("isnull"|"isnotnull")                              |
  ("eq"|"ne"), $basic_or_object_value                 |
  ("le"|"lt"|"gt"|"ge"), $not_pattern_value           |
  "between", [ $not_pattern_value, $not_pattern_value ] |
  ("in"|"notin"|"any"|"all"|"exactly"), $array_value
)
]
$basic_or_object_value = (
                          $basic_value  |
                          $object_value
                          )

$object_value = { // : any * }

$basic_value = @{not} (
                       { // : any * } |
                       [ any * ]      |
                       null
                      )

$not_pattern_value = @{not} (
                    { // : any * } |
                    [ any * ]      |
```

```
                    null           |
                    $pattern_value
                             )

$pattern_value = /^[^\*]*\*[^\*]*$/

$array_value = [ $not_pattern_value + ]
```

Property names

- Property names that can be used in the case of query parameter:
  - **Domain**: *name*, *nsLdhName*, *nsIp*, *entityHandle*, *entityFn*, *entityEmail*, *entityAddr*
  - **Nameserver**: *name*, *ip*
  - **Entity**: *handle*, *fn*
- Property names that can be used in the case of filter parameter:
  - **Domain**: *registrationDate*, *lastChangedDate*, *expirationDate*, *transferDate*, *status*
  - **Entity**: *registrationDate*, *lastChangedDate*, *org*, *cc*, *city*, *country*, *email*, *voice*, *roles*, *status*

Operators

Together with the logical operators, it is reasonable to expect the presence of commonly used comparison operators like *equal*, *not equal*, *less than* and so on. Specific operators about strings like *contains*, *starts with*, *ends with* can be implemented using the *equal* operator and the wildcard. Appropriate operators on arrays should be considered too. In the following, a list of the operators is shown:

- no values: *isnull*, *isnotnull*
- one value: *eq*, *ne*, *le*, *ge*, *lt*, *gt*
- array of two values: *between*
- array of N values: *in*, *notin*

The *any*, *all* and *exactly* operators are used with the following meaning:

- *any* means that the property must contain at least one of the values in the array;
- *all* means that the property must contain all the values in the array, but it could contain also additional values;
- *exactly* means that the property must contain all the values in the array and cannot contain any additional value.

If an array contains only one value, any and all have the same meaning.

isnull and *isnotnull* are used when the predicate represents, respectively, the absence or the

presence of a property in the expected results:

- ["transferDate","isnull"]

All predicates in an array are implicitly combined by *and*:

- {"and":[["registrationDate","ge","2018-01-20"],["expirationDate","le","2019-01-20"]]}
- [["registrationDate","ge","2018-01-20"],["expirationDate","le","2019-01-20"]]

The operator *between* is a shortcut for two predicates combined by *and* including the same property:

- {"and":[["registrationDate","ge","2018-01-20"],["registrationDate","le","2019-01-20"]]}
- ["registrationDate","between",["2018-01-20","2019-01-20"]]

The operator *in* is a shortcut for N predicates combined by *or* including the same property and the *eq* operator:

- {"or":[["cc","eq","it"],["cc","eq","ch"],["cc","eq","de"],["cc","eq","fr"]]}
- ["cc","in",["it","ch","de","fr"]]

Value types

Value types can be:

- *string*
- *number*
- *boolean*
- *datetime* (a string written according to RFC 3339 *full date* and *date time* formats)
- *array* of a primitive type
- *object*

Examples

Here in the following some examples:

- <https://rdap.pubtest.nic.it/domains?name=we*.it&filter=["registrationDate","ge","2018-01-20"]>
- <https://rdap.pubtest.nic.it/domains?name=we*.it&filter={"or":[["registrationDate","ge","2018-01-20"],["expirationDate","le","2019-01-20"]]}>
- <https://rdap.pubtest.nic.it/domains?name=we*.it&filter={"not":{"or":[["registrationDate","ge","2018-01-20"],["expirationDate","le","2019-01-20"]]}}>
- <https://rdap.pubtest.nic.it/domains?name=wu*it&filter=["transferDate","isnull"]>
- <https://rdap.pubtest.nic.it/domains?query=[["name","eq","test-*.it"],["nsLdhName","eq","wns1.rtr-dev.com"]]>
- <https://rdap.pubtest.nic.it/domains?query=[["name","eq","test-*.it"],["entityAddr","eq",{"value":{"cc":"be"},"role":"registrant"}]]&filter={"or":[["registrationDate","ge","2018-01-20"],["expirationDate","le","2019-01-20"]]}>

Current filter and other available filters are reported in the response section *filtering_metadata*.

10. Search queries mixing the query parameters

Here in the following some examples:

- Search domains whose name starts with *we* or *wu*
  - <https://rdap.pubtest.nic.it/domains?query={"or":[["name","eq","we.*it*"],["*name*","*eq*","*w**u*.it"]]}>
- How many are there ?
