RIPE Database Reference Manual

João Luis Silva Damas
Andrei Robachevsky

Document ID: ripe-223
Date: 5 July 2001
Obsoletes: ripe-189 and ripe-157

Abstract

This document describes the functionality of version 3.0 of the RIPE Database that uses the Routing Policy Specification Language (RPSL) [1] to represent all database objects. It also implements the Routing Policy System Security (RPSS) [2] to provide authorisation mechanisms to enable a higher level of security for Internet Routing Registries (IRR). Though this document tends to be self-contained, the reader is encouraged to read the RPSL [1] and RPSS [2] specifications for more detailed information, examples of usage and definitions. For a tutorial on RPSL, the reader should read the RPSL applications document [3].

Table of Contents

Introduction

The RIPE Network Management Database contains information about IP address space allocations and assignments, routing policies and reverse delegations in the RIPE region and beyond.

It also contains some information about forward domain names. However, the information about domain names is there as a convenient reference only; it is not the domain name registry that is run by the country code Top Level Domain (ccTLD) administrators of Europe and surrounding areas. Please see the IANA ccTLD Database for a full list of the ccTLD administrators.

The information in the RIPE Database is available to the public for agreed Internet operation purposes, but is under copyright. Please see Appendix A3 "Copyright information".

The phrase "RIPE Database" is often used to refer to the interface software, rather than the information that is stored in the database. In ambiguous situations, this reference manual will make clear what is being discussed.

This document describes the functionality of version 3.0 of the RIPE Database software. This version uses the Routing Policy Specification Language (RPSL) [1], a successor of RIPE-181 language [4], to represent all database objects. It also implements the Routing Policy System Security(RPSS) [2] to provide authorisation mechanisms to enable a higher level of security for the routing registry, which is part of the RIPE Database.

This document is self-contained. However, it does not contain examples of usage and illustrations of principles and concepts related to the RIPE Database functionality and implementation. The "RIPE Database User's Manual" [5] and the "RIPE Database Operations Manual" [5] will fill this gap and provide examples and detailed information on how to interact with the RIPE Database and how to configure and run RIPE Database server software. The reader is also encouraged to read the RPSL [1] and RPSS [2] specifications for more detailed information, examples of usage and definitions. For a tutorial on RPSL, the reader should read the RPSL applications document [3].

1.0 Database objects and attributes

The RIPE Network Management Database (often called the ?RIPE Database?) contains records of:

- allocations and assignments of IP address space (the IP address registry);
- domain names (the domain registry);
- routing policy information (the routing registry);
- contact information (details of people who are responsible for the operation of networks or routers, and those who are responsible for maintaining information in the RIPE Database).

Note: the information in the domain registry, except for in-addr.arpa domains, has no effect on operations; it is stored in the RIPE Database as a convenient reference. It should not be considered complete or authoritative for any purpose.

1.1 Object representation

Records in the RIPE Database are known as **"objects"**. The syntax of the database objects is defined by RPSL, which is described in [1]. An object belongs to one of the object types, or classes. These two terms are used interchangeably through the document. The following object types are stored in the RIPE Database:

Table 1 Object types supported in the RIPE Database

Object type (Class name)	**Abbreviated name**	**Description**
as-block	ak	Represents delegation of a range of AS numbers to a given repository.
as-set	as	Defines a set of **aut-num** objects.
aut-num	an	Represents an Autonomous System (AS) in the database. Is used to describe external routing policy of the AS.
domain	dn	Represents forward or reverse domain registrations.
filter-set	fs	Defines a set of routes that are matched by its filter.
inet6num	i6	Contains information on allocations and assignments of IPv6 address space.
inetnum	in	Contains information on allocations and assignments of IPv4 address space.
inet-rtr	ir	Represents a router in the database.
key-cert	kc	Represents a public key certificate that is stored on the server and may be used with a **maintainer** object for authentication when performing updates.
limerick	li	Represents a humorous poem that has five lines and the rhyme scheme "aabba".
mntner	mt	Specifies authentication information required to authorise creation, deletion or modification to the objects protected by the mntner.
peering-set	ps	Defines a set of peerings.
person	pn	Contains information about technical or administrative contacts.
role	ro	Contains information about technical or administrative contacts, but describes a role performed by one or more human beings.
route	rt	Represents a route advertised in the Internet.
route-set	rs	Defines a set of routes.
rtr-set	is	Defines a set of routers.

A **database object** is defined as a list of attribute-value pairs in text. Each attribute-value pair is written on a separate line. The attribute name starts at column 0, followed by the character ? : ? and followed by the value of the attribute. The attribute that has the same name as the object?s class should be specified first. An attribute?s value can be split over multiple lines, by having a space, a tab or a plus ("+") character as the first character of the continuation lines. The character "+" for line continuation allows attribute values to contain blank lines. More spaces may optionally be used after the continuation character to increase readability. The order of attribute-value pairs is significant. An object’s description may contain comments. A comment can be anywhere in an object’s definition, it starts at the first "#" character on a line and ends at the first end-of-line character. A comment cannot start at column 0. White space characters can be used to improve readability. The object’s representation ends when a blank line is encountered.

Attributes can be mandatory or optional: A mandatory attribute MUST be defined for all objects of the class; optional attributes can be skipped. Attributes can also be single or multiple-valued. Multiple-valued attributes may have several attribute-value records in an object, while a single valued attribute may appear only once. Each object is uniquely identified by a set of attributes, referred to as the class primary key.

The value of an attribute has a type, which defines the syntax of the attribute value. Please refer to Appendix A1 "Object attributes" for a detailed description of the attributes supported in the RIPE Database.

1.2 Object types

This section describes object types (classes) supported in the RIPE Database along with the object templates. The following definitions are used in the templates:

[mandatory]	At least one instance of this attribute must be defined in an object of the class.
[optional]	Attribute is optional in the objects of the class and can be omitted.
[generated]	Attribute is automatically generated by the server.
[single]	An object MUST NOT contain more than one instance of this attribute.
[multiple]	An object MAY contain more than one instance of this attribute.
[look-up key]	Attribute is indexed.
[inverse key]	Attribute is in the ?reverse? index.
[primary key]	Attribute is (part of) the primary key.
[primary/look-up key]	Attribute is both indexed and is (part of) the primary key.

In an object template the first column represents an attribute, the second and third columns specify the type of the attribute and the fourth column tells whether the attribute is (part of) a database key for the object.

1.2.1 as-block

An **as-block** object is needed to delegate a range of AS numbers to a given repository. This object may be used for authorisation of the creation of **aut-num** objects within the range specified by the "as-block:" attribute. The template of **as-block** class is shown in Figure 1.2.1.

```
as-block:       [mandatory]  [single]     [primary/look-up key]
descr:          [optional]   [multiple]   [ ]
remarks:        [optional]   [multiple]   [ ]
tech-c:         [mandatory]  [multiple]   [inverse key]
admin-c:        [mandatory]  [multiple]   [inverse key]
notify:         [optional]   [multiple]   [inverse key]
mnt-lower:      [optional]   [multiple]   [inverse key]
mnt-by:         [mandatory]  [multiple]   [inverse key]
```

```
changed:        [mandatory]  [multiple]   [ ]
source:         [mandatory]  [single]     [ ]
```

Fig. 1.2.1 as-block template

1.2.2 as-set

An **as-set** object defines a set of **aut-num** objects. The attributes of the **as-set** class are shown in Figure 1.2.2. The "as-set:" attribute defines the name of the set. It is an RPSL name that starts with "as-". The "members:" attribute lists the members of the set. The "members:" attribute is a list of AS numbers, or other as-set names.

```
as-set:        [mandatory]  [single]     [primary/look-up key]
descr:         [mandatory]  [multiple]   [ ]
members:       [optional]   [multiple]   [ ]
mbrs-by-ref:   [optional]   [multiple]   [inverse key]
remarks:       [optional]   [multiple]   [ ]
tech-c:        [mandatory]  [multiple]   [inverse key]
admin-c:       [mandatory]  [multiple]   [inverse key]
notify:        [optional]   [multiple]   [inverse key]
mnt-by:        [mandatory]  [multiple]   [inverse key]
changed:       [mandatory]  [multiple]   [ ]
source:        [mandatory]  [single]     [ ]
```

Fig. 1.2.2 as-set template

1.2.3 aut-num

An object of the **aut-num** class is a database representation of an Autonomous System (AS), which is a group of IP networks operated by one or more network operators that has a single and clearly defined external routing policy.

Objects of this class are used to specify routing policies. The attributes of the **aut-num** class are shown in Figure 1.2.3. The value of the "aut-num:" attribute is the AS number of the AS described by this object. The "as-name:" attribute is a symbolic name (in RPSL name syntax) of the AS. The import, export and default routing policies of the AS are specified using the "import:", "export:" and "default:" attributes, respectively.

```
aut-num:        [mandatory]  [single]     [primary/look-up key]
as-name:        [mandatory]  [single]     [ ]
descr:          [mandatory]  [multiple]   [ ]
member-of:      [optional]   [multiple]   [inverse key]
import:         [optional]   [multiple]   [ ]
export:         [optional]   [multiple]   [ ]
default:        [optional]   [multiple]   [ ]
remarks:        [optional]   [multiple]   [ ]
admin-c:        [mandatory]  [multiple]   [inverse key]
tech-c:         [mandatory]  [multiple]   [inverse key]
cross-mnt:      [optional]   [multiple]   [inverse key]
cross-nfy:      [optional]   [multiple]   [inverse key]
notify:         [optional]   [multiple]   [inverse key]
mnt-lower:      [optional]   [multiple]   [inverse key]
mnt-routes:     [optional]   [multiple]   [inverse key]
```

```
mnt-by:         [mandatory]  [multiple]   [inverse key]
changed:        [mandatory]  [multiple]   [ ]
source:         [mandatory]  [single]     [ ]
```

Fig. 1.2.3 aut-num template

1.2.4 domain

The **domain**object represents Top Level Domain (TLD) and other domain registrations. It is also used for Reverse Delegations. The domain name is written in fully qualified format, without a trailing " . " . The template of this class is shown in Figure 1.2.4

```
domain:         [mandatory]  [single]     [primary/look-up key]
descr:          [mandatory]  [multiple]   [ ]
admin-c:        [mandatory]  [multiple]   [inverse key]
tech-c:         [mandatory]  [multiple]   [inverse key]
zone-c:         [mandatory]  [multiple]   [inverse key]
nserver:        [optional]   [multiple]   [inverse key]
sub-dom:        [optional]   [multiple]   [inverse key]
dom-net:        [optional]   [multiple]   [ ]
remarks:        [optional]   [multiple]   [ ]
notify:         [optional]   [multiple]   [inverse key]
mnt-by:         [optional]   [multiple]   [inverse key]
mnt-lower:      [optional]   [multiple]   [inverse key]
refer:          [optional]   [single]     [ ]
changed:        [mandatory]  [multiple]   [ ]
source:         [mandatory]  [single]     [ ]
```

