RIPE Database Update Reference Manual

Abstract

This document describes how to update series 3.x of the RIPE Database. This series uses the Routing Policy Specification Language (RPSL) [1] to represent all database objects. It uses the Routing Policy System Security (RPSS) [2] for authorisation. This provides better security for Internet Routing Registries (IRR). It makes use of RPSL next generation specifications [14]. This allows for registering of multicast and IPv6 routing policies. Though this document is self-contained, you may also read the RPSL [1] and RPSS [2] specifications. For a tutorial on RPSL, you can read the RPSL applications document [3].

Intended Audience

This reference manual is for intermediate and advanced users who update the RIPE Database. If you are new to the database, you might find the "RIPE Database User Manual: Getting Started" [5] to be a more helpful place to start.

Conventions Used in This Document

We use <label> for a placeholder or to indicate syntax.
We use [option] to indicate an optional text or command argument.
In object templates, we use square brackets "[ ]" to indicate an attribute type.

"RIPE Database" is used with different meanings depending on the context. It may mean the interface software, the logical database or the information in the database. Where there may be doubt, this manual will make clear what is being discussed.

Table of Contents

Introduction

1.0 Database Objects and Attributes
1.1 Object Representation
1.2 Object Types
1.2.1 as-block
1.2.2 as-set
1.2.3 aut-num
1.2.4 domain
1.2.5 filter-set
1.2.6 inet6num
1.2.7 inetnum
1.2.8 inet-rtr
1.2.9 irt
1.2.10 key-cert
1.2.11 mntner
1.2.12 organisation
1.2.13 peering-set
1.2.14 person
1.2.15 poem
1.2.16 poetic-form
1.2.17 role
1.2.18 route6
1.2.19 route
1.2.20 route-set
1.2.21 rtr-set
2.0 Updates in the RIPE Database
2.1 Format of an Update Message
2.2 Creating, Modifying and Deleting an Object
2.2.1 Object Processing
2.2.2 Creating a New Object
2.2.3 Modifying an Existing Object
2.2.4 Deleting an Object
2.3 E-mail Updates
2.3.1 MIME Support
2.3.2 PGP and X.509 Support
2.3.3 Subject Line Processing
2.3.3.1 NEW Keyword
2.3.3.2 HELP (HOWTO) Keyword
2.3.3.3 DIFF Keyword
2.4 Acknowledgements and Notifications
2.4.1 Acknowledgements
2.4.2 Notifications
2.5 Protecting Data
2.5.1 Authorisation Model
2.5.2 Protection of Individual Objects
2.5.3 Protection of aut-num Object Space
2.5.4 Protection of Address Space

2.5.5 Protection of Route Object Space
2.5.6 Protection of Objects with Hierarchical Names
2.5.7 Protection of Domain Object Space
2.5.8 Protecting Membership of a Set
2.5.9 Referencing an irt Object
2.5.10 Referencing an Organisation Object
3.0 Mirroring of the RIPE Database
4.0 Using the RIPE Database Efficiently
4.1 Using the Role Object
4.2 Using the Organisation Object
4.3 Abuse Handling
Appendices
A1. Object Attributes
A2. Copyright Information
A2.1 RIPE Database Copyright
A2.2 RIPE NCC Copyright
Acknowledgements
References

Introduction

The RIPE Network Management Database (often called the "RIPE Database") contains information about IP address space allocations and assignments, routing policies and reverse delegations in the RIPE NCC service region:

http://www.ripe.net/membership/maps/index.html

It also contains some information about forward domain names. However, this information about domain names is for reference only. The RIPE Database is not the same as a domain name registry, which are run by the country code Top-Level Domain (ccTLD) administrators of Europe and surrounding areas. If this is what you need, you should use the IANA ccTLD Database for a full list of the ccTLD administrators [17].

While the information in the RIPE Database is made freely available to the public for Internet operational purposes, it is subject to copyright restrictions. See Appendix A2, 'Copyright information'.

This document describes how to update series 3.x of the RIPE Database. This series uses the Routing Policy Specification Language (RPSL) [1] to represent all database objects. It uses the Routing Policy System Security (RPSS) [2] for authorisation. This provides better security for Internet Routing Registries (IRR). The RIPE Database includes an IRR as well as an Internet Number Registry (INR). It makes use of RPSL next generation specifications [14]. This allows for registering of multicast and IPv6 routing policies.

This document focuses on how to update the database. It does not explain in any detail how the database software works. It does not always make clear why updates work this way. For that you may need to refer to RIPE Policy documents [19]. This document is not a statement of any RIPE Policy.

This document is self-contained, but does not provide many examples of usage or illustrations of how the RIPE Database works. If this is what you want, you should read the RPSL [1] and RPSS [2] specifications. If you are looking for a tutorial on RPSL, you should read the RPSL applications document [3]. The "RIPE Database User Manual: Getting Started" [5] contains some examples. You may also need to read the "RIPE Database Query Reference Manual" [18] which explains how to query the RIPE Database. These two documents form a set.

1.0 Database Objects and Attributes

The RIPE Database contains records of:

- allocations and assignments of IP address space (the IP address registry or INR);
- domain names (mainly for reverse domains);
- routing policy information (the routing registry or IRR);
- contact information (details of people and organisations who are registered as the contacts for the operation of networks or routers. The RIPE NCC does not maintain the contents of the database. The registered contacts are the people who do this).

The information on domain names (except for in-addr.arpa reverse domains) has no effect on operations. Forward domain information is included in the RIPE Database for reference only and may not be complete or authoritative. It may be removed in the future, except for referral links. If you want authoritative information on forward domains, you need to contact ccTLD administrators [17].

1.1 Object Representation

The records in the RIPE Database are known as **"objects"**. RPSL defines the syntax of database objects (how they are written). You can find out about RPSL in [1]. An object belongs to one of the object types or classes. We use the two terms - 'type' and 'class' - interchangeably throughout this document.

Table 1.1 Object Types Supported by the RIPE Database

Object type (Class name)	Short name	Description
as-block	ak	Represents delegation of a range of Autonomous System (AS) Numbers to a given repository.
as-set	as	Set of **aut-num** objects.
aut-num	an	AS in the database. It describes the external routing policy of the AS.
domain	dn	Forward or reverse domain registrations.
filter-set	fs	Set of routes matched by its filter.
inet6num	i6	Allocations and assignments of IPv6 address space.
inetnum	in	Allocations and assignments of IPv4 address space.
inet-rtr	ir	Router in the database.
irt	it	Contact and authentication information about a Computer Security Incidence Response Team (CSIRT).
key-cert	kc	Public key certificate that is stored on the server and may be used with a **mntner** object for authentication when performing updates.
mntner	mt	Authentication information needed to authorise creation, deletion or modification of the objects protected by the **mntner**.
organisation	oa	Organisation that holds the resources.
peering-set	ps	Set of peerings.
person	pn	Technical or administrative contacts.
poem	po	Humorous poem.
poetic-form	pf	Type of humour for a **poem** object.
role	ro	List of person objects for a set of people who perform a role.
route	rt	IPv4 route advertised on the Internet.
route6	r6	IPv6 route advertised on the Internet.
route-set	rs	Set of routes.
rtr-set	is	Set of routers.

The RIPE NCC defines a database object as a list of attribute-value pairs in plain text form. The database software only recognises the printable ASCII character set. If you use any other character sets, or non-printable characters, it may cause problems and your updates could fail.

When an object is stored in the database, the attributes and values are unchanged and the order is maintained. The software might adjust spacing between them to align the values and make them easier to read.

Each attribute-value pair must start on a separate line. The software is set up to treat a blank line as the end of an object. This is why you cannot include a completely blank line in the middle of an object.

Attribute names have a precisely defined syntax and only use alpha numeric and the hyphen (-) characters. They are not case sensitive, but most people use lower case. The attribute name must start at column 0 and must immediately be followed by a colon (:). No spaces or tabs are allowed in between the attribute name and the colon. If you enter anything different, you will see an error message and your update will fail.