  - <https://rdap.pubtest.nic.it/domains?query={"or":[["name","eq","we.*it*"],["*name*","*eq*","*w**u*.it"]]}&count=1>
- Which is the oldest ?
  - <https://rdap.pubtest.nic.it/domains?query={"or":[["name","eq","we.*it*"],["*name*","*eq*","*w**u*.it"]]}&count=1&sortby=registrationDate>
- What are the domains registered since *2015* ?
  - <https://rdap.pubtest.nic.it/domains?query={"or":[["name","eq","we.*it*"],["*name*","*eq*","*w**u*.it"]]}&count=1&sortby=registrationDate&filter=["registrationDate","gt","2015-01-01"]>
- What are the *inactive* domains registered since *2015* ?
  - <https://rdap.pubtest.nic.it/domains?query={"or":[["name","eq","we.*it*"],["*name*","*eq*","*w**u*.it"]]}&count=1&sortby=registrationDate&filter=[["registrationDate","gt","2015-01-01"],["status","any",["inactive"]]]>
- Return only the domain names sorted by *ldhName*
  - <https://rdap.pubtest.nic.it/domains?query={"or":[["name","eq","we.*it*"],["*name*","*eq*","*w**u*.it"]]}&count=1&sortby=ldhName&filter=[["registrationDate","gt","2015-01-01"],["status","any",["inactive"]]]&fieldSet=id>

11. Metadata

This RDAP server includes in the response four metadata sections:

- The **sorting_metadata** section contains the following fields:
  - *currentSort*: the value of sort parameter as specified in the query string;
  - *availableSorts*: an array of objects each one describing available sorting criterion not reported in the current sort:
    - *property*: the name that can be used by the client to request the sorting criterion;
    - *jsonPath*: the JSON Path of the RDAP field corresponding to the property;
    - *default*: whether the sorting criterion is applied by default;
    - *links*: an array of links containing the query string that applies the sorting criterion.

- The **paging_metadata** section contains the following fields:
    - *totalCount*: a numeric value representing the total number of objects found;
    - *pageCount*: a numeric value representing the number of objects returned in the current page;
    - *offset*: a numeric value identifying the start of current page in the result set;
    - *nextOffset*: a numeric value identifying the start of the next page in the result set or null if the result set has been completely scrolled;
    - *links*: an array of links containing the reference to next page.

Both *offset* and *nextOffset* appear if offset-based pagination occurs.

- The **subsetting_metadata** section contains the following fields:
    - *currentFieldSet*: the value of fieldSet parameter as specified in the query string;
    - *availableFieldSets*: an array of objects each one describing an available field set not corresponding to the current field set:
        - *name*: the field set name;
        - *description*: a human-readable description of the field set;
        - *default*: whether the field set is applied by default;
        - *links*: an array of links containing the query string that applies the field set.
- The **filtering_metadata** section contains the following fields:
    - *currentFilter*: the value of filter parameter as specified in the query string not reported in the current filter;
    - *availableFilters*: an array of objects each one describing another available filter:
        - *property*: the name that can be used by the client to request the filter;
        - *jsonPath*: the JSON Path of the RDAP field corresponding to the property.

12. HTTP HEAD method

The HEAD method can be used only for lookup queries (help included).

13. Bootstrapping

Here in the following some examples:

- https://rdap.pubtest.nic.it/domain/nic.cz
- https://rdap.pubtest.nic.it/domain/people.cz
- https://rdap.pubtest.nic.it/nameserver/a.ns.nic.cz
- https://rdap.pubtest.nic.it/autnum/3741
- https://rdap.pubtest.nic.it/ip/192.12.193.108

14. Specification

RDAP servers can provide different capabilities:

- some query paths cannot be available;
- bootstrapping is not implemented;
- queries can be extended with additional parameters;
- authentication and access levels can be implemented;
- responses can contain proprietary extensions.

How could RDAP clients face with such a diversity?

Servers could provide their own policies via a REST API specification format (OpenAPI, RAML, API Blueprint, JSON API, JSON Schema). This server provides a specification based on OpenAPI 3.0 format. Bootstrapping based on objectTag can help find the desired specification. This implementation simulate the response coming from some RDAP servers.

Here in the following some examples:

- https://rdap.pubtest.nic.it/specification
- https://rdap.pubtest.nic.it/specification/BRNIC
- https://rdap.pubtest.nic.it/specification/GOOGLE
- https://rdap.pubtest.nic.it/specification/VRSN
- https://rdap.pubtest.nic.it/specification/STD (RDAP standard server)

15. Contact Information

For any question or remark, please contact hostmaster@nic.it