Fig. 1.2.4 domain template

1.2.5 filter-set

A **filter-set** object defines a set of routes that are matched by its filter. The "filter-set:" attribute defines the name of the filter. It is an RPSL name that starts with "fltr-". The "filter:" attribute defines the set’s policy filter. A policy filter is a logical expression which when applied to a set of routes returns a subset of these routes. The template of this class is shown in Figure 1.2.5.

```
filter-set:     [mandatory]  [single]     [primary/look-up key]
descr:          [mandatory]  [multiple]   [ ]
filter:         [mandatory]  [single]     [ ]
remarks:        [optional]   [multiple]   [ ]
tech-c:         [mandatory]  [multiple]   [inverse key]
admin-c:        [mandatory]  [multiple]   [inverse key]
notify:         [optional]   [multiple]   [inverse key]
mnt-by:         [mandatory]  [multiple]   [inverse key]
changed:        [mandatory]  [multiple]   [ ]
source:         [mandatory]  [single]     [ ]
```

Fig. 1.2.5 filter-set template

1.2.6 inet6num

An **inet6num** object contains information on allocations and assignments of IPv6 address space. The template of this class is shown in Figure 1.2.6.

```
inet6num:       [mandatory]  [single]     [primary/look-up key]
netname:        [mandatory]  [single]     [lookup key]
descr:          [mandatory]  [multiple]   [ ]
country:        [mandatory]  [multiple]   [ ]
admin-c:        [mandatory]  [multiple]   [inverse key]
tech-c:         [mandatory]  [multiple]   [inverse key]
rev-srv:        [optional]   [multiple]   [inverse key]
status:         [generated]  [single]     [ ]
remarks:        [optional]   [multiple]   [ ]
notify:         [optional]   [multiple]   [inverse key]
mnt-by:         [mandatory]  [multiple]   [inverse key]
mnt-lower:      [optional]   [multiple]   [inverse key]
changed:        [mandatory]  [multiple]   [ ]
source:         [mandatory]  [single]     [ ]
```

Fig. 1.2.6 inet6num template

1.2.7 inetnum

An **inetnum** object contains information on allocations and assignments of IPv4 address space. The template of this class is shown in Figure 1.2.7.

```
inetnum:        [mandatory]  [single]     [primary/look-up key]
netname:        [mandatory]  [single]     [lookup key]
descr:          [mandatory]  [multiple]   [ ]
country:        [mandatory]  [multiple]   [ ]
admin-c:        [mandatory]  [multiple]   [inverse key]
tech-c:         [mandatory]  [multiple]   [inverse key]
rev-srv:        [optional]   [multiple]   [inverse key]
status:         [mandatory]  [single]     [ ]
remarks:        [optional]   [multiple]   [ ]
notify:         [optional]   [multiple]   [inverse key]
mnt-by:         [mandatory]  [multiple]   [inverse key]
mnt-lower:      [optional]   [multiple]   [inverse key]
mnt-routes:     [optional]   [multiple]   [inverse key]
changed:        [mandatory]  [multiple]   [ ]
source:         [mandatory]  [single]     [ ]
```

Fig. 1.2.7 inetnum template

1.2.8 inet-rtr

Routers are specified using the **inet-rtr** class. The attributes of the **inet-rtr** class are shown in Figure 1.2.8. The "inet-rtr:" attribute is a valid DNS name of the router described. Each "alias:" attribute, if present, is a canonical DNS name for the router. The "local-as:" attribute specifies the AS number of the AS that owns/operates this router. The template of this class is shown in Figure 1.2.8.

```
inet-rtr:       [mandatory]  [single]     [primary/look-up key]
descr:          [mandatory]  [multiple]   [ ]
alias:          [optional]   [multiple]   [ ]
local-as:       [mandatory]  [single]     [inverse key]
ifaddr:         [mandatory]  [multiple]   [lookup key]
peer:           [optional]   [multiple]   [ ]
```

```
member-of:      [optional]   [multiple]   [inverse key]
remarks:        [optional]   [multiple]   [ ]
admin-c:        [mandatory]  [multiple]   [inverse key]
tech-c:         [mandatory]  [multiple]   [inverse key]
notify:         [optional]   [multiple]   [inverse key]
mnt-by:         [mandatory]  [multiple]   [inverse key]
changed:        [mandatory]  [multiple]   [ ]
source:         [mandatory]  [single]     [ ]
```

Fig. 1.2.8 inet-rtr template

1.2.9 key-cert

A **key-cert** object is a database public key certificate that is stored on the server and may be used with a mntner object for authentication when performing updates. Currently only PGP keys are supported. The template of this class is shown in Figure 1.2.9.

```
key-cert:       [mandatory]  [single]     [primary/look-up key]
method:         [generated]  [single]     [ ]
owner:          [generated]  [multiple]   [ ]
fingerpr:       [generated]  [single]     [ ]
certif:         [mandatory]  [multiple]   [ ]
remarks:        [optional]   [multiple]   [ ]
notify:         [optional]   [multiple]   [inverse key]
mnt-by:         [mandatory]  [multiple]   [inverse key]
changed:        [mandatory]  [multiple]   [ ]
source:         [mandatory]  [single]     [ ]
```

Fig. 1.2.9 key-cert template

1.2.10 limerick

The **limerick** object represents a humorous poem that has five lines and the rhyme scheme "aabba". The template of this class is shown in Figure 1.2.10.

```
limerick:       [mandatory]  [single]     [primary/look-up key]
descr:          [optional]   [multiple]   [ ]
text:           [mandatory]  [multiple]   [ ]
admin-c:        [mandatory]  [multiple]   [inverse key]
author:         [mandatory]  [multiple]   [inverse key]
remarks:        [optional]   [multiple]   [ ]
notify:         [optional]   [multiple]   [inverse key]
mnt-by:         [mandatory]  [multiple]   [inverse key]
changed:        [mandatory]  [multiple]   [ ]
source:         [mandatory]  [single]     [ ]
```

Fig. 1.2.10 limerick template

1.2.11 mntner

Objects in the RIPE Database may be protected using **mntner** (pronounced "maintainer") objects. A **mntner** object specifies authentication information required to authorise creation, deletion or modification of the objects protected by the **mntner**. The template of this class is shown in Figure 1.2.11.

```
mntner:         [mandatory]  [single]     [primary/look-up key]
descr:          [mandatory]  [multiple]   [ ]
admin-c:        [mandatory]  [multiple]   [inverse key]
tech-c:         [optional]   [multiple]   [inverse key]
upd-to:         [mandatory]  [multiple]   [inverse key]
mnt-nfy:        [optional]   [multiple]   [inverse key]
auth:           [mandatory]  [multiple]   [ ]
remarks:        [optional]   [multiple]   [ ]
notify:         [optional]   [multiple]   [inverse key]
mnt-by:         [mandatory]  [multiple]   [inverse key]
auth-override:  [optional]   [single]     [ ]
referral-by:    [mandatory]  [single]     [inverse key]
changed:        [mandatory]  [multiple]   [ ]
source:         [mandatory]  [single]     [ ]
```

Fig. 1.2.11 mntner template

1.2.12 peering-set

A **peering-set** object defines a set of peerings that are listed in its "peering:" attributes. The "peering-set:" attribute defines the name of the set. It is an RPSL name that starts with "prng-". The template of this class is shown in Figure 1.2.12.

```
peering-set:    [mandatory]  [single]     [primary/look-up key]
descr:          [mandatory]  [multiple]   [ ]
peering:        [mandatory]  [multiple]   [ ]
remarks:        [optional]   [multiple]   [ ]
tech-c:         [mandatory]  [multiple]   [inverse key]
admin-c:        [mandatory]  [multiple]   [inverse key]
notify:         [optional]   [multiple]   [inverse key]
mnt-by:         [mandatory]  [multiple]   [inverse key]
changed:        [mandatory]  [multiple]   [ ]
source:         [mandatory]  [single]     [ ]
```

Fig. 1.2.12 peering-set template

1.2.13 person

A **person** object contains information about technical or administrative contact responsible for the object where it is referenced. Once the object is created, the value of the "person:" attribute cannot be changed. The template of this class is shown in Figure 1.2.13.

```
person:         [mandatory]  [single]     [lookup key]
address:        [mandatory]  [multiple]   [ ]
phone:          [mandatory]  [multiple]   [ ]
fax-no:         [optional]   [multiple]   [ ]
e-mail:         [optional]   [multiple]   [lookup key]
nic-hdl:        [mandatory]  [single]     [primary/look-up key]
remarks:        [optional]   [multiple]   [ ]
notify:         [optional]   [multiple]   [inverse key]
mnt-by:         [optional]   [multiple]   [inverse key]
changed:        [mandatory]  [multiple]   [ ]
source:         [mandatory]  [single]     [ ]
```

Fig. 1.2.13 person template

1.2.14 role

The **role** class is similar to the **person** class. However, instead of describing a human being, it describes a role performed by one or more human beings. Examples include help desks, network monitoring centres, system administrators, etc. A **role** object is particularly useful since often a person performing a role may change; however the role itself remains. The "nic-hdl:" attributes of the **person** and **role** classes share the same name space. Once the object is created, the value of the "role:" attribute cannot be changed. The template of this class is shown in Figure 1.2.14.

```
role:           [mandatory]  [single]     [lookup key]
address:        [mandatory]  [multiple]   [ ]
phone:          [optional]   [multiple]   [ ]
fax-no:         [optional]   [multiple]   [ ]
e-mail:         [mandatory]  [multiple]   [lookup key]
trouble:        [optional]   [multiple]   [ ]
admin-c:        [mandatory]  [multiple]   [inverse key]
tech-c:         [mandatory]  [multiple]   [inverse key]
nic-hdl:        [mandatory]  [single]     [primary/look-up key]
remarks:        [optional]   [multiple]   [ ]
notify:         [optional]   [multiple]   [inverse key]
mnt-by:         [optional]   [multiple]   [inverse key]
changed:        [mandatory]  [multiple]   [ ]
source:         [mandatory]  [single]     [ ]
```

Fig. 1.2.14 role template

1.2.15 route

Each interAS route (also referred to as an interdomain route) originated by an AS is specified using a **route** object. The "route:" attribute is the address prefix of the route and the "origin:" attribute is the AS number of the AS that originates the route into the interAS routing system. The "route:" and "origin:" attribute pair constitutes the primary key. The template of this class is shown in Figure 1.2.15.

```
route:          [mandatory]  [single]     [primary/look-up key]
descr:          [mandatory]  [multiple]   [ ]
origin:         [mandatory]  [single]     [primary/inverse key]
holes:          [optional]   [multiple]   [ ]
member-of:      [optional]   [multiple]   [inverse key]
inject:         [optional]   [multiple]   [ ]
aggr-mtd:       [optional]   [single]     [ ]
aggr-bndry:     [optional]   [single]     [ ]
export-comps:   [optional]   [single]     [ ]
components:     [optional]   [single]     [ ]
remarks:        [optional]   [multiple]   [ ]
cross-mnt:      [optional]   [multiple]   [inverse key]
cross-nfy:      [optional]   [multiple]   [inverse key]
notify:         [optional]   [multiple]   [inverse key]
mnt-lower:      [optional]   [multiple]   [inverse key]
mnt-routes:     [optional]   [multiple]   [inverse key]
mnt-by:         [mandatory]  [multiple]   [inverse key]
```

```
changed:        [mandatory]  [multiple]   [ ]
source:         [mandatory]  [single]     [ ]
```