The first attribute that you need to specify is the one that has the same name as the object type. The database software uses this to identify the object type. If the object type is not recognised, that part of the message will be treated as 'other' text. Any 'other' text will be disregarded by the software. Other attributes can appear in any order, but most people stick to the order as shown in the object templates. Each object is uniquely identified by a set of attribute values. We call this set of attributes the 'class primary key'. For most object types this is the value of the first attribute. In some cases it is a different attribute value or a composite of more than one attribute value. The attributes which make up the 'class primary key' are shown in the object templates.

The value part of the attribute-value pair starts after the colon (:). It can contain some pre-defined keywords, references to other objects and free text. You can refer to other objects by using their 'class primary key' values. These references and the keywords have a precisely defined syntax. If you enter anything different, or if the objects you refer to do not already exist in the database, you will see error messages and your update will fail. The free text has no syntax, but may only contain recognisable characters.

Attribute values may contain spaces and tab characters to help make the information easier to read. Note that spaces work better than tabs, as tabs can display differently on different machines.

You can also split a value over several lines by placing a space, a tab or a plus (+) sign in column 0 of each of the continuation lines. If you wish, you can use more spaces after the continuation character to make it easier to read. The plus (+) sign for line continuation allows attribute values to contain an almost blank line. The software is set up to treat a blank line as the end of an object. This is why you cannot include a completely blank line in the middle of an object.

An attribute value may also contain 'end of line' comments. These start with a hash (#) and continue until the end of the physical line. If a value is split over several lines, any of the lines may include an 'end of line' comment. You cannot continue the comment on another line. These comments always stop at the end of the line in which they start. An end of line comment cannot start at column 0. It is possible to add end of line comments on several consecutive lines which together form a block of text. However, for long comments, it is better to use the "remarks:" attribute.

Any free form value cannot contain a hash (#) as the software would treat this as a comment. Although the software does not process comments, in some situations it does strip off the comments before using the values.

Long end of line comments or long free form attribute values can cause problems. Some mail clients automatically break lines at a certain point. If your mail client does this on an update message then your update will fail. It may not be obvious at first sight that this has happened.

Attributes can be mandatory, optional or generated.

You MUST define 'mandatory attributes' in all instances of an object type. If you do not, then the update will fail.

You can skip 'optional attributes'. However, if you do decide to define them, then both the attribute and its value must be syntactically correct. If they are not, then your update will fail. When you skip an optional attribute, remove it completely from the object. You cannot include the attribute name and leave the value blank (unless it is free text).

The database software creates any 'generated attributes'. You can skip generated attributes. Where a user-supplied value is not correct, the software will replace it with a generated value. If the software changes a user-supplied value, it will explain the change in a warning message returned to the user. When you skip a generated attribute, remove it completely from the object. You cannot include the attribute name and leave the value blank (unless it is free text).

Attributes can have single or multiple values.

You can only include a 'single valued attribute' once in an object. That one instance can only have a single value.

You can include a 'multiple valued attribute' many times within an object. Each of these attribute instances may also have multiple values separated by commas. However, if the value is free text, the attribute may only have a single value - as a comma may be a part of the free text. But it can still have multiple instances.

Only attributes with free text values (no keywords or references to other objects) can have a blank value. The software will treat any other attribute, including optional and generated attributes, with a blank value as a syntax error and the update will fail.

The value for each type of attribute has a format that defines its syntax. For a detailed description of the attributes supported in the RIPE Database, you should refer to Appendix A1, 'Object Attributes'.

1.2 Object Types

This section describes the object types (classes) that the RIPE Database supports. Some description is included about the definition and use of the attributes of each object type. For a strict definition of the syntax of each attribute see Appendix A1, 'Object Attributes'.

A set of object templates show which attributes are allowed in each object type. We use the following definitions in the templates:

[mandatory]	You must include at least one instance of this attribute in an object of the class.
[optional]	This attribute is optional in the objects of the class and you can miss it out completely.
[generated]	The server automatically generates this attribute and you can miss it out completely. If you provide the value it may be replaced.
[single]	Objects must not contain more than one instance of this attribute - value pair.
[multiple]	Objects may contain more than one instance of this attribute. An instance may contain more than one value, separated by commas.
[look-up key]	This attribute is indexed.
[inverse key]	This attribute is in the "reverse" index.
[primary key]	This attribute is (part of) the class primary key.
[primary/lookup key]	This attribute is indexed and is also (part of) the class 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

The RIPE Database Administrators create **as-block** objects manually; it is not an automatic process. If a user attempts to create one, the request is forwarded to the RIPE Database Administrator.

An **as-block** object delegates a range of AS Numbers to a given repository.

This object sets the authorisation required for the creation of **aut-num** objects within the range specified by the "as-block:" attribute. This is set by the "mnt-lower:" and "mnt-by:" attributes.

Here is an **as-block** object template:

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

1.2.2 as-set

An **as-set** object defines a set of **aut-num** objects.

- The "as-set:" attribute defines the name of the set. It is an RPSL name that starts with "as-".
- The name of an **as-set** object can be hierarchical. A hierarchical as-set name is a sequence of as-set names and AS Numbers separated by colons. At least one component must be an actual as-set name (i.e. start with "as-"). All the set name components of a hierarchical as-name have to be as-set names.
- The "members:" attribute lists the members of the set. It can be either a list of AS Numbers, or other as-set names.
- The "mbrs-by-ref:" 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.
- This object sets the authorisation required for the creation of other **as-set** objects one level down in a hierarchy. This is set by the "mnt-lower:" and "mnt-by:" attributes.

Here is an **as-set** object template:

```
as-set:       [mandatory]  [single]     [primary/lookup key]
descr:        [mandatory]  [multiple]   [ ]
members:      [optional]   [multiple]   [ ]
```

```
mbrs-by-ref:  [optional]   [multiple]   [inverse key]
remarks:      [optional]   [multiple]   [ ]
org:          [optional]   [multiple]   [inverse key]
tech-c:       [mandatory]  [multiple]   [inverse key]
admin-c:      [mandatory]  [multiple]   [inverse key]
notify:       [optional]   [multiple]   [inverse key]
mnt-by:       [mandatory]  [multiple]   [inverse key]
mnt-lower:    [optional]   [multiple]   [inverse key]
changed:      [mandatory]  [multiple]   [ ]
source:       [mandatory]  [single]     [ ]
```

1.2.3 aut-num

The **aut-num** object specifies routing policies. It refers to a group of IP networks that have a single and clearly defined external routing policy, operated by one or more network operators – an Autonomous System (AS).

These can now be 16 bit or 32 bit numbers. The representation is slightly different. For 16 bit numbers the class primary key values will be in this format:

ASn where n is a 16 bit number.

For 32 bit numbers class primary key values will be in this format:

ASx.y where x and y are the high and low end 16 bit numbers respectively which together represent a 32 bit number.

If a 16 bit number is supplied in 32 bit number format (AS0.432) it is converted to 16 bit number format (AS432).

Leading zeroes (AS0352) are not allowed in either format and will be removed (AS352).

- The value of the "aut-num:" attribute is the AS Number of the Autonomous System that this object describes and starts with AS.
- The "as-name:" attribute is a symbolic name of the AS.
- The "member-of:" attribute value 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.
- The import, export and default routing policies of the AS are specified using the "import:", "export:" and "default:" attributes respectively.
- Corresponding attributes with "mp-" prefix, "mp-import:", "mp-export:" and "mp-default:" are used to specify IPv6 and multicast routing policies.
- Only a single value for the "org:" attribute is allowed in the **aut-num** object. This is to ensure only one organisation is responsible for this resource.
- This object sets part of the authorisation required for the creation of **route** and **route6** objects. This is set by the "mnt-routes:", "mnt-lower:" and "mnt-by:" attributes.