Fig. 1.2.15 route template

1.2.16 route-set

A **route-set** object defines a set of routes that can be represented by **route** objects or by address prefixes. In the first case, the set is populated by means of the "mbrs-by-ref:" attribute, in the latter, the members of the set are explicitly listed in the "members:" attribute. The "members:" attribute is a list of address prefixes or other route-set names. Note that the **route-set** class is a set of route prefixes, not of database route objects. The template of this class is shown in Figure 1.2.16.

```
route-set:      [mandatory]  [single]     [primary/look-up key]
descr:          [mandatory]  [multiple]   [ ]
members:        [optional]   [multiple]   [ ]
mbrs-by-ref:    [optional]   [multiple]   [inverse key]
remarks:        [optional]   [multiple]   [ ]
tech-c:         [mandatory]  [multiple]   [inverse key]
admin-c:        [mandatory]  [multiple]   [inverse key]
notify:         [optional]   [multiple]   [inverse key]
mnt-by:         [mandatory]  [multiple]   [inverse key]
changed:        [mandatory]  [multiple]   [ ]
source:         [mandatory]  [single]     [ ]
```

Fig. 1.2.16 route-set template

1.2.17 rtr-set

A **rtr-set** object defines a set of routers. A set may be described by the "members:" attribute, which is a list of inet-rtr names, IPv4 addresses or other rtr-set names. A set may also be populated by means of the "mbrs-by-ref:" attribute, in which case it is represented by **inet-rtr** objects. The template of this class is shown in Figure 1.2.17.

```
rtr-set:        [mandatory]  [single]     [primary/look-up key]
descr:          [mandatory]  [multiple]   [ ]
members:        [optional]   [multiple]   [ ]
mbrs-by-ref:    [optional]   [multiple]   [inverse key]
remarks:        [optional]   [multiple]   [ ]
tech-c:         [mandatory]  [multiple]   [inverse key]
admin-c:        [mandatory]  [multiple]   [inverse key]
notify:         [optional]   [multiple]   [inverse key]
mnt-by:         [mandatory]  [multiple]   [inverse key]
changed:        [mandatory]  [multiple]   [ ]
source:         [mandatory]  [single]     [ ]
```

Fig. 1.2.17 rtr-set template

2.0 Queries in the RIPE Database

This section describes the functionality provided in version 3.0 of the RIPE Database to enable users to retrieve database objects. Querying the database is done using a client who uses the *whois* protocol to query and get the responses.

The newly incorporated mechanism to enable the whois server to automatically track query responses and limit the retrieval of contact information from the RIPE Database is also described in this section. The intention is to limit non-operational uses of the data (such as advertising) while permitting the operational uses of the database to be carried out.

There is a set of general rules about server responses:

- Output starting with the % sign is either a server response code or an informational message. A comment contains a white space after the % sign, while server messages start right after the % sign. Please see Appendix A2 "RIPE Database response codes and messages" for more information.
- An empty line ("\n\n") is an object delimiter.
- Two empty lines mean the end of a server response.
- When using the referral mechanism, the output of the referred server is passed to the client without modification. See section 2.7 "Referral mechanism for domains" for more information.

The general format of a query is:

[-query_flags [query argument]] <lookup-key>

2.1 Queries using primary and lookup keys

Normal queries are performed using primary and lookup keys as an argument to a query. These queries are presented in Table 2. Please refer to section 1.2 "Object types" for primary and lookup keys definition for a class.

2.2 IP address lookups

Probably, the most important service provided by the RIPE Database is to provide information about **IP networks** in the Internet. This information is stored in the database in the form of **inetnum** and **route** objects, among others.

The **inetnum**, **inet6num** and **route** objects store information about ranges of IP addresses.

The **route** objects use prefix notation to specify the range of IP addresses that they contain information about. This type of notation specifies ranges using two components: the prefix and its length.

For IPv4, the prefix is a 32-bit integer written in dotted quad notation and having the value of the lowest IP address in the range. The prefix length is an integer number in the range 0-32 (e.g. 193.0.0.0/22 specifies the range of 1024 IPv4 addresses starting with, and including, 193.0.0.0).

The **inetnum** objects represent an IPv4 address space in range notation where the range is explicitly specified as two 32-bit integers written in dotted quad notation separated by a dash ("-") (e.g. 193.0.0.0-193.0.3.255, which specifies the same range as above).

When dealing with IPv6 address ranges, only the standard IPv6 prefix notation is allowed (prefix length must be in the range 0-128 and the prefix is a 128-bit integer, written in hexadecimal groups of 2 bytes separated by colons and with the possible use of shorthand notation for strings of consecutive 0s).

When querying the database for information about a range of IP addresses, the user can use as search keys the following range notations:

- a prefix, which has the same meaning as above;
- an explicit range, also as above;
- a single IP number, which when used as an argument to a query is interpreted as a range of exactly 1 address.

These types of queries are presented in Table 3. The remainder of this section describes how a user can request different types of information to be returned, relative to a particular range of IP addresses.

Before going into details, it is useful to define three concepts frequently used in this type of queries and which are defined relative to the user-specified (reference) range:

- A *less* specific range is a range that contains the whole of the reference range and is bigger (contains more IP addresses) than the reference range.
- A *more* specific range is a range contained within the reference range and contains less IP addresses than the reference range.
- An *exact* match refers to a range that is identical to the reference range.

2.2.1 Default lookup for IP ranges in the RIPE Database

When no flags are specified and the query key sent to the whois server is a range of IP addresses (either IPv4 or IPv6, expressed as a single IP address, two IP addresses separated by a hyphen ("-") or a range of IP addresses in prefix notation), the RIPE whois server will try to find an *exact* match for that range.

If an *exact* match is found, it is returned. If not, a lookup for the smallest *less* specific range is performed and this is returned.

2.2.2 More and less specific queries

Sometimes the *exact* match is not the desired information. In that case there is a set of flags that modify the response of the whois server.

This set of 4 flags ("-M", "-m", "-L" and "-l") provides two generic types of queries known as *more* and *less* specific queries.

2.2.2.1 Less specific queries

These refer to queries triggered by the use of the "-l" and "-L" flags. These queries will return information about ranges of IP addresses that fully contain the user-supplied range and may contain more addresses.

The "-L" flag requests that the server return the *exact* match, if any, and all the objects containing information about IP ranges that are bigger than the user-supplied range and fully contain it.

The "-l" flag requests that the server NOT return the *exact* match but only the smallest of the IP ranges that is bigger than the user-supplied range and that fully contains it. This is usually referred to as the one level *less* specific range.

2.2.2.2 More specific queries

These refer to queries triggered by the use of the "-m" and "-M" flags. These queries will return information about ranges of IP addresses that are fully contained in the user-supplied range and contain fewer addresses.

The "-M" flag requests that instead of returning the *exact* match, the server should return all the sub-ranges completely contained within the user-provided range no matter what their size is.

The "-m" is the *more* specific equivalent of the "-l" flag. It requests that instead of returning the *exact* match, the server should return sub-ranges that are fully contained within the user-provided range. But instead of reporting all existing sub-ranges, it will only return the biggest ranges contained in the user range. These are usually called one level more specific.

2.3 Inverse queries

Inverse queries are performed on inverse keys as defined in the RIPE Database object templates. For a complete listing of these templates, please refer to section 1.2 "Object types". By issuing this type of query, the client requests all objects that reference the object with the key specified as a query argument to be returned by the Database.

One can also request an inverse query for several attributes (e.g. find out which objects reference a specific **mntner** object by "mnt-by:", "mnt-lower:" or "mnt-routes:" attributes). In such case, the query flag should be represented by a comma-separated list of attributes to be searched. No white space is allowed in the list.

Please refer to Table 4 for a complete list of supported inverse queries.

2.4 Query support for tools

Several query types were implemented in the RIPE Database to support various client tools. However, they can also be used by normal whois clients.

2.4.1 RAToolset support

The RAToolSet [6] is a suite of routing policy analysis tools created by the Information Sciences Institute at the University of Southern California. Some of the tools in this set access Routing Registry servers via a specialised whois interface in order to perform their work.

Support for some of the queries required has not been present in the RIPE whois server up to now. Version 3.0 of the RIPE whois server includes support for these query types. This section describes the additions to the RIPE whois user interface that allow it to support the RAToolset. The required queries are:

- Return the prefixes of all **route** objects with a specified origin.
- Return only the primary keys of the **route** objects, not full objects.
- Return the prefixes of all **route** objects referenced in a given **route-set**.
- Return only the primary keys of the **route** objects, not full objects.
- Return all the members (**aut-num** or **as-set** object) of a specified **as-set**.

  Return only the "members:" attributes, not the full object.
- Optionally, include support for expansion of the previous query, if the returned value contains references to as-sets, so that the end result contains only a list of **aut-num** objects. The RIPE Database does not support this and it is up to the client to perform the expansion. The RAToolset currently does the expansion itself.
- Return **route** objects that exactly match a specified prefix.
- Return **route** objects that exactly match a specified prefix (as above), but return only the "route:" attributes.
- The new flags introduced in RIPE whois server v3.0 to support the above-mentioned functionality are presented in Tables 3 and 5.

2.4.2 Persistent connections and keeping state

To make more efficient use of the server for batched queries, a client can request a persistent connection, one that will not be closed by the server after the reply to the whois query has been sent to the client. This avoids having to set up and tear down a new TCP connection for every query.

The client can request this by sending the "-k" flag.

During this type of connection the server will keep state for some internal variables.

Currently state is only kept for the list of sources to query. To exit a persistent connection, send the "-k" flag as the only query argument to the server or an empty query ("\n").

2.5 Getting all the members of set objects

In RPSL[3] there are two ways in which an object can be a member of a set object.

The first one is by listing objects in a "members:" attribute in the set object. This is the kind of member relationship present in "Representation of IP Routing Policies in a Routing Registry"[4] (e.g. "as-list:" attribute in **as-macro** objects).

The other way of specifying a membership relation is through the use of the "member-of:" attribute. This attribute can be used in the **route**, **aut-num** and **inet-rtr** classes. The value of the "member-of:" attribute identifies a set object that this object wants to be a member of.

However, specifying member-of is not enough. The set object must also have a "mbrs-by-ref:" attribute listing the maintainer of the object wanting to be a member of the set. That is, the set owner must validate the membership claim of an object with a "member-of:" attribute, and it does that by matching the mnt-by line of the object with one of the maintainers in the "mbrs-by-ref:" attribute of the set object.

2.6 More/less specific lookups for in-addr.arpa and ip6.int domains

Version 3.0 of the RIPE Database supports the use of the "-m", "-M", "-l" and "-L" flags for lookups on reverse delegation domains. To enable this behaviour, use the "-d" flag in your whois query.

The use of these flags is similar to that described for IP numbers except that lookup keys have to be on 8-bit boundaries for IPv4 lookups and 4-bit boundaries for IPv6 lookups.

2.7 Referral mechanism for domains

The referral mechanism provides a way for administrators of domain registries to instruct the whois server to reply to the user by fetching data from the domain registry database rather than from local data. Its implementation consists of two different parts:

1. Domain name stripping. When no matching **domain** object is found in the database with the name specified in the query, the domain name is stripped towards higher-level domains (xxx.yyy.zzz becoming yyy.zzz) and the lookup is repeated until a **domain** object is found or the search string becomes empty.

   For backwards compatibility, if the **domain** object found by domain stripping does not contain a "refer:" attribute, then it is considered that no objects are found and an appropriate message is displayed. Please see Appendix A2 "RIPE Database response codes and messages" for more information.

2. The "refer:" attribute. The "refer:" attribute gives domain name administrators the possibility to point the whois server to an authoritative server for information about the domain name. This attribute specifies the hostname, port and referral type that the server should use to redirect the query. Please see Appendix A1 "Object attributes" for the syntax of this attribute.

   If a query to the whois server results in the retrieval of a local object that contains a "refer:" attribute, the server will connect to the remote server and issue a whois query for the requested domain name. Whatever is returned is then sent to the user.

   This mechanism is disabled by including the "-R" flag in the original query.

2.8 Access control for queries

This section describes the mechanism for controlling how many database objects a particular whois client can retrieve from the whois server. The objective is to control and prevent abusive use of the Database done by systematic queries for contact information potentially used for non-agreed purposes (e.g. spam).

Therefore, the access control system is limiting only the amount of contact information (number of **person** and **role** objects) that can be retrieved in a certain amount of time. No limitations exist on the number of other objects.

However, one should keep in mind that many lookups for object types, other than **person** and **role**, also return contact information by default. To avoid being blocked in such cases, it is advisable to use the "-r" query flag.

Every time a **person** or **role** object is retrieved, a counter is decreased. When it reaches zero, the query execution is aborted and the connection is terminated, displaying an error message to the client (see "Access errors" in Appendix A2 "RIPE Database response codes and messages"), also a counter of exceeds (denials) is incremented. The Database may limit the number of denials, after which the host is permanently blocked from access to the RIPE Database. The Database Administration may also block a client manually in case any abusive behaviour is discovered.

The counter recovers in time. The host can resume querying contact information after the counter has recovered so that the host gets a non-zero limit for contacts next time.

Accounting is based on the IP address of a client.

The RIPE Database server provides a facility for proxy clients (for example, web servers running cgi interfaces to the RIPE Database) that allows accounting to be based not on the IP address of the proxy, but rather of the clients that use this proxy to query the RIPE Database. The "-V" flag supports this feature. In such case, the format of the query is as follows:

-V <version>,<ipv4-address>

where

<version> is a client tag that usually represents the version of the proxyss client software;
<ipv4-address> is the IPv4 address of the proxy.

Note that the proxy?s IP address should be registered in the access control list of the RIPE Database. Please contact RIPE Database Administration if you need this option.

2.9 Other server features

The RIPE Database version 3.0 server supports the retrieval of certain information about itself and the data sets served using a ?-q? query flag.

The "-q " flag takes arguments that make the server reply with information that is not extracted from the Databases it serves but rather about the system setup. This flag can currently take two arguments:

- version (usage: -q version). Display version information for the server software.
- sources (usage: -q sources). List all available sources. The format of the output is:
  < source>:<NRTM_protocol_version_#>:<mirroring>:<first>-<last>
  Where
  <source> - is the string that identifies the Database (e.g. RIPE);
  <NRTM_protocol_version_#> identifies the version of the mirroring protocol.
  <mirroring> can take one of the following values:
  -- Y/N/X - can mirror/cannot mirror/obscured
  <first> is the lowest serial number available
  <last> is the most recent serial number available.

The following semantics apply:

Y:<first>-<last> mirroring allowed, serials from range first-last available.
N:<first>-<last> mirroring not allowed for administrative reasons. Ask Database Administration for permission.
N:0-<last> mirroring physically not possible (e.g. static snapshot of serial last).
X:<message> no mirroring allowed. An explanation is given in the <message>.

Tables 2 to 7 contain all query types supported by the RIPE Database:

Table 2 Queries using primary and lookup keys

Flag	Lookup Key(s)	Effect
	<ip-lookup> (IPv4 address prefix, range or single address)	Returns **inetnum**, **route** objects with exact match on the specified key. If the exact match does not exist, the objects with the smallest less specific match are returned. When a single address is specified, an **inet-rtr** object whose "ifaddr:" attribute matches the query argument is also returned.
	<ip-lookup> (IPv6 address or IPv6 prefix)	Returns an **inet6num** object with exact match on a specified key. If the exact match does not exist, the object with the smallest less specific match is returned.
	<as-number>	Returns an **aut-num** object whose "aut-num:" attribute matches the query argument and an **as-block** object with the range containing the **aut-num** object, if it exists.
	<as-number> - <as-number> (range of <as-number> separated by "-")	Returns an **as-block** object whose primary key defines a range that matches or fully contains the range specified in the query argument.
	<domain-name>	Returns **domain** and **inet-rtr** objects whose primary keys match the query argument. For domains, a referral query may be performed. In such case the actual query is performed by the referred server and the results are transparently passed to the client. See section 2.7 "Referral mechanism for domains" for more information.
	<person-name>	Returns all **person** and **role** objects whose "person:" or "role:" attributes contain the name specified in the query argument.
	<set-name>	Returns a set whose primary key matches the query argument.
	<nic-handle>	Returns a **person** or **role** object whose "nic-hdl:" attribute matches the query argument.
	<mntner-name>	Returns a **mntner** object whose primary key matches the query argument.

Table 3 IP address lookups

Flag	Lookup Key(s)	Effect
-l	<ip-lookup>	Returns first level less specific **inetnum**, **inet6num** or **route** objects, excluding exact matches.
-L	<ip-lookup>	Returns all level less specific **inetnum**, **inet6num** or **route** objects, including exact matches.
-m	<ip-lookup>	Returns first level more specific **inetnum**, **inet6num** or **route** objects, excluding exact matches.
-M	<ip-lookup>	Returns all level more specific **inetnum**, **inet6num** or **route** objects, excluding exact matches.
-x	<ip-lookup>	Requests that only an exact match on a prefix be performed. If no exact match is found, no objects are returned.
-d	<ip-lookup>	Enables use of the "-m", "-M", "-l" and "-L" flags for lookups on reverse delegation domains.

Table 4 Inverse queries

Flag (alternative form)	**Lookup Key(s)**	**Effect**
-i ac (-i admin-c)	<nic-handle> or <person-name>	Returns all objects whose "admin-c:" attributes match the query argument.
-i ah (-i author)	<nic-handle> or <person-name>	Returns all **limerick** objects whose "author-c:" attribute matches the query argument.
-i pn	<nic-handle> or <person-name>	Returns all objects whose "admin-c:", "tech-c:", "zone-c:", "author:" or "cross-nfy:" attribute matches the query argument.
-i ct (-I cross-mnt)	<mntner-name>	Returns all **route** and **aut-num** objects whose "cross-mnt:" attributes matches the query argument.
-i cn (-i cross-nfy)	<nic-handle> or <person-name>	Returns all **route** and **aut-num** objects whose ?cross-nfy:? attribute matches the query argument.
-i la (-I local-as)	<as-number>	Returns all **inet-rtr** objects whose "local-as:" attribute matches the query argument.
-i mr (-i mbrs-by-ref)	<mntner-name>	Returns all set objects (**as-set**, **route-set** and **rtr-set**) whose "mbrs-by-ref:" attribute matches the query argument.
-i mo (-i member-of)	<set-name>	Returns all objects whose "member-of:" attribute matches the query argument and their membership claim is validated by the "mbrs-by-ref:" attribute of the set. Absence of the "mbrs-by-ref:" attribute means that the membership is only defined by the "members:" attribute of the set.
-i mb (-i mnt-by)	<mntner-name>	Returns all objects whose "mnt-by:" attribute matches the query argument.

-i ml (-i mnt-lower)	<mntner-name>	Returns all objects whose "mnt-lower:" attribute matches the query argument.
-i mn (-i mnt-nfy)	<e-mail>	Returns all **mntner** objects whose "mnt-nfy:" attribute matches the query argument.
-i mu (-i mnt-routes)	<mntner-name>	Returns all **aut-num**, **inetnum** and **route** objects whose "mnt-routes:" attribute matches the query argument.
-i ny (-i notify)	<e-mail>	Returns all objects whose "notify:" attribute matches the query argument.
-i ns (-I nserver)	<domain-name> or <ip-lookup> (IPv4/IPv6 address)	Returns all **domain** objects whose "nserver:" attribute matches the query argument.
-i or (-i origin)	<as-name>	Returns all **route** objects whose "origin:" attribute matches the query argument.
-i rb (-i referral-by)	<mntner-name>	Returns all **mntner** objects whose "referral-by:" attribute matches the query argument.
-i rz (-i rev-srv)	<domain-name> or <ip-lookup> (IPv4/IPv6 address)	Returns all **inetnum** and **inet6num** objects whose "rev-srv:" attribute matches the query argument.
-i sd (-i sub-dom)	<domain-name>	Returns all **domain** objects whose "sub-dom:" attribute matches the query argument.
-i tc (-i tech-c)	<nic-handle> or <person-name>	Returns all objects whose "tech-c:" attribute matches the query argument.
-i dt (-i upd-to)	<e-mail>	Returns all **mntner** objects whose "upd-to:" attribute matches the query argument.
-i zc (-i zone-c)	<nic-handle> or <person-name>	Returns all objects whose "zone-c:" attribute matches the query argument.

Table 5 Query support for tools

Flag	Lookup Key(s)	Effect
-F		Produces output using shorthand notation for attribute names. Produces slower responses.
-K		Requests that only the primary keys of an object be returned. The exceptions are set objects, where the "members:" attributes will also be returned. This flag does not apply to **person** and **role** objects.
-k	(optional normal query)	Requests a persistent connection. After returning the result, the connection will not be closed by the server and a client may issue multiple queries on the same connection. Note that the server implements a "stop-and-wait" protocol, where no next query can be sent before receiving a reply for the previous one. Use RIPE whois client to be able to send queries in batch mode. Except the first "-k query", "-k" without an argument closes the persistent connection.
-g	(mirroring request)	Request a NRTM stream from the server. See section 4.0 "Mirroring of the RIPE Database" for more information.

Table 6 Miscellaneous queries>

Flag	Argument	Effect
-R		Switches off use of the referral mechanism for domain lookups, so that the database returns an object in the RIPE database with the exact match with the lookup argument, rather than doing a referral lookup.
-r		Switches off recursion for contact information after retrieving the objects that match the lookup key.
-T	(comma separated list of object types, no white space is allowed)	Restricts the types of objects to lookup in the query.
-a		Specifies that the server should perform lookups in all available sources. See also "-q sources" query.
-s	(comma separated list of sources, no white space is allowed)	Specifies which sources and in which order are to be looked up when performing a query.