Here is an **aut-num** object template:

```
aut-num:        [mandatory]  [single]     [primary/lookup key]
as-name:        [mandatory]  [single]     [ ]
descr:          [mandatory]  [multiple]   [ ]
member-of:      [optional]   [multiple]   [inverse key]
import:         [optional]   [multiple]   [ ]
mp-import:      [optional]   [multiple]   [ ]
export:         [optional]   [multiple]   [ ]
mp-export:      [optional]   [multiple]   [ ]
default:        [optional]   [multiple]   [ ]
mp-default:     [optional]   [multiple]   [ ]
remarks:        [optional]   [multiple]   [ ]
org:            [optional]   [multiple]   [inverse key]
admin-c:        [mandatory]  [multiple]   [inverse key]
tech-c:         [mandatory]  [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]     [ ]
```

1.2.4 domain

The **domain** object represents reverse delegations . You can also use this for Top-Level Domain (TLD) and other forward domain registrations and referrals

- You should write the domain name in fully qualified format, without a trailing dot. If a trailing dot is included it will be removed by the software and a warning message returned to the user.
- If the nameserver name in the "nserver:" attribute is inside the domain being delegated it may be optionally followed by an IP address (IPv4 or IPv6).
- The "ds-rdata:" attribute holds information about a signed delegation record for DNSSEC (short for DNS Security Extensions)
- The "sub-dom:" attribute specifies list of sub-domains of a domain. Domain names are relative to the domain represented by the **domain** object that contains this attribute.
- The "dom-net:" attribute contains a list of IP networks in a domain.
- The **domain** object sets the authorisation required for the creation of other **domain** objects one level down in a hierarchy. This is set by the "mnt-lower:" and "mnt-by:" attributes.
- The "refer:" attribute is used to refer a query to another authorative database. See the "RIPE Database Query Reference Manual" [18] for an explanation of its use.

Here is a **domain** object template:

```
domain:         [mandatory]  [single]     [primary/lookup key]
descr:          [mandatory]  [multiple]   [ ]
org:            [optional]   [multiple]   [inverse key]
admin-c:        [mandatory]  [multiple]   [inverse key]
tech-c:         [mandatory]  [multiple]   [inverse key]
zone-c:         [mandatory]  [multiple]   [inverse key]
nserver:        [optional]   [multiple]   [inverse key]
ds-rdata:       [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]     [ ]
```

1.2.5 filter-set

A **filter-set** object defines a set of routes that match the criteria that you specify in your 'filter' – in other words it filters out routes that you do not want to see.

The "filter-set:" attribute defines the name of your filter. It is an RPSL name that starts with "fltr-".

- The "filter:" attribute defines the policy filter of the set.
    - A policy filter is a logical expression which, when applied to a set of routes, returns a subset of these routes – the ones that you have said you want to see.

- The "mp-filter:" attribute extends the "filter:" attribute to allow you to specify IPv6 prefixes and prefix ranges.

    - A policy mp-filter is a logical expression which when applied to a set of multiprotocol routes returns a subset of these routes.

- The "filter:" and "mp-filter:" attributes are optional. However, if you plan to use a **filter-set** object, it must contain at least one of these two attributes.
- The name of a **filter-set** object can be hierarchical.
    - A hierarchical filter-set name is a sequence of filter-set names and AS Numbers separated by colons. At least one component of the name must be an actual filter-set name (i.e. start with "fltr-"). All the set name components of a hierarchical filter-name have to be filter-set names.
- This object sets the authorisation required for the creation of other **filter-set** objects one level down in a hierarchy. This is set by the "mnt-lower:" and "mnt-by:" attributes.
- Unlike other set objects this one does not include the "mbrs-by-ref:" attribute.

Here is a **filter-set** object template:

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

1.2.6 inet6num

An **inet6num** object contains information on allocations and assignments of IPv6 address space.

- The "inet6num:" attribute specifies a range of IPv6 addresses that the **inet6num** object presents. The range may be a single address.
- Addresses can only be expressed in prefix notation
- The prefix notation expresses addresses in hexadecimal groups of two bytes separated by colons and with the possible use of shorthand notation for strings of consecutive zeros. Leading zeros from any two-byte group will be removed by the software. In this case a warning message will be returned to the user.
- The "netname:" attribute is the name of a range of IP address space. It is recommended that the same netname is used for any set of assignment ranges used for a common purpose.
- The "country:" attribute identifies the country. It has never been specified if this is the country where the addresses are used, where the issuing organisation is based or some transit country in between. There are no rules defined for this attribute. It cannot therefore be used in any reliable way to map IP addresses to countries.
- Only a single value for the "org:" attribute is allowed in the **inet6num** object. This is to ensure only one organisation is responsible for this resource.
- The "rev-srv:" attribute specifies a DNS nameserver for a range of IP addresses represented by the **inet6num** object that contains this attribute. It is for information only and has no operational use.
- The "status:" attribute indicates where the address range represented by an **inet6num** object sits in a hierarchy and how it is used.
- Status can have one of these values:
    - ALLOCATED-BY-RIR
    - ALLOCATED-BY-LIR
    - ASSIGNED
    - ASSIGNED ANYCAST

- The **inet6num** object sets the authorisation required for the creation of more specific **inet6num** objects within the range specified by this **inet6num**. This is set by the "mnt-lower:" and "mnt-by:" attributes.
- This object sets the authorisation required for the creation of a **domain** object for reverse delegation. This is set by the "mnt-domains:", "mnt-lower:" and "mnt-by:" attributes.
- This object sets part of the authorisation required for the creation of a **route6** object. This is set by the "mnt-routes:", "mnt-lower:" and "mnt-by:" attributes.
- "mnt-irt:" references an **irt** object. Authorisation is required from the **irt** object to be able to add this reference.

Here is an **inet6num** object template:

```
inet6num:      [mandatory]  [single]     [primary/lookup key]
netname:       [mandatory]  [single]     [lookup key]
descr:         [mandatory]  [multiple]   [ ]
country:       [mandatory]  [multiple]   [ ]
org:           [optional]   [single]     [inverse key]
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]
mnt-domains:   [optional]   [multiple]   [inverse key]
mnt-irt:       [optional]   [multiple]   [inverse key]
changed:       [mandatory]  [multiple]   [ ]
source:        [mandatory]  [single]     [ ]
```

1.2.7 inetnum

An **inetnum** object contains information on allocations and assignments of IPv4 address space.

- The "inetnum:" attribute specifies a range of IPv4 addresses that the **inetnum** object presents. The range may be a single address. If the range represents multiple addresses, the ending address should be greater than the starting one.
- Addresses can be expressed in either range or prefix notation. If prefix notation is used, the software will convert this to range notation and a warning message will be returned to the user.
- The range notation expresses addresses as 32 bit whole numbers in dotted quad notation. Leading zeros from any quad will be removed by the software and a warning message will be returned to the user.
- The "netname:" attribute is the name of a range of IP address space. It is recommended that the same netname be used for any set of assignment ranges used for a common purpose, such as a customer or service.

- The "country:" attribute identifies the country. It has not been specified if this is the country where the addresses are used, where the issuing organisation is based or some transit country in between. There are no rules defined for this attribute. It cannot therefore be used in any reliable way to map IP addresses to countries.
- Only a single value for the "org:" attribute is allowed in the **inetnum** object. This is to ensure only one organisation is responsible for this resource.
- The "rev-srv:" attribute specifies a DNS nameserver for a range of IP addresses represented by the **inetnum** object that contains this attribute. It is for information only and has no operational use.
- The "status:" attribute indicates where the address range represented by an **inetnum** object sits in a hierarchy and how it is used.
- Status can have one of these values:
  - ALLOCATED UNSPECIFIED
  - ALLOCATED PA
  - ALLOCATED PI
  - LIR-PARTITIONED PA
  - LIR-PARTITIONED PI
  - SUB-ALLOCATED PA
  - ASSIGNED PA
  - ASSIGNED PI
  - ASSIGNED ANYCAST
  - EARLY-REGISTRATION
  - NOT-SET
- The **inetnum** object sets the authorisation required for the creation of more specific **inetnum** objects within the range specified by this **inetnum**. This is set by the "mnt-lower:" and "mnt-by:" attributes.
- This object sets the authorisation required for the creation of a **domain** object for reverse delegation. This is set by the "mnt-domains:", "mnt-lower:" and "mnt-by:" attributes.
- This object sets part of the authorisation required for the creation of a **route** object. This is set by the "mnt-routes:", "mnt-lower:" and "mnt-by:" attributes.
- The "mnt-irt:" attribute references an **irt** object. Authorisation is required from the **irt** object to be able to add this reference.