Table 7 Informational queries

Flag	Argument	Effect
-q	sources	Returns the current set of sources along with the information required for mirroring. See section 2.9 "Other server features" for more information.
-q	version	Displays the current version of the server.
-t	<object-type>	Requests a template for the specified object type.
-V<client-tag>		Sends information about the client to the server.
-v	<object-type>	Requests a verbose template for the specified object type.

The following notations are used in this table:
<object-type> means full or abbreviated name of a specific class;
<client-tag> is a string without a white space that usually bears the name of the client's software.

Please refer to section 2.8 "Access control for queries" for more information about using this flag for proxy clients.

Other notations are explained in Table A1.

3.0 Updates in the RIPE Database

To *create* a new object, *update* an existing one, or *delete* an object from the RIPE Database, an update message should be sent to the Database for processing.

Two types of submissions are possible:

- Updates via e-mail which is the main public interface, and on-line;
- Updates via "networkupdate" interface. This method is internal, and is only used from authorised nodes; typically, nodes belonging to the internal office LAN of the RIPE NCC.

3.1 Format of an update message

If sending the message via e-mail, the message may also be a MIME encoded message. The update is normally in plain text. In the former case, each valid MIME part is treated as a separate message. The update message may contain more than one object. Please see section 3.3.1 "MIME support" for more information.

In an update message an object should be textually represented as a list of attribute-value pairs. Each attribute-value pair is written on a separate line. The attribute name starts at column 0, followed by character ":" and followed by the value of the attribute. The attribute, which has the same name as the object's class, should be specified first. An attribute's value can be split over multiple lines, by having a space, a tab or a plus ("+") character as the first character of the continuation lines. The character "+" for line continuation allows attribute values to contain blank lines. More spaces may optionally be used after the continuation character to increase readability. The order of attribute-value pairs is significant. No empty attributes (an attribute with empty value) are allowed, unless the type of an attribute value is <free-form>. Object definition starts with the class attribute and ends with the first blank line ("\n\n"). No blank lines are allowed within the object.

An object's definition may contain comments. A comment can be anywhere in an object's definition, it starts at the first "#" character on a line and ends at the first end-of-line character. A comment cannot start at column 0. White space characters can be used to improve readability.

For more information on the format of the objects, please see section 1.0 "Database objects and attributes".

Each part of the message that is not recognised as a database object is ignored and the error message is issued in the acknowledgement.

3.2 Creating, modifying and deleting an object

To create, update or delete objects, a message containing the objects should be prepared following object templates and sent to the database for processing. One message may contain several objects, each of them may require different operation: creation, modification or deletion.

3.2.1 Object processing

As a general rule, the order of objects in the message is not changed. The database processes objects one by one, so it is the user's responsibility to ensure that all references can be resolved. The only exception is related to using "AUTO" NIC handles for automatic assignment of a value for the "nic-hdl:" attribute in the **person** or **role** objects. In such cases, objects with "AUTO" NIC handles are processed before any other object that can reference them is processed.

While processing an object, the server performs the following checks:

- Verifies that the syntax of an object is correct.
- Verifies that the object passes authorisation checks.
- Verifies that all references can be resolved without conflicts.
- Verifies that the operation does not compromise referential integrity. This is performed for the deletion of an object to ensure that it is not referenced from any other object in the RIPE Database.
- Verifies that the requested NIC handle is not in use and can be allocated. This is performed only for the creation of **person** or **role** objects that request a particular NIC handle.

If all checks were passed successfully, the server creates the object in the RIPE Database. If one of these steps fails, the operation fails for the object. This is reflected in the acknowledgement message and, under certain conditions, in notification messages.

After the database finishes processing the whole message, an acknowledgement message is composed and sent to the sender of the original update as specified in the "Reply-to:" field or "From:" field in the update message, if "Reply-to:" was not specified.

Also in some cases, notification messages are sent. Please see section 3.5.2 "Notifications" for more information.

3.2.2 Creating a new object

If an object with the same primary keys as the object in the update message does not exist in the database, the assumed operation is object creation. For **person** and **role** objects creation, one can use "AUTO NIC handles", requesting the server to automatically assign a NIC handle. In such case, the value of the "nic-hdl:" attribute should be:

nic-hdl: AUTO-1[<initials>]

If the <initials> (2 to 4 characters) are specified, then the server will try to use them for constructing the NIC handle. If the <initials> are omitted, the server will guess the initials from the "person:" or "role:" attribute.

3.2.3 Modifying an existing object

If an object with the same primary keys as the object in the update message already exists in the database, the assumed operation is object modification. The server compares the old and new versions of the object and reports a no-operation error if they are identical. When comparing the versions, white space characters are not considered.

3.2.4 Deleting an object

The object deletion is requested by adding a special attribute "delete:" to the object:

delete: <comment>

The deletion is only accepted if the object in the message is exactly the same as the one in the database about to be deleted. When comparing the versions, white space characters are not considered. Also, operation cannot succeed if the object is referenced from any other objects in the Database.

3.3 E-mail updates

A mail message containing an update should be sent to the e-mail address of the RIPE Database robot: auto-dbm@ripe.net. After processing the message, an acknowledgement is sent back to the user, containing information about which object updates succeeded and which failed, along with the reason for failure. In some cases notification messages are sent to relevant users. Please see section 3.5.2 "Notifications" for more information.

3.3.1 MIME support

The Database software supports MIME. This feature is meant mainly to make the cryptographic signing of the message easier when using mail agents that place the signature in a separate MIME part, not included in the body of the message.

It also allows the definition of scopes of authorisation within the message (e.g. parts where different passwords apply) and nested signing of messages which may be necessary under some conditions when updating objects whose authorisation must be derived from more than one party.

The following rules apply when submitting updates using MIME encapsulation.

A. The software recognises the following headers and appropriate actions are taken:

- multipart/signed
- multipart/alternative
- multipart/mixed
- multipart/unknown
- application/pgp-signature
- text/plain

All other content-types are treated as text/plain.

B. Each MIME part is treated as a separate message with the following implications:

- Authorisation information is valid only within a single part, except for MAIL-FROM type, which is valid across the entire message.
- AUTO NIC handle assignment is made only within a single part (see next section 3.3.2 "PGP support").

3.3.2 PGP support

The Database supports PGP-signed messages. The following rules apply when submitting updates using this authorisation scheme.

When using MIME encapsulation a PGP-signed portion of an update message should be submitted using multipart/signed composite type. In this case the first body part contains the update message (which may also be a MIME encapsulated message), and the second body contains a PGP-signature encapsulated with application/pgp-signature MIME discrete type.

Regarding AUTO NIC handle assignment, the PGP-signed portion is treated as a separate message.

If one of the signatures fails in a nested signed portion, the whole portion is rejected.

3.3.3 Subject line processing

The three keywords valid in a subject line of an update message are **NEW**, **HELP** and

HOWTO. Use NEW keyword by itself if you want the database to only accept new objects. HOWTO and HELP keywords can be used to get a help text that contains information about how to query and update the database (in this case the body of the message is ignored). If there is more than one keyword or there is a non-keyword in the subject line, then all the subject line is ignored.

3.4 Updates using *networkupdate* utility

No MIME encapsulation is possible when using *networkupdate*.
No PGP authentication is possible when using *networkupdate*.

3.5 Acknowledgements and Notifications

3.5.1 Acknowledgements

Processing of every valid object in the submission message (that is every object in plain text parts, supported MIME parts and/or valid PGP signed portions) is reflected in the acknowledgement message. This message contains information about whether the creation, modification or deletion has been successful or not. When there is a MIME part with an unsupported type in the incoming message, a warning will be added to the acknowledgement, saying that that MIME part is ignored. The acknowledgement message is sent back to the sender's e-mail address ("Reply-To:" or "From:", if the former is not specified).

The acknowledgement message starts with the header of the original update message as a quotation followed by the results of the updates performed for every valid object of the message.

The format of a successful operation report is:

<operation_name> OK: [*<object_type>*] *<object_key>*

where

<operation_name> may be New, Update, Delete, depending on the operation performed.
<object_type> is the class of the object that was processed.
<object_key> is the primary key of the object.

For example:

Update OK: [person] FOO12770-RIPE

Failed updates are reported as:

<operation_name> FAILED: [*<object_type>*] *<object_key>*

followed by the text of the object with more detailed explanation of what caused the failure.

There are several reasons why the operation may fail for the object. They are:

- syntax error
  The submitted object is syntactically incorrect. Please consult Appendix A1 for the description of attributes.

- referential integrity violation
  If an object references objects (by means of "admin-c:", "tech-c:", "zone-c:", "mnt-by:", etc. attributes) that do not exist in the database, creation or modification operations will fail. For the deletions, it is not allowed to delete an object that is referenced from any other object.

- authorisation failure
  When authorisation checks fail, the operation fails. Please refer to section 3.6 "Data Protection" for more information.

3.5.2 Notifications

There are three types of notifications:

- normal notifications
- forward messages
- cross notifications.

3.5.2.1 Normal notifications

Normal notifications are sent:

- when an object with a "notify:" attribute is updated. The "notify:" attribute of the old version of the object is used if the object was already in the database, and the "notify:" attribute of the new object is used if the object is a new one.

- when an object that is maintained (that is, it has a "mnt-by:" attribute) is updated, and the maintainer(s) have "mnt-nfy:" attributes. The e-mail boxes mentioned in the "mnt-nfy:" attributes of the relevant maintainers must be used.

- when an **inetnum**, **route** or a **domain** object is created in a space protected by a less specific object (by "mnt-lower:" attribute of the object).

3.5.2.2 Forward messages

When an update fails to pass authorisation checks, that object in the update message is forwarded to the e-mail address specified in the "upd-to:" attribute of the relevant maintainer(s).

3.5.2.3 Cross notifications

Cross notifications are sent:

- when a new or deleted **route** object overlaps with another **route** object. A notification will be sent to the sender of the **route** object.

- when a new or deleted **route** object overlaps with another **route** object that has a "cross-nfy:" or "cross-mnt:" attribute. Notification must be sent to the "mnt-nfy:" attributes of the maintainers mentioned in the "cross-mnt:" attribute and to the "e-mail:" attributes of the **person** and **role** objects whose NIC handle is mentioned in the "cross-nfy:" attribute of the overlapping **route** objects.

- when a new or deleted **route** object overlaps with another **route** object whose "origin" is an **aut-num** object that has "cross-nfy:" or "cross-mnt:" attributes. Notification must be sent to the "mnt-nfy:" attributes of the maintainers mentioned in the "cross-mnt:" attribute and to the "e-mail:" attributes of the **person** or **role** objects whose NIC handle is mentioned in the "cross-nfy:" attribute of the **aut-num** object.

3.6 Data protection

The RIPE Database provides mechanisms to control who can make changes in the database and what changes they can make. The distinction of "who" vs. "what" separates authentication from authorisation.

- Authentication is the means to determine who is attempting to make a change.
- Authorisation is the determination of whether a transaction passing a specific authentication check is allowed to perform a given operation.

Different portions of the database require different levels of protection. Some applications require authentication based on strong encryption. In other cases strong encryption may not be necessary or may not be legally available. For this reason, multiple authentication methods are supported by the server.

3.6.1 Authorisation model

The **mntner** objects serve as a container to hold authentication filters. A reference to a maintainer within another object defines authorisation to perform operations on the object or on a set of related objects. Such reference is provided by means of the "mnt-by:", "mnt-lower:", "mnt-routes:" and "mbrs-by-ref:" attributes.

The maintainer contains one or more "auth:" attributes. Each "auth:" attribute begins with a keyword identifying the authentication method followed by the authentication information needed to enforce that method.

Authentication methods currently supported include the following:

Method	Description
NONE	No authorisation checks are performed.
MAIL-FROM	This is a very weak authentication check and is discouraged. The authentication information is a regular expression over ASCII characters. The maintainer is authenticated if the "From:" field in RFC 822 mail headers are matched by this regular expression. Since mail header forgery is quite easy, this is a very weak form of authentication.
CRYPT-PW	This is a relatively weak form of authentication. The authentication information is a fixed encrypted password in UNIX crypt format. The maintainer is authenticated if the transaction contains the clear text password of the maintainer. Since the password is in clear text in transactions, it can be captured by snooping. Since the encrypted form of the password is exposed, it is subject to password guessing attacks.
PGPKEY	Strong form of authentication. The authentication information is a signature identity pointing to a public key certificate, which is stored in a separate object. The maintainer is authenticated if the transaction is signed by the corresponding private key. The RIPE NCC does not guarantee that a key belongs to any specific entity; it is not a certificate authority. Anyone can supply any public keys with any ownership information to the database and these keys can be used to protect other objects by checking that the update comes from someone who knows the corresponding secret key.

3.6.2 Protection of individual objects

Individual objects can be protected with a **mntner** object. The mntner is referenced by the "mnt-by:" attribute in the object. The attribute type is multiple, so several mntners can protect the object.

Only those mntners referenced by the "mnt-by:" attributes are authorised to modify or delete the object. Note that authentication checks are OR-ed, so if at least one mntner is authenticated, the operation is authorised. That means that object protection is as weak as the weakest authentication method used in the mntners referenced by the object.

When the "mnt-by:" attribute is added to an object for the first time (as part of object creation or modification), the operation should pass authentication checks for the mntner(s) referenced by this attribute.

3.6.3 Protection of aut-num object space

Protection of **aut-num** object space is done using an **as-block** class. The mntner that authorises the creation of more specific **as-block** objects or **aut-num** objects is specified by the "mnt-lower:" attribute of the **as-block** object. When no "mnt-lower:" attribute is specified, the ?mnt-by:? attribute is used.

3.6.4 Protection of address space

Address space allocations and assignments are represented by the**inetnum** and **inet6num** objects. The "mnt-lower:" attribute is used to reference a mntner that authorises the creation of more specific **inetnum** or **inet6num** objects. When no "mnt-lower:" attribute is specified, the address space is unprotected.

3.6.5 Protection of route object space

The **route** object creation must satisfy several authentication criteria. It must match the authentication specified in the **aut-num** and the authentication specified in either a **route** object or, if no applicable **route** object is found, then an **inetnum**. Finally the creation must be authorised by the maintainer of the **route** object itself referenced by the "mnt-by:" attribute of the object.

When checking for prefix authorisation, an exact **route** object prefix match is checked for first. If there is no exact match, then a longest prefix match that is less specific than the prefix is searched for. If the route prefix search fails, then a search is performed for an **inetnum** object that exactly matches the prefix or for the most specific **inetnum** object that is less specific than the **route** object submission. The **aut-num** object used for authentication checks is referenced by the "origin:" attribute of the **route** object.

A **route** object must pass authorisation from both the referenced **aut-num** object and the **route** or **inetnum** object. Authorisation shall be tested using the maintainer(s) referenced in the "mnt-routes:" attribute(s) first. If that check fails, the "mnt-lower:" attributes are checked. If that check fails, the "mnt-by:" attributes are used for the authorisation check.

3.6.6 Protection of objects with hierarchical names

Many RPSL objects do not have a natural hierarchy of their own but allow hierarchical names. Some examples are the object types **as-set** and **route-set**. An **as-set** may have a name corresponding to no naming hierarchy such as "AS-Foo" or it may have a hierarchical name of the form "AS1:AS-BAR".

When a hierarchical name is not used, authorisation for objects such as **as-set** and **route-set** correspond to the rules for individual objects described in section 3.6.2 "Protection of individual objects".

If hierarchical names are used, then the addition of an object must be authorised by the **aut-num** whose key is named by everything to the left of the rightmost colon in the name of the object being added.

Authorisation is determined by first using the "mnt-lower:" maintainer reference or, if absent, by using the "mnt-by:" reference.

3.6.7 Protection of domain object space

Protection of the **domain** object space is done with the "mnt-lower:" attribute. When used in the **domain** object, this attribute references the mntner that authorises the creation of **domain** objects directly below in the domain tree registered in the RIPE Database. For example, a top-level **domain** object (ccTLD) protects the creation of the second-level **domain** objects, third-level **domain** objects if there is no corresponding second-level **domain** object, etc. Note that the top-level **domain** object (ccTLD or gTLD) cannot be created in the RIPE Database by users, only by the Database administrator.

3.6.8 Protecting membership of a set

When membership of a set is specified through the use of the "member-of:" attribute, the server checks the validity of the membership when creating or updating an object-member. This "member-of:" attribute can be used in the **route**, **aut-num** and **inet-rtr** classes. The value of the "member-of:" attribute identifies a set object that this object wants to be a member of.

However, specifying "member-of:" is not enough. The set object must also have a "mbrs-by-ref:" attribute listing the maintainer of the object wanting to be a member of the set. That is, the set owner must validate the membership claim of an object with a "member-of:" attribute, and it does that by matching the mnt-by line of the object with one of the maintainers in the "mbrs-by-ref:" attribute of the set object. If the claim is not valid at the time when the server creates or modifies an object-member (**route**, **aut-num** or **inet-rtr**), the operation fails. If the "mbrs-by-ref:" attribute is missing, the set is defined explicitly by the "members:" attribute.

4.0 Mirroring of the RIPE Database

Near real time mirroring (NRTM) is a mechanism that enables whois servers, other than the primary for a given database, to have an up to date copy of all the data in the main server by getting modifications as they are processed by the main server.

Periodically (defined by configuration, usually around every 15 minutes) the remote server connects to the main server and requests all the modifications that have been processed since the last connection. When all the data is retrieved the connection is closed, until it is time for a new connection.

In RIPE Database version 3.0, the NRTM server listens on a port that is different from the whois port (43).

The RIPE whois server generates a sequence number every time it processes an update in the database. For the purpose of generating this serial numbers, the server describes all modifications to the database in terms of two atomic operations: *deletion* and *addition*.

The old (v2.x) database supported a NRTM protocol (Version 1) when a modification of an existing object was considered to be a deletion followed by a creation of a new object with the modified data. The two operations for an object modification are not necessary (although still supported) in v3.0 database as an update is considered as an atomic operation. This behaviour is defined as Version 2 of the NRTM protocol.

When sending data to a mirror server, the main server will send one of two strings (ADD or DELETE) followed by two newline characters and the corresponding object (either the object as it was before deletion or the object as it should be added or modified). If the "ADD" string is followed by an object already existing in the database, then this operation is considered as a modification.

Near real time mirroring is requested with the "-g" flag. The arguments to this flag are:

-g < *source*> :*<NRTM_Protocol_version_#>*:*<first>*-*<last>*

where

- <source> is the string that identifies the Database (e.g. RIPE).
- *<NRTM_protocol_version_#>* a version of the mirroring protocol that the <source> supports (2 for RIPE).
- <first> is the lowest serial number requested.
- <last> is the most recent serial number requested or the keyword LAST that tells the main server to send all updates up to the most recent one available at the time of the query. Note that the server will never yield the last object processed, or display that it is available for mirroring. This is done to protect secondary servers in case the last update causes database corruption and server crash.

A client may request a persistent connection by including the "-k" flag with a mirroring request ("-g" query). In this case, the last argument is ignored and the new objects are supplied by the server as soon as they are processed. The client is responsible for closing the connection.

The "-q sources" flag may be used with the mirror server to retrieve information regarding available mirroring possibilities. Please see section 2.9 "Other server features" for more details.

Note that the server never returns the latest serial when serving a mirroring request. So, if the latest serial causes a server crash and database corruption, it never propagates to the secondary servers. The latest serial is not displayed with the "-q sources" query as well.

At the beginning of the data stream, the main server will send the following string:

%START Version: *NRTM_Protocol_version_# source first-last*

For example:

```
%START Version: 2 RIPE:1539595-1539597
```

After the last piece of data is sent to the mirror server, the main server will send the string:

```
%END source
```

to signal the end of transmission.

For example:

```
%END RIPE
```

Appendices

A1. Object attributes

Shown below are the syntax definitions of the object attributes supported by the RIPE Database.

The value of an attribute has a type. Some of the most commonly used types are shown in Table A1. Others are explained in the descriptions of the attributes.

Table A1. Commonly used attribute types

Type	Description
<quad>	<xdigit>.){1,4}
<dlabel>	Domain name label as specified in RFC 1034 [7]. The total length should not exceed 63 characters (octets) <alnum>(-<alnum>+)+
<action>	Please see RFC 2622 [1]
<address-prefix>	An address prefix is represented as an IPv4 address followed by the character slash "/" followed by an integer in the range from 0 to 32. The following are valid address prefixes: 128.9.128.5/32, 128.9.0.0/16, 0.0.0.0/0; and the following address prefixes are invalid: 0/0, 128.9/16 since 0 or 128.9 are not strings containing four integers. <ipv4-address>/<integer>
<address-prefix-range>	An address prefix range is an address prefix followed by an optional range operator. Please see RFC-2622 [1]
<as-expression>	Please see RFC 2622 [1]
<as-number>	An "AS" string followed by an integer in the range from 1 to 65534 AS<integer>
<condition>	Please see RFC 2622 [1]

<table>
<tr><td><domain-name></td><td>Domain name as specified in RFC 1034 [7] without trailing dot ("."). The total length should not exceed 255 characters (octets)

<dlabel>(\.<dlabel>)+</td></tr>
<tr><td><e-mail></td><td>e-mail address as defined in RFC 2822 [8],</td></tr>
<tr><td><filter></td><td>Please see RFC 2622 [1]</td></tr>
<tr><td><free-form></td><td>A sequence of ASCII characters</td></tr>
<tr><td><ipv4-address></td><td>An IPv4 address is represented as a sequence of four integers in the range from 0 to 255 separated by the character dot ("."). For example, 128.9.128.5 represents a valid IPv4 address.