Here is an **inetnum** object template:

```
inetnum:       [mandatory]  [single]     [primary/lookup key]
netname:       [mandatory]  [single]     [lookup key]
descr:         [mandatory]  [multiple]   [ ]
country:       [mandatory]  [multiple]   [ ]
org:           [optional]   [single]     [inverse key]
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-domains:   [optional]   [multiple]   [inverse key]
mnt-routes:    [optional]   [multiple]   [inverse key]
mnt-irt:       [optional]   [multiple]   [inverse key]
changed:       [mandatory]  [multiple]   [ ]
source:        [mandatory]  [single]     [ ]
```

1.2.8 inet-rtr

The **inet-rtr** object specifies routers.

- The "inet-rtr:" attribute is a valid DNS name for a router without a trailing dot.
- Each "alias:" attribute, if present, is also standard DNS name for the specified router.
- The "local-as:" attribute specifies the AS Number of the AS that owns or operates this router.
- The "ifaddr:" attribute specifies the interface address within an Internet router, as well as an optional action to set other parameters on this interface.
- The "interface:" attribute specifies a multi-protocol interface address within an Internet router, optional action and tunnel definition.
- The "peer:" attribute specifies the details of any interior or exterior router peering.
- The "mp-peer:" attribute extends the "peer:" attribute for IPv6 addresses.
- The "member-of:" attribute value 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.

Here are the attributes of the **inet-rtr** object:

```
inet-rtr:      [mandatory]  [single]     [primary/lookup key]
descr:         [mandatory]  [multiple]   [ ]
alias:         [optional]   [multiple]   [ ]
local-as:      [mandatory]  [single]     [inverse key]
ifaddr:        [mandatory]  [multiple]   [lookup key]
interface:     [optional]   [multiple]   [lookup key]
peer:          [optional]   [multiple]   [ ]
mp-peer:       [optional]   [multiple]   [ ]
member-of:     [optional]   [multiple]   [inverse key]
remarks:       [optional]   [multiple]   [ ]
org:           [optional]   [multiple]   [inverse key]
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]     [ ]
```

1.2.9 irt

An **irt** object represents a Computer Security Incident Response Team (CSIRT). It includes contact and security information. It may be referenced from **inetnum** or

inet6num objects to show which CSIRT is responsible for handling computer and network incidents for that address range.

It is also used to link "abuse-mailbox:" attributes to **inetnum** and **inet6num** objects.

- The **irt** object name starts with "IRT-".
- The "abuse-mailbox:" attribute specifies the e-mail address to which abuse complaints should be sent. When this attribute is specified no other e-mail address should be used for abuse complaints.
- The "signature:" attribute references a key-cert object representing a CSIRT public key used by the team to sign their correspondence.
- The "encryption:" attribute references a **key-cert** object representing a CSIRT public key used to encrypt correspondence sent to the CSIRT.
- The "auth:" defines an authentication scheme to be used. Any of the current authentication schemes used by the RIPE Database are allowed.
- To add a reference to an **irt** in an **inetnum** or **inet6num** object the authorisation must be passed from one of the "auth:" values in the referenced **irt** object.
- The "irt-nfy:" attribute specifies the e-mail address to be notified when a reference to the **irt** object is added or removed.

Here is a template of an **irt** object:

```
irt:           [mandatory]  [single]     [primary/lookup key]
address:       [mandatory]  [multiple]   [ ]
phone:         [optional]   [multiple]   [ ]
fax-no:        [optional]   [multiple]   [ ]
e-mail:        [mandatory]  [multiple]   [lookup key]
abuse-mailbox: [optional]   [multiple]   [inverse key]
signature:     [optional]   [multiple]   [ ]
encryption:    [optional]   [multiple]   [ ]
org:           [optional]   [multiple]   [inverse key]
admin-c:       [mandatory]  [multiple]   [inverse key]
tech-c:        [mandatory]  [multiple]   [inverse key]
auth:          [mandatory]  [multiple]   [inverse key]
remarks:       [optional]   [multiple]   [ ]
irt-nfy:       [optional]   [multiple]   [inverse key]
notify:        [optional]   [multiple]   [inverse key]
mnt-by:        [mandatory]  [multiple]   [inverse key]
changed:       [mandatory]  [multiple]   [ ]
source:        [mandatory]  [single]     [ ]
```

1.2.10 key-cert

A **key-cert** object is a database public key certificate that is stored on the server. It is used with a **mntner** object for authentication when performing updates. Currently the RIPE Database supports two types of keys.

- For PGP **key-cert** objects, the value of the "key-cert:" attribute must be PGP-"key-id". These keys are compliant with the Open PGP Internet Standard [23].
- For X.509 **key-cert** objects, the database software assigns this value as X.509-n. Here, 'n' is the next available number assigned by the software. If you want to create an X.509 **key-cert** object, you should specify the value as AUTO-xx. If you delete an X.509 **key-cert** object, it is not possible to recreate it with the same name.
- The "method:", "owner:" and "fingerpr:" attributes are all generated by the software. It is not necessary to include these attributes when you create or modify this object. If they are supplied, the software will check the values. If necessary the software will replace the supplied values with generated values. In this case a warning is returned to the user.
- The "certif:" attribute contains the public key. The value of the public key should be supplied either using multiple "certif:" attributes, or in one "certif:" attribute split over several lines. In the first case, this is easily done by exporting the key from your local key ring in ASCII armored format and adding the string "certif:" to the start of each line of the key. 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, as well as the start/end markers and the empty line which separates the header from the key body.

Here is a **key-cert** object template:

```
key-cert:      [mandatory]  [single]     [primary/lookup key]
method:        [generated]  [single]     [ ]
owner:         [generated]  [multiple]   [ ]
fingerpr:      [generated]  [single]     [inverse key]
certif:        [mandatory]  [multiple]   [ ]
org:           [optional]   [multiple]   [inverse key]
remarks:       [optional]   [multiple]   [ ]
notify:        [optional]   [multiple]   [inverse key]
admin-c:       [optional]   [multiple]   [inverse key]
tech-c:        [optional]   [multiple]   [inverse key]
mnt-by:        [mandatory]  [multiple]   [inverse key]
changed:       [mandatory]  [multiple]   [ ]
source:        [mandatory]  [single]     [ ]
```

1.2.11 mntner

Objects in the RIPE Database are protected by using **mntner** objects. A **mntner** object contains the information needed to authorise creation, deletion or modification of any objects that it protects.

- Objects are protected by a **mntner**, if they contain a reference to the **mntner** in the object. This is done by including a "mnt-by:" attribute. Other attributes offer hierarchical protection. The "mnt-by:" attribute is mandatory in all objects except

person and **role** types. Most users set the "mnt-by:' value in a **mntner** to reference itself.

- The "referral-by:" attribute can refer to the **mntner** object itself. The database software does not currently use this attribute, even though it is mandatory to include it.
- Routing Policy System Security specification [2] also defines an "auth-override:" attribute in the **mntner** object template. Together with "referral-by:" attribute, they allow for a **mntner** to be modified if it becomes unresponsive. As this is not part of the core functionality of the RIPE Database, it has not been implemented in the current version of the database. See [2] for more information.
- The "upd-to:" attribute specifies the e-mail address to be notified when an attempt to update an object protected by this mntner is unsuccessful.
- The "mnt-nfy:" attribute specifies the e-mail address to be notified when an object protected by this mntner is successfully updated.
- The "auth:" attribute defines an authentication scheme to be used. . Any of the current authentication schemes used by the RIPE Database are allowed.
- To update an object protected by a **mntner** the authorisation must be passed from one of the "auth:" values in the **mntner** object referenced in the "mnt-by:" attributes of the updated object.
- If an object references more than one **mntner** in the "mnt-by:" attributes they act as a logical 'OR'. If the authorisation is passed by any "auth:" value from any of the referenced **mntner** objects then the update will be authorised.
- The "mnt-lower:", "mnt-routes:" and "mnt-domains:" attributes all provide for hierarchical authorisation. These also work in a logical 'OR' when multiple values are included in an object. How they are used is described in the object descriptions where these attributes are valid.