[0-9]+(\.[0-9]+){3,3}</td></tr>
<tr><td><ipv6-address></td><td><quad>(:<quad>){7,7}</td></tr>
<tr><td><mntner-name></td><td><object-name></td></tr>
<tr><td><nic-handle></td><td>From 2 to 4 characters optionally followed by up to 5 digits optionally followed by a source specification. Source specification starts with "-" followed by source name up to 9-character length.

(<alpha>{2,4}(<digit><digit>{0,5})?(-([a-zA-Z]{1,9}))?)|(AUTO-<digit>+(<alpha>{2,4})?)</td></tr>
<tr><td><object-name></td><td>Many objects in RPSL have a name. An <object-name> is made up of letters, digits, the character underscore "_", and the character hyphen "-"; the first character of a name must be a letter, and the last character of a name must be a letter or a digit. The following words are reserved by RPSL, and they can not be used as names:

any as-any rs-any peeras and or not
atomic from to at action accept announce
except refine networks into inbound outbound

Names starting with certain prefixes are reserved for certain object types. Names starting with "as-" are reserved for as set names. Names starting with "rs-" are reserved for route set names. Names starting with "rtrs-" are reserved for router set names. Names starting with "fltr-" are reserved for filter set names. Names starting with "prng-" are reserved for peering set names.</td></tr>
<tr><td><peering></td><td>Please see RFC 2622 [1]</td></tr>
</table>

<person-name>	Is a list of at least 2 words separated by white space. The first and the last word cannot end with dot ("."). The following words are not allowed: "Dr", "Prof", "Mv", "Ms", "Mr", no matter whether they end with dot (".") or not. A word is made up of letters, digits, the character underscore "_", and the character hyphen "-"; the first character of a name must be a letter, and the last character of a name must be a letter or a digit.
<protocol>	Please see RFC 2622 [1]
<registry-name>	RIPE
<router-expression>	Please see RFC 2622 [1]
<telephone-number>	Contact telephone number. Can take one of the forms: '+' <integer-list> '+' <integer-list> "(" <integer-list> ")" <integer-list> '+' <integer-list> ext. <integer list> '+' <integer-list> "(" integer list ")" <integer-list> ext. <integer-list>
<integer>	An integer
<alpha>	Any alphabetical character. [A-Za-z]
<alnum>	Any alphabetical or numeric character. [A-Za-z0-9]
list of	A list of words separated by comma (","). Cannot be empty.
ripe-list of	A list of words separated by white space. Cannot be empty.
<integer-list>	A list of <integer> with white space or dash ("-") as separators.

Descriptions of the attributes are listed below in the following format:

<attribute_name>   <attribute_value(type)>
<description>

address: <free-form>
Full postal address of a contact.

admin-c: <nic-handle>
References an on-site administrative contact.

aggr-bndry: <as-expression>
Defines a set of ASes, which form the aggregation boundary.

aggr-mtd: inbound | outbound [<as-expression>]
Specifies how the aggregate is generated. Please see [1] for more information.

as-name: <object-name>
A descriptive name associated with an AS.

as-set: <object-name>
Defines the name of the set.

auth: <auth-scheme> <scheme-info>
Defines an authentication scheme to be used.
<auth-scheme> and <scheme-info> can take the following values:

<auth-scheme>	**<scheme-info>**	**Description**
NONE		No authentication is required to make changes to the object protected with such authentication scheme.
MAIL-FROM	regular expression that matches e-mail addresses	This authentication method checks the content of the RFC 822 "From:" header of an update request against the regular expression specified as <scheme-info>. If the regular expression matches the content of the "From:" header, the update request is authenticated successfully. The regular expressions supported are described in POSIX 1003.2 section 2.8. As it is expected that most regular expressions will either be literals or of a form similar to .*@some.domain.or.other an extensive description of the possibilities will not be given. Note that the matching is applied to the whole content of the "From:" header including comments if present. No attempt is made to isolate the mailbox part. This authentication scheme is very weak. Forging RFC 822 headers does not take much effort or ingenuity. The reason for the scheme's existence is that it easily prevents accidental updates rather than allowing them first and fixing them later when notified.

CRYPT-PW	encrypted password, produced by UNIX crypt(3) routine	This scheme is slightly stronger than the MAIL-FROM scheme. It is by no means meant to keep out a determined malicious attacker. The crypt function is vulnerable to exhaustive search by (lots of) fast machines and programs to do the searching are widely available. For this reason it is strongly discouraged to use encrypted passwords also used for other purposes such as UNIX login accounts in this scheme. As you are publishing the encrypted password in the database, it is open to attack. The usual caveats about crypt passwords apply, so it is not very wise to use words or combinations of words found in any dictionary of any language.
PGPKEY-<id>		Strong scheme of authentication. <id> is the PGP key ID to be used for authentication. This string is the same one that is used in the corresponding **key-cert** object's "key-cert:" attribute.

author: <nic-handle>
References a limerick author.

aut-num: <as-number>
The autonomous system number.

certif: <public-key>
Contains the public key. The value of the public key should be supplied either using multiple "certif:" attributes, or in one ?certif:? attribute. In the first case, this is easily done by exporting the key from your local key ring in ASCII armored format and prepending each line of the key with the string "certif:". In the second case, line continuation should be used to represent an ASCII armored format of the key. All the lines of the exported key must be included; also the begin and end markers and the empty line which separates the header from the key body.

changed: <e-mail> [<date>]
Specifies who submitted the update, and when the object was updated. The format of the date is YYYYMMDD; dates in the future are not allowed. If the date is not specified, the database will put the date when the update was actually processed.

components: [ATOMIC] [[<filter>] [protocol <protocol> <filter> ...]]
The "components:" attribute defines what component routes are used to form the aggregate.

<protocol> is a routing protocol name such as BGP4, OSPF or RIP and <filter> is a policy expression.
Please refer to RFC 2622 [1] for more information.

country: <country-code>
Identifies the country. <country-code> must be a valid two-letter ISO 3166 country code.

cross-mnt: list of <mntner-name>
references mntner(s) to be notified. The "cross-mnt:" attribute may be used in:

- **route object** - to get a notification for any overlaps with the prefix specified in that **route** object;
- **aut-num** object - to get a notification for overlaps with any of the prefixes announced in **route** objects in which the given AS number is specified in the "origin:" attribute.

A notification will be sent to mailbox(es) listed in cross-nfy and mailbox(es) listed in the "mnt-nfy:" attributes of the maintainers referenced by the cross-mnt whenever an overlapping route is added or removed.

cross-nfy: <nic-handle>
The cross-nfy attribute contains a NIC-handle pointing to a **person** or **role** object, the email address of which will be used for notification about the addition or removal of **route** objects, which overlap the prefixes of the already registered **route** objects. The "cross-nfy:" attributes may be used in:

- **route** object - to get a notification for any overlaps with the prefix specified in that **route** object;
- **aut-num** object - to get a notification for overlaps with any of the prefixes announced in **route** objects in which the given AS number is specified in the "origin:" attribute.

See also "cross-mnt:" attribute.

default: to <peering> [action <action>] [networks <filter>]
Specifies default routing policies. Please refer to RFC 2622 [1] for more information.

descr: <free-form>
A short description related to the object

domain: <domain-name>
DNS name. <domain-name> is a fully qualified domain name without trailing ".".

dom-net: ripe-list of <ipv4-address>
List of IP networks in a domain.

export: to <peering-1> [action <action-1>]
. . .
to <peering-N> [action <action-N>]
announce <filter>
Specifies an export policy expression. Please refer to RFC 2622 for more information.

export-comps: <filter>
Specifies an RPSL filter that matches the more specifics that need to be exported outside the aggregation boundary. Please refer to RFC 2622 [1] for more information.

fax-no: <telephone-number>
The fax number of a contact.

filter: <filter>
Defines the set's policy filter, a logical expression which when applied to a set of routes returns a subset of these routes. Please refer to RFC 2622 [1] for more information.

filter-set: <object-name>
Defines the name of the filter. Please refer to RFC 2622 [1] for more information.

fingerpr: <generated>

A fingerprint of a key certificate generated by the database.

holes: list of <address-prefix>
Lists the component address prefixes that are not reachable through the aggregate route (perhaps that part of the address space is unallocated). Please refer to RFC 2622 [1] for more information.

ifaddr: <ipv4-address> masklen <integer> [action <action>]
Specifies an interface address within an Internet router. Please refer to RFC 2622 [1] for more information.

import: [protocol <protocol-1>] [into <protocol-2>]
 from <peering-1> [action <action-1>]
 . . .
 from <peering-N> [action <action-N>]
 accept <filter>
Specifies import policy expression. Please refer to RFC 2622 [1] for more information.

inetnum: <ipv4-address> - <ipv4-address>
Specifies a range of IPv4 that **inetnum** object presents. The ending address should be greater than the starting one.

inet6num: <ipv6-address>/<prefix-length>
Specifies a range of IPv6 addresses in prefix notation. The <prefix length> is an integer in the range from 0 to 128.

inet-rtr: <domain-name>
Fully qualified DNS name of the **inet-rtr** without trailing ".". Please refer to RFC 2622 [1] for more information.

inject: [at <router-expression>]
 [action <action>]
 [upon <condition>]
Specifies which routers perform the aggregation and when they perform it. Please refer to RFC 2622 [1] for more information.

key-cert: PGPKEY-<id>
Defines the public key stored in the database. <id> is the PGP key ID of the public key in 8-digit hexadecimal format without "0x" prefix.

local-as: <as-number>
Specifies the autonomous system that operates the router. Please refer to RFC 2622 [1] for more information.

limerick: LIM-<string>
Specifies the title of the limerick. <string> can include alphanumeric characters, "-" character.

method: <generated>
Defines the type of the public key. Currently, only PGP keys are supported.

member-of: list of <set-name>
This attribute can be used in the **route**, **aut-num** and **inet-rtr** classes. The value of the "member-of:" attribute identifies a set object that this object wants to be a member of. This claim, however, should be acknowledged by a respective "mbrs-by-ref:" attribute in the referenced object. Please refer to RFC 2622 [1] for more information.

members: list of <as-number> *or* <as-set-name>

or

members: list of <address-prefix-range>*or*
<route-set-name><range-operator>

or

members: list of <inet-rtr-name> *or* <rtr-set-name> *or*
<ipv4 address>

Lists the members of the set. The first form appears in the **as-set** object. The syntax of <as-set-name> is the same as the syntax of <object-name>. The second form appears in the **route-set** object. The syntax of <route-set-name> is the same as the syntax of <object-name>. The third form appears in the **rtr-set** object. The syntax of <intet-rtr-name> is the same as the syntax of <object-name>. Please refer to RFC-2622 [1] for more information.

mbrs-by-ref: list of <mntner-name> | ANY
This attribute can be used in all "set" objects; it allows indirect population of a set. If this attribute is used, the set also includes objects of the corresponding type (**aut-num** objects for **as-set**, for example) that are protected by one of these maintainers and whose "member-of:" attributes refer to the name of the set. If the value of a "mbrs-by-ref:" attribute is ANY, any object of the corresponding type referring to the set is a member of the set. If the "mbrs-by-ref:" attribute is missing, the set is defined explicitly by the "members:" attribute.