Here is a **mntner** object template:

```
mntner:        [mandatory]  [single]     [primary/lookup key]
descr:         [mandatory]  [multiple]   [ ]
org:           [optional]   [multiple]   [inverse key]
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]   [inverse key]
remarks:       [optional]   [multiple]   [ ]
notify:        [optional]   [multiple]   [inverse key]
abuse-mailbox: [optional]   [multiple]   [inverse key]
mnt-by:        [mandatory]  [multiple]   [inverse key]
referral-by:   [mandatory]  [single]     [inverse key]
changed:       [mandatory]  [multiple]   [ ]
source:        [mandatory]  [single]     [ ]
```

1.2.12 organisation

The **organisation** object provides information about an organisation entity that has registered a network resource in the RIPE Database. This entity may be a company, non profit group or individual.

- "organisation:" specifies the ID of the **organisation** object. This is set by the database software. It is only used as a reference label. When creating an organisation this should be set to "AUTO-n <optional letter Combination>", where n is any number.
- "org-name:" attribute defines the name of the organisation.
- "org-type:" specifies the type of an organisation and can be.one of these:
  - IANA
  - RIR
  - NIR (Note - there are no NIRs in the RIPE NCC service region.)
  - LIR
  - OTHER
- Users can only set the "org-type:" to OTHER. All other types are only for use by the Database Administrators.
- The "org:" attribute is used to reference an **organisation** object for an entity that has registered the resource or other data in which this reference is made.
- The "ref-nfy:" attribute specifies the e-mail address to be notified when a reference to the **organisation** object is added or removed.
- The "mnt-ref:" attribute specifies the maintainer objects that are entitled to authorise the addition of references to the **organisation** object in other objects.
- If the **organisation** object includes more than one "mnt-ref:" attribute they act as a logical 'OR'. If the authorisation is passed by any **mntner** object referenced in any "mnt-ref:" attribute then the update will be authorised.
- The "abuse-mailbox:" attribute specifies the e-mail address to which abuse complaints should be sent. When this attribute is specified no other e-mail address should be used for abuse complaints.

Here is an **organisation** object template:

```
organisation:   [mandatory]  [single]     [primary/look-up key]
org-name:       [mandatory]  [single]     [lookup key]
org-type:       [mandatory]  [single]     [ ]
descr:          [optional]   [multiple]   [ ]
remarks:        [optional]   [multiple]   [ ]
address:        [mandatory]  [multiple]   [ ]
phone:          [optional]   [multiple]   [ ]
fax-no:         [optional]   [multiple]   [ ]
e-mail:         [mandatory]  [multiple]   [lookup key]
org:            [optional]   [multiple]   [inverse key]
admin-c:        [optional]   [multiple]   [inverse key]
tech-c:         [optional]   [multiple]   [inverse key]
```

```
ref-nfy:        [optional]   [multiple]   [inverse key]
mnt-ref:        [mandatory]  [multiple]   [inverse key]
notify:         [optional]   [multiple]   [inverse key]
abuse-mailbox:  [optional]   [multiple]   [inverse key]
mnt-by:         [mandatory]  [multiple]   [inverse key]
changed:        [mandatory]  [multiple]   [ ]
source:         [mandatory]  [single]     [ ]
```

1.2.13 peering-set

A **peering-set** object defines the set of peerings that appear in the "peering:" or "mp-peering:" attribute.

The "peering-set:" attribute defines the name of the set. It is an RPSL name that starts with 'prng-'.

- The "peering:" attribute defines a peering that you can use to import or export routes.
- The "mp-peering:" attribute extends the "peering:" attribute and defines a multiprotocol peering that can be used to import or export IPv6 routes.
- The "peering:" and "mp-peering:" attributes are optional. However, a peering-set must contain at least one of these two attributes. It cannot contain both in the same object.
- The name of a **peering-set** object can be hierarchical. A hierarchical peering-set name is a sequence of peering-set names and AS Numbers separated by colons. At least one part of the name must be an actual **peering-set** name (i.e. start with "prng-"). All the set name components of a hierarchical **peering-set** have to be **peering-set** names.
- This object sets the authorisation required for the creation of other **peering-set** objects one level down in a hierarchy. This is set by the "mnt-lower:" and "mnt-by:" attributes.

Here is a **peering-set** object template:

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

1.2.14 person

A **person** object contains information about the technical or administrative contact(s) responsible for an object. After it has been created, the "person:" attribute cannot be changed by users. Under some circumstances it can be changed by the Database Administrator.

- The person object is identified by the "nic-hdl:" attribute. This is a label, usually made up from the initials of the person's name and the database "source:" (for example, DW-RIPE).
- The "nic-hdl:" can use an international country code instead of the database source as the suffix or omit the suffix (for example DW-NL or just DW).
- The user can specify the "nic-hdl:" when the object is created or can use the "AUTO-n<optional letter Combination>" construction, where n is any number.
- The "nic-hdl:" attributes of the **person** and **role** objects share the same name space in the database. So you cannot create a **person** and **role** using the same "nic-hdl:". The "nic-hdl:" must be unique across both object types.
- The "address:" attribute is the full postal address of the contact in free format text.
- The "phone:" attribute specifies a telephone number of the contact in international shorthand. It must start with a '+' followed by the international country code, area code and number.
- The "fax-no:" attribute has the same format as the "phone:" attribute.
- The "e-mail:" attribute represents the contact's e-mail address as defined by RFC 2822 [8].
- The "remarks:" attribute can be any free format text.
- The "notify:" attribute specifies the e-mail address to which notifications of changes to an object should be sent.
- The "abuse-mailbox:" attribute specifies the e-mail address to which abuse complaints should be sent. When this attribute is specified no other e-mail address should be used for abuse complaints.
- The "mnt-by:" attribute specifies the identifier of a registered **mntner** object used for authorisation of operations performed on the object that contains this attribute. The **mntner** object must exist in the database before it can be referenced in other objects.
- The "changed:" attribute is for the user's own reference. It must contain an e-mail address and a timestamp. If the timestamp is not included the software will add the current timestamp at the time of the update. Nothing can be reliably determined about the object or its change history by looking at the "changed:" attributes.
- There must be at least one "changed:" attribute in any object. If there are more they must be in ascending date order.
- Any changed "attribute:" can be modified or deleted by the user as long as at least one remains.
- The "source:" attribute specifies the registry where the object is registered. This should be "RIPE" for the RIPE Database.

Here is a **person** object template:

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

1.2.15 poem

A **poem** object contains a poem that is submitted by a user. This object is included in the database to show that engineers do have a sense of humour.

- The "poem:" attribute specifies the title of the poem.
- The "form:" attribute specifies the identifier of a registered poem type.
- The "text:" attribute specifies the body of the poem. It must be humorous, but not malicious or insulting. It should be written in the style of the "form:".
- The "author:" attribute is the "nic-hdl:" of the person who entered the poem.

Here is a **poem** object template:

```
poem:           [mandatory]  [single]     [primary/look-up key]
descr:          [optional]   [multiple]   [ ]
form:           [mandatory]  [single]     [inverse key]
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]     [ ]
```

1.2.16 poetic-form

A **poetic-form** object defines the supported poem types.

- The "poetic-form:" attribute starts with "FORM-". It is followed by the name of an internationally recognised poetic format of humorous writing. For example, limerick or English-sonnet.

- The "descr:" attribute describes the style of the poetic form, written in the form style. For example, if it is a FORM-LIMERICK, the description will be written as a limerick.
- This object cannot be created automatically. It will be forwarded to the Database Administrators for approval of the content.

Here is a **poetic-form** object template:

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

1.2.17 role

A **role** object is similar to a **person** object. However, instead of describing a single person, it describes a role performed by one or more people. This might be a help desk, network monitoring centre, system administrator etc. A **role** object is useful since often a person performing a specific function may change while the role itself remains.

- If one person needs to be referenced in a large number of objects, it is better to use a **role** object and then place that person in the **role** object. If that person leaves your company, it is simple to modify the **role** object.
- The **role** object is identified by the "nic-hdl:" attribute. This is a label, usually made up from the initials of the function and the database "source:" (for example, CREW-RIPE).
- The "nic-hdl:" can use an international country code instead of the database source as the suffix or omit the suffix (for example CREW-NL or just CREW).
- The user can specify the "nic-hdl:" when the object is created or can use the "AUTO-n<optional letter Combination>" construction, where n is any number.
- The "nic-hdl:" attributes of the **person** and **role** objects share the same name space in the database. You cannot create a **person** and **role** using the same "nic-hdl:". The "nic-hdl:" must be unique across both object types.
- After you have created a **role** object, the "role:" attribute cannot be changed by users. Under some circumstances it can be changed by the Database Administrator.
- Under exceptional circumstances the Database Administrator can convert a **person** object into a **role** object.

Here is a **role** object template:

```
role:          [mandatory]  [single]     [lookup key]
address:       [mandatory]  [multiple]   [ ]
phone:         [optional]   [multiple]   [ ]
```

```
fax-no:        [optional]   [multiple]   [ ]
e-mail:        [mandatory]  [multiple]   [lookup key]
org:           [optional]   [multiple]   [inverse key]
admin-c:       [mandatory]  [multiple]   [inverse key]
tech-c:        [mandatory]  [multiple]   [inverse key]
nic-hdl:       [mandatory]  [single]     [primary/lookup key]
remarks:       [optional]   [multiple]   [ ]
notify:        [optional]   [multiple]   [inverse key]
abuse-mailbox: [optional]   [multiple]   [inverse key]
mnt-by:        [optional]   [multiple]   [inverse key]
changed:       [mandatory]  [multiple]   [ ]
source:        [mandatory]  [single]     [ ]
```

1.2.18 route6

Each interAS route (also known as an interdomain route) originated by an Autonomous System can be specified by using a **route6** object for IPv6 addresses

It is possible to create **route6** objects in the RIPE Database for address space registered in other RIR regions. You will first need to create an **aut-num** object to represent the Autonomous System in the RIPE Database.

- The "route6:" attribute is the IPv6 address prefix of the route.
- The "origin:" attribute is the AS Number of the Autonomous System that originates the route into the interAS routing system. The corresponding **aut-num** object for this Autonomous System must already be registered in the RIPE Database.
- The "route6:" and "origin:" attribute pair make up the class primary key.
- The following attributes have the same meaning as described for the **route** object in Section 1.2.19.

Here is a **route6** object template:

```
route6:         [mandatory]  [single]     [primary/look-up key]
descr:          [mandatory]  [multiple]   [ ]
origin:         [mandatory]  [single]     [primary/inverse key]
holes:          [optional]   [multiple]   [ ]
org:            [optional]   [multiple]   [inverse key]
member-of:      [optional]   [multiple]   [ ]
inject:         [optional]   [multiple]   [ ]
aggr-mtd:       [optional]   [single]     [ ]
aggr-bndry:     [optional]   [single]     [ ]
export-comps:   [optional]   [single]     [ ]
components:     [optional]   [single]     [ ]
remarks:        [optional]   [multiple]   [ ]
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]     [ ]
```

1.2.19 route

Each interAS route (also known as an interdomain route) originated by an Autonomous System can be specified by using a **route** object for IPv4 addresses.

It is possible to create **route** objects in the RIPE Database for address space registered in other RIR regions. You will first need to create an **aut-num** object to represent the Autonomous System in the RIPE Database.

- The "route:" attribute is the address prefix of the route.
- The "origin:" attribute is the AS Number of the Autonomous System that originates the route into the interAS routing system. The corresponding **aut-num** object for this Autonomous System must already be registered in the RIPE Database.
- The "route:" and "origin:" attribute pair make up the class primary key.
- The "holes:" attributes list the component address prefixes that are not reachable through the aggregate route (perhaps that part of the address space is unallocated).
- The "member-of:" attribute can be used in the **route**, **route6**, **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.
- The "inject:" attribute specifies which routers perform the aggregation and when they perform it.
- The "aggr-mtd:" attribute specifies how the aggregate is generated.
- The "aggr-bndry:" attribute defines a set of Autonomous Systems, which form the aggregation boundary.
- The "export-comps:" attribute defines the set's policy filter, a logical expression which when applied to a set of routes returns a subset of these routes.
- The "components:" attribute defines what component routes are used to form the aggregate.
- This object sets the authorisation required for the creation of more specific **route** objects within the range specified by this **route**. This is set by the "mnt-lower:" and "mnt-by:" attributes.
- This object sets part of the authorisation required for the creation of a more specific **route** object. This is set by the "mnt-routes:", "mnt-lower:" and "mnt-by:" attributes.
- The "mnt-routes:" attribute can include an optional list of prefix ranges inside of curly braces ("{}") or the keyword "ANY". This should follow after the reference to the maintainer. The default, when no additional set items are specified, is "ANY" or all more specifics. Please refer to RFC-2622 [1] for more information.

Here is a **route** object template:

```
route:         [mandatory]  [single]     [primary/lookup key]
descr:         [mandatory]  [multiple]   [ ]
origin:        [mandatory]  [single]     [primary/inverse key]
holes:         [optional]   [multiple]   [ ]
org:           [optional]   [multiple]   [inverse key]
member-of:     [optional]   [multiple]   [ ]
inject:        [optional]   [multiple]   [ ]
aggr-mtd:      [optional]   [single]     [ ]
aggr-bndry:    [optional]   [single]     [ ]
export-comps:  [optional]   [single]     [ ]
components:    [optional]   [single]     [ ]
remarks:       [optional]   [multiple]   [ ]
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]     [ ]
```

1.2.20 route-set

A **route-set** object is a set of route prefixes and not a set of database **route** objects. The "route-set:" attribute defines the name of the set. It is an RPSL name that starts with "rs-".

- It defines a set of routes that can be represented by **route** objects or by address prefixes.

- In the case of **route** objects, the set is populated by means of the "mbrs-by-ref:" attribute. In the case of address prefixes, 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.
  - The "mp-members:" attribute is a list of IPv6 address prefixes or other route-set names.
- The name of a **route-set** object can be hierarchical. A hierarchical route-set name is a sequence of route-set names and AS numbers separated by colons. At least one component of such a name must be an actual route-set name (i.e. start with "rs-").
- The "mbrs-by-ref:" 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.
- This object sets the authorisation required for the creation of other **route-set** objects one level down in a hierarchy. This is set by the "mnt-lower:" and "mnt-by:" attributes.

Here is a **route-set** object template:

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

1.2.21 rtr-set

A **rtr-set** object defines a set of routers.

- A set may be described by the "members:" attribute that is a list of inet-rtr names, IPv4 addresses or other rtr-set names. The "mp-members:" attribute extends the "members:" attribute to use IPv6 addresses.
- 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 "rtr-set:" attribute defines the name of the set. It is an RPSL name that starts with "rtrs-".
- The name of a **rtr-set** object can be hierarchical. A hierarchical rtr-set name is a sequence of rtr-set names and AS Numbers separated by colons. At least one component of such a name must be an actual rtr-set name (i.e. start with "rtrs-").
- This object sets the authorisation required for the creation of other **rtr-set** objects one level down in a hierarchy. This is set by the "mnt-lower:" and "mnt-by:" attributes.

Here is a **rtr-set** object template:

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

2.0 Updates in the RIPE Database

To create, modify or delete an object you need to send an update message to the database.

There are two databases that you can update. The RIPE Database is the authoritative database. There is also a RIPE TEST Database. This operates in the same way but contains only test data. The test data is cleaned out at the start of each month and a pre-determined set of basic objects is re-inserted. The TEST Database can be used to learn how to update the database and try out 'what if' scenarios. As there are fewer restrictions, this allows you to create encompassing or parent objects that you may need for your tests. It is good practise to delete any objects you created after your tests. This avoids blocking access for other users.