mntner: <object-name>
A unique identifier of the mntner object.

mnt-by: list of <mntner-name>
Specifies the identifier of a registered **mntner** object used for authorisation of operations performed with the object that contains this attribute.

mnt-lower: list of <mntner-name>
Specifies the identifier of a registered **mntner** object used for hierarchical authorisation. Protects creation of objects directly (one level) below in the hierarchy of an object type (only for **inetnum**, **inet6num**, **as-block**, **aut-num**, **route** or **domain** objects). The authentication method of this **maintainer** object will then be used upon creation of any object directly below the object that contains the "mnt-lower:" attribute.

mnt-nfy: <e-mail>
Specifies the e-mail address to be notified when an object protected by a mntner is successfully updated.

mnt-routes: <mnt-name> [{ list of <address-prefix-range> } | ANY]
May appear in an **aut-num**, **inetnum** or **route** object. This attribute references a **maintainer** object which is used in determining authorisation for the creation of **route** objects. After the reference to the maintainer, an optional list of prefix ranges inside of curly braces or the keyword "ANY" may follow. The default, when no additional set items are specified, is "ANY" or all more specifics. Please refer to RFC-2622 [1] for more information.

netname: <netname>
Specifies the name of a range of IP address space. The syntax of the <netname> attribute is the same as the syntax of the <object-name> attribute, but it does not have a restriction on RPSL reserved prefixes.

nic-hdl: <nic-handle>
Specifies the NIC handle of a **role** or **person** object. When creating an object, one can also specify an "AUTO" NIC handle by setting the value of the attribute to "AUTO-1" or AUTO-1 <Initials>. In such case the database will assign the NIC handle automatically.

notify: <e-mail>
Specifies the e-mail address to which notifications of changes to an object should be sent.

nserver: ripe-list of (<domain-name> | <ipv4-address>)
Specifies the nameservers of the domain.

origin: <as-number>
Specifies the AS that originates the route. The corresponding **aut-num** object should be registered in the database.

owner: <generated>
Specifies the owner of the public key.

peering: <peering>
Defines a peering that can be used for importing or exporting routes. Please refer to RFC 2622 [1] for more information.

peering-set: <object-name>
Specifies the name of the peering-set. Please refer to RFC 2622 [1] for more information.

person: <person-name>
Specifies the full name of an administrative, technical or zone contact person for other objects in the database. <person name> cannot contain titles such as "Dr.", "Prof.", "Mv.", "Ms.", "Mr.", etc. It is composed of alphabetic characters.

peering-set: <object-name>
Specifies the name of the peering-set. Please refer to RFC 2622 [1] for more information.

phone: <telephone-number>
Specifies a telephone number of the contact.

refer: <type> <hostname> [<port>]
Specifies the referral type, hostname and port that the server should use to redirect the query when using referral mechanism for lookups for **domain** objects. Please see section 2.7 "Referral mechanism for domains" for more information.

<type> specifies the type of referral to be used. Please see the table below for the supported types.
<hostname> is the DNS name or <ipv4 address> of the referred host.
<port> is an integer specifying TCP port number at which queries are accepted by the referred host. If <port> is omitted, the default number of 43 is used.

Referral type	**Description**
SIMPLE	Only lookup key (domain name) is passed to the referred server. All query flags are stripped.
INTERNIC	Same as SIMPLE. Supported for backward compatibility.
RIPE	Used when the referred server understands RIPE query flags. With this type of referral, all query flags specified by the client will be passed to the referred server unmodified.
CLIENTADDRESS	Same as SIMPLE, but the server will add "-V <version>, <ipv4 address>" flag to the query, where <version> is the version number of the server and <ipv4 address> is the IP address of the client that made this query. This referral type allows the referred host to perform accounting and implement an access control for clients using the RIPE Database server as a proxy.

referral-by: <mntner-name>
This attribute is required in the **mntnr** object. It may never be altered after the addition of the maintainer. This attribute refers to the maintainer that created this maintainer. Currently, all **mntner** objects are created manually by the Database Administration, so the value of the "referral-by:" attribute should be ?RIPE-DBM-MNT"

remarks: <free-form>

Contains remarks.

role: <person-name>
Specifies the full name of a role entity, e.g. RIPE DBM.

rev-srv: ripe-list of <domain-name>
Specifies a DNS nameserver for a range of IP addresses represented by the **inetnum** object that contains this attribute.

route: <address-prefix>
Specifies the prefix of the interAS route. Together with the "origin:" attribute, constitutes a primary key of the **route** object.

route-set: <object-name>
Specifies the name of the route set. It is a primary key for the route-set object. Please refer to RFC 2622 [1] for more information.

rtr-set: <object-name>
Defines the name of the **rtr-set**. Please refer to RFC 2622 [1] for more information.

source: <registry-name>
Specifies the registry where the object is registered. Should be "RIPE" for the RIPE Database.

status: <status>
Specifies the status of the address range represented by inetnum or **inet6num** object.
For an **inetnum** object <status> can have one of these values:

- ALLOCATED PA
- ALLOCATED PI
- ALLOCATED UNSPECIFIED
- ASSIGNED PA
- ASSIGNED PI

Please refer to the RIPE document "European Internet Registry Policies and Procedures" for further information.
For inet6num, <status> is automatically generated by the database based on the

range size specified by the "inet6num:" attribute.

sub-dom: ripe-list of <domain-name>
Specifies list of sub-domains of a domain. Domain names are relative to the domain represented by the **domain** object that contains this attribute.

tech-c: <nic-handle>
References a technical contact.

text: <free-form>
Contains text of the limerick. Must be humorous, but not malicious or insulting.

trouble: <free-form>
Contains information on who to contact when there are problems.

upd-to: <e-mail>
Specifies the e-mail address to be notified when an object protected by a mntner is unsuccessfully updated. See also section 3.5.2. "Notifications".

zone-c: <nic-handle>
References a zone contact.

A2. RIPE Database response codes and messages

If the server encounters a problem, an error message is returned as a query result. The format of an error message is as follows:

%ERROR:#:<message>,

where # is the error code and <message> is a short description of the problem, possibly followed by a more descriptive message, each line of which starts with % followed by a white space and a text.

Example:

% This is the RIPE Whois server v3.0 beta.

% This server mirrors the RIPE Database in RPSL format.
% Rights restricted by copyright.
% See http://www.ripe.net/ripencc/pub-services/db/copyright.html

%ERROR:101: no entries found
%
% No entries found in the selected source(s).

A2.1 Query errors

%ERROR:101: no entries found

No entries were found in the selected source(s).

%ERROR:102: unknown source

Unknown source is requested (-s flag). Use -q sources for a list of available sources.

%ERROR:103: unknown object type

Unknown object type is specified in the filter expression (-T flag)

%ERROR:104: unknown attribute

Unknown attribute is specified in the inverse query (-i flag). See section 2.0 "Queries in the RIPE Database" for more information.

%ERROR:105: attribute is not searchable

The attribute specified in the inverse query is not searchable (-i flag). See section 2.0 "Queries in the RIPE Database" for more information.

%ERROR:106: no search key specified

No search key has been specified in the query.

A2.2 Access errors

%ERROR:201: access denied

Access from the host has been permanently denied because of excessive querying. You need to contact Database Administration; ripe-dbm@ripe.net) to discuss this problem.

%ERROR:202: access control limit reached

Limit of returned objects has been reached. The connection is terminated. Continued attempts to excessively query the database will result in permanent denial of service. See section 2.8 "Access control for queries" for more information.

%ERROR:203: address passing not allowed

The host is not registered as a proxy, thus it is not allowed to pass addresses on the query line ("-V" flag). See section 2.8 "Access control for queries" for more information.

%ERROR:204: maximum referral lines exceeded

The referral query result exceeded maximum number of lines. Only maximum lines are output and connection is closed by the server.

A2.3 Connection errors

%ERROR:301: connection has been closed

The connection is administratively closed.

%ERROR:302: referral timeout

The connection was closed due to referral timeout.

%ERROR:303: no referral host

Referral host cannot be found.

%ERROR:304: referral host not responding

The connection to the referral host cannot be established.

A2.4 NRTM errors

%ERROR:401: invalid range: Not within <first>-<last>

This happens when the requested range or part of it is outside the serial numbers available at the server. <first> is the lowest serial number available. <last> is the most recent serial number available.

%ERROR:402: not authorised to mirror the database

See section 2.8 "Access control for queries" for more information. One may use "-q sources" query to get more information about the NRTM source.

%ERROR:403: unknown source

The database identified by the <source> is not served by the server. Use "-q sources" for a list of available sources.

A2.5 Referral text

%REFERRAL START

After this line, output of the referred host starts.

% The object shown below is NOT in the %s database.
% It has been obtained by querying a remote server:
% cctld.dom.int at port 43.
% To see the object stored in the %s database
% use the -R flag in your query
%
%REFERRAL START

%REFERRAL END

Referral trailer, which frames the output of the referred host.

A3. Copyright information

A3.1 RIPE Database copyright

The information in the RIPE Database is available to the public for agreed Internet operation purposes, but is under copyright. The copyright statement is:

"Except for agreed Internet operational purposes, no part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, recording, or otherwise, without prior permission of the RIPE NCC on behalf of the copyright holders. Any use of this material to target advertising or similar

A3.2 RFC copyright statement

A3.3 RIPE NCC copyright

Acknowledgements

The authors wish to acknowledge the effort done by the developers of the version 3.0 of the RIPE Database at the RIPE NCC: Daniele Arena, Marek Bukowy, Engin Guenduez, Roman Karpiuk, Shane Kerr, A.M.R. Magee Chris Ottrey and Filippo Portera.

References

[1] C. Alaettinoglu, C. Villamizar, E. Gerich, D. Kessens, D. Meyer, T. Bates, D. Karrenberg and M. Terpstra, "Routing Policy Specification Language (RPSL)", RFC 2622, June 1999.

[2] C. Villamizar, C. Alaettinoglu, D. Meyer and S. Murphy, "Routing Policy System Security", RFC 2725, December 1999.

[3] D. Meyer, J. Schmitz, C. Orange, M. Prior, and C. Alaettinoglu, "Using RPSL in Practice", RFC 2650, August 1999.

[4] T. Bates, E. Gerich, L. Joncheray, J.M. Jouanigot, D. Karrenberg, M. Terpstra and J. Yu, "Representation of IP Routing Policies in a Routing Registry", ripe-181, October 1994. See http://www.ripe.net/docs/ripe-181.html

[5] The "RIPE Database User's Manual" and the "RIPE Database Operations Manual" are scheduled to be written.

[6] RAToolset. See http://www.isi.edu/ra/RAToolSet/

[7] P. Mockapetris, "Domain names - Concepts and Facilities", RFC 1034, November1987.

[8] P. Resnick, ed., "Internet Message Format", RFC 2822, April 2001.