You can submit updates to either database using two access methods.

- By sending an e-mail.
- By using the synchronous interface. There are two ways to do this:
  - By using a client to access the "syncupdates" interface
  - By using the "webupdates" interface from the www.ripe.net website [22]. This is a cgi interface to the syncupdates interface

Although the Application Programming Interface (API) is different, the main content of the update message and the structure of the objects to be updated is the same with both access methods. The pre-processing done by dbupdate (the update part of the database software) differs for the two access methods.

The original method of access was by e-mail. When the synchronous method was introduced, the interface was made to be as compatible as possible with the e-mail interface. This meant simulating the e-mail subject line keywords with special flags in the synchronous interface. But later additional keywords are only available using the e-mail update method.

Currently updates are split approximately equally between e-mail and synchronous update methods.

2.0.1 E-mail update method

E-mail update messages may be in plain text or MIME encoded. If encoded, the database will treat each valid MIME part as a separate message. But where messages are multipart/signed the signature part will be associated with the corresponding MIME text part. Please see Section 2.3.1, 'MIME Support' for more information. An update message, or MIME part, may contain more than one object.

To submit an e-mail update to the RIPE Database send an e-mail to auto-dbm@ripe.net.
To submit an e-mail update to the TEST Database send an e-mail to test-dbm@ripe.net.

E-mail update messages are placed by the database software into a series of queues depending on the size and content of the message. Each queue is handled sequentially on a "first in, first out" basis.

Acknowledgements from the database are returned to the sender of the e-mail based on the "Reply-to:" and "From:" fields in the e-mail header. Notifications may also be sent based on e-mail addresses found in the attributes within each object in the update message.

There are no restrictions on the number of objects in any one e-mail message. But there is a limit on the total size of an update message. It is up to the user to determine how many objects to include in an update message. There are some points to consider.

- Every e-mail update message will result in one acknowledgement e-mail message returned.
- Each object in an update message may generate several notification e-mail messages.
- If you update 1000 objects by sending 1000 e-mails, each containing one object, this will cause a large number of e-mails to be returned. Your mail system needs to be able to handle the volume of e-mails generated in this way.
- The acknowledgement and notifications are not sent until all the objects in an update message have been processed.
- If you submit one update message containing 1000 objects, no response will be received until all the objects have been processed.
- You need to optimise the number of update messages and number of objects per update to suit your business practises.
- Occasionally an update message causes system problems. The Database Administrator will monitor the progress of such an update. In rare occasions it is necessary for the Database Administrator to intervene in the process.

2.0.2 Synchronous update method

The synchronous update method allows near instant updating of the RIPE Database. It is designed for applications that need to update the RIPE Database and see an immediate change. The syncupdates front end is a cgi script that accepts input from either a POST or a GET request. It returns the response header and an acknowledgement (if any) in text/plain format. The protocol is http1.1, specified in RFC 2068 [24]. See the separate document for details of the protocol for communicating with syncupdates [20].

There is no maximum amount of time for an update operation to be completed. You should set your timeout to the highest possible value. Problems sometimes occur with reverse **domain** objects. Because of all the DNS checks made, reverse **domain** objects sometimes take longer to process, and your connection may occasionally timeout. The update will still be completed by the database server, but no acknowledgement will be returned if the connection has timed out. Notification messages are always sent by e-mail, and will be sent regardless of your connection status.

Synchronous update messages are placed by the database software into a series of queues depending on the size and content of the message. Each queue is handled sequentially on a "first in, first out" basis.

There are sample clients to use with syncupdates which access either the RIPE Database or the TEST Database. Although they are sample clients they can be used 'as is' from our web site [21]. There is also a short perl script which you can use as a client to access syncupdates [21].

You can type the objects directly into the text area of this web form. Or you can have them prepared in another file and cut and paste them into this web form. There is no limit to the number of objects you can enter in one submission. But the same principles apply as described in Section 2.0.1, 'E-Mail Update Method'. The main processing of an update message is the same for any method of submission. So the rules about responses and notification e-mails are the same. With syncupdates, if you enter many objects your client connection may time out waiting for the final response after processing all the objects.

Webupdates is run as a cgi script on our website [22]. You can select either the RIPE Database or TEST Database by setting the source. Details of how to use webupdates can be found in the "RIPE Database User Manual: Getting Started" [5].

Webupdates is normally used for individual object updates. Multiple objects can be submitted by switching the view to use the text input area similar to syncupdates. If more than one object is entered into the text area it is then not possible to switch view back to the attribute working mode.

2.1 Format of an Update Message

The format of the body of an update message is the same regardless of the access method. It contains one or more objects. The structure of the objects is described in Section 1.1, 'Object Representation'.

Object definition starts with the class attribute and ends with the first blank line ("\n\n"). You cannot use a blank line in the object, as the software will read this as being the end of the object.

We apply a heuristic method to each paragraph of text in the input to determine if it is an object. Any part of the message that is not recognised as a database object is ignored. These parts are grouped together at the end of the acknowledgement message.

2.2 Creating, Modifying and Deleting an Object

To create, modify or delete objects, you need to send a message to the database by one of the access methods. This message must contain one or more database objects. Table 1.1, 'Object Types Supported by the RIPE Database' lists all the object types that are recognised by the database. No other object types can be created. You must use the object templates described in Section 1.2, 'Object Types' to specify the objects. Each instance of an object must contain at least one of each of the mandatory attributes for that object type. An object can contain zero or more instances of each available optional attribute for that object type. One message may contain several objects, even if they each require different operations, for example, creation, modification or deletion.

2.2.1 Object Processing

As a rule, the order of objects in the message is not changed. The database processes objects one by one, starting with the first recognised object in the message. It is the user's responsibility to order the objects in the message to make sure that all references can be resolved. The only exception to this is when "AUTO NIC handles" are used. These generate an automatically assigned value for the "nic-hdl:" attribute in the **person** or **role** objects. When the database software finds an object that references an "AUTO NIC handle", this object is placed on an internal queue for later processing. This is to ensure that all objects containing "AUTO NIC handle" are processed before any other object that can reference them. This internal queue is processed after the first pass through all objects in the update message.

It is recommended that you avoid complex arrangements of auto generated values in a single update message. An example of a complex arrangement might be an update containing a **person** object with a "nic-hdl:" value of <auto-1>, a **role** object with a "nic-hdl:" value of <auto-2> and referencing the **person** <auto-1> and a **mntner** referencing the **role** object <auto-2>. This can be made to work with careful ordering of the objects in the update message, but the wrong ordering will cause part of this update to fail. Such an update is better done with multiple update messages. "Keep it simple" and you will avoid many potential problems.

When processing each individual object, the software makes many checks including that:

- The syntax of the object is correct.
- The object passes all required authorisation checks.
- All references to other objects can be resolved without conflicts.
- The operation does not compromise referential integrity. For example, when an object is to be deleted, the server checks that it is not referenced from any other object in the RIPE Database.
- The requested NIC handle is not in use and can be allocated. This is done only for the creation of **person** or **role** objects that request a particular NIC handle. Note

that you cannot create a **person** object with a "nic-hdl:" that is already in use by a **role** object, and vice versa.
- The object complies with relevant policies. For example the "status:" value in **inetnum** objects.

If all checks are successful, the server processes the operation on the object in the RIPE Database. If one of these steps fails, the operation fails for the object as a whole. This is shown in the acknowledgement message and sometimes in notification messages.

Each object in the update message is processed independently of others, so even if one operation fails, the following objects will still be processed. There may, however, be consequences of a previous failed operation. For example if a **person** object creation fails, a later object creation which references this **person** object will fail the referential integrity checks because the **person** object does not exist in the database.

After the software finishes processing all the objects in the update message, an acknowledgement message is returned to the sender of the original update. For e-mail update messages, this will be as specified in the "Reply-to:" field or "From:" field; for synchronous update messages it will be returned via the open connection.

The database server may also send notification messages. See Section 2.5.2, 'Notifications' for more information about this.

2.2.2 Creating a New Object

If the database does not contain an object with the same class primary key as the object in the update message, the server will assume that you want to create it (remembering that a **person** and **role** object cannot use the same NIC handle).

To create **person** and **role** objects, you can use "AUTO NIC handles". If you do this, then the server will automatically assign a NIC handle.

To do this, the value of the "nic-hdl:" attribute should be:

```
nic-hdl:      AUTO-<digit>[<initials>]
```

If you specify the <initials> (between two and four characters), then the server will try to use them to make the NIC handle. If you forget to include <initials>, the server will take the initials from the "person:" or "role:" attribute.

If you want to create an **organisation** object, you must specify the ID of the object as AUTO-<digit>[<initials>]. The database software will then assign an appropriate ID by using the initials of the "org-name:" attribute of the object. If you prefer, you can specify the letter combination you would like to use.

For example, if you want TTR as the letter combination in the organisation ID, you should put AUTO-1TTR into the "organisation:" attribute, when you create the object. If you delete an **organisation** object you cannot re-create one with the same ID as the one you deleted.

2.2.3 Modifying an Existing Object

If an object type with the same class 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 two versions, white space characters are not considered.

2.2.4 Deleting an Object

You can delete an object by adding a "delete:" pseudo attribute to the object.

```
delete: <comment>
```

The software will only accept this request if the object in the message is exactly the same as the one in the database which is to be deleted. When comparing the versions, white space characters are not considered. The operation will fail if the object is referenced from any other object in the database.

This pseudo attribute applies to one object only in an update message. It must be a part of the object in the update message that is to be deleted. It can be added at any point within the object or immediately after the object. It cannot be placed before the object (this would result in the object not being recognised by the database software as a valid type).

Objects can still be deleted from the database even if they are not syntactically correct. This allows for old objects to be deleted long after the syntax has been changed.

2.3 E-mail Updates

2.3.1 MIME Support

The database software supports MIME. This means that you can cryptographically sign an update message using e-mail agents that attach the signature in a separate MIME part, not in the body of the message.

It also allows the definition of scopes of authorisation within the message (for example, parts where different passwords apply) and nested signing of messages. This 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.

The software will recognise the following headers and take the appropriate actions:

- multipart/signed
- multipart/alternative
- multipart/mixed
- multipart/unknown
- application/pgp-signature
- application/x-pkcs7-signature
- application/pkcs7-signature
- text/plain

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

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

- Authorisation information is valid only within a single part
- AUTO NIC handle assignment is made only within a single part (see Section 2.3.2, 'PGP and X.509 support')

2.3.2 PGP and X.509 Support

The database supports PGP and X.509 signed messages. The following rules apply when submitting updates using these authorisation schemes.

- When using MIME encapsulation a 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 signature. For a PGP signature, it is encapsulated with application/pgp-signature MIME discrete type. For an X.509-signature it is encapsulated with either application/x-pkcs7-signature or application/pkcs7-signature MIME discrete types.

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

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

2.3.3 Subject Line Processing

The subject line may have a special meaning in e-mail update messages. Some keywords can be used in certain circumstances. Sometimes users also want to use the subject line for their own reference. All keywords are case insensitive. The available keywords are:

- NEW
- HELP or HOWTO

- DIFF
- KEYWORDS:

One way to use a keyword is to put it in the subject line of the e-mail message, with NO other words present. If any other word is found (for example, Subject: NEW objects) then none of the words will be treated as keywords. All words in the subject line will be reported in the acknowledgement reply as invalid keywords, along with a WARNING message. In this context it is impossible for dbupdate to know if a word is a keyword.

Many users often include their own references in the subject line. Using this method it is not possible to also use a keyword. The user's references are reported in the WARNING message as invalid keywords.

Another way to use keywords is with the KEYWORDS: tag. When this tag is found in a subject line all text up to and including the KEYWORDS: tag is ignored by dbupdate. Only the text following this tag is checked against the list of valid keywords. The same rules apply as before: if a word that is not a valid keyword is found after the KEYWORDS: tag, none of the words are actioned as keywords. In this case all the words following the KEYWORDS: tag are reported as invalid keywords in the acknowledgement reply WARNING message.

Adding the KEYWORDS: tag at the end of the Subject: line with no keywords following it means that the user references will not be reported as invalid keywords. This allows you to avoid receiving a WARNING message.

2.3.3.1 NEW Keyword

Use NEW keyword if you want the database to only accept new objects. In this case all objects found in the update are assumed to be creation operations. If an object already exists in the database, that object will result in an error message in the acknowledgement.

2.3.3.2 HELP (HOWTO) Keyword

The HELP keyword causes a help text to be returned in the acknowledgement that contains information about how to query and update the database. When this keyword is used the body of the message is ignored. HOWTO has the same effect as HELP.

2.3.3.3 DIFF Keyword

The DIFF keyword will highlight changes in the notification message between the old and new objects for a modification operation. This is particularly useful when "import:" and "export:" attributes are changed in large **aut-num** objects. In this case the subject line may contain this:

Subject: changes to aut-num: as1234 KEYWORDS: diff

When the DIFF keyword is used each object in a notification message that has been modified will include a section specifying the changes made, followed by the old version of the object, and finally the new version of the object. The output will have the difference listing followed by the old and new objects.

The difference is the standard unix diff, but with one slight change. In the acknowledgement and notification messages three dashes (---) followed by a new line signifies the start of a section relating to an object. This is to make it easy to parse the output and find the start of each object in the message. Note that the standard unix diff also uses --- to separate lines that have changed, so we have replaced --- with === in the diff output presented in the notification messages.

Using a simple **person** object as the example, the output is as follows:

```
---
OBJECT BELOW MODIFIED:

Differences in [person] TP10-DB-TEST

7c7,8
< notify: case040-1@localhost
===
> changed: dbtest@localhost 20040101
> notify: case040-2@localhost

person: Test Person
address: Singel 258
address: Amsterdam
phone: +31 20 535 4444
nic-hdl: TP10-DB-TEST
changed: dbtest@localhost 20020101
notify: case040-1@localhost
source: DB-TEST


REPLACED BY:

person: Test Person
address: Singel 258
address: Amsterdam
phone: +31 20 535 4444
nic-hdl: TP10-DB-TEST
changed: dbtest@localhost 20020101
changed: dbtest@localhost 20040101
notify: case040-2@localhost
source: DB-TEST
```

2.4 Acknowledgements and Notifications

2.4.1 Acknowledgements

One acknowledgement message (ACK) is returned to the user for each update received. This is split into sections.

The first section shows where the update was received from. This may be a copy of the update e-mail header or the IP address for a synchronous connection. This section also includes a summary of the update results, explaining how many objects were recognised by the database software, how many operations were successful and how many failed. The subject line of the ACK message states "SUCCESS" or "FAILED". If the update message contains no objects or any one of the operations fails, the ACK reports "FAILED". Otherwise it reports "SUCCESS". Following this status is the original subject line.

An example first section would look like this:

```
From: RIPE Database Administration <ripe-dbm@ripe.net>
To: admin@here.com
Subject: SUCCESS: UPDATE person
Reply-To: ripe-dbm@ripe.net



>  From:       admin@here.com
>  Subject:    UPDATE person
>  Date:       Wed, 28 Mar 2007 01:00:06 +0300 (EEST)
>  Reply-To:   admin@here.com
>  Message-ID: <20070327170006.983D6124225@here.com >



SUMMARY OF UPDATE:

Number of objects found:           2
Number of objects processed successfully:  1
  Create:        1
  Modify:        1
  Delete:        0
  No Operation:   0
Number of objects processed with errors:   0
  Create:        0
  Modify:        0
  Delete:        0
  Syntax Errors:  0
```

For a synchronous update the details of where the update was received from would look like this:

```
- From-Host: 193.0.0.1
- Date/Time: Wed Mar 28 00:17:29 2007
```

The next section is the "DETAILED EXPLANATION". This is split into three parts. The first part lists all the objects where the operation failed. This will include "ERROR" messages as well as possible "INFO" and "WARNING" messages. The next part shows all the operations that were successful. This may include additional "INFO" and "WARNING" messages with each operation listed. The third part lists all the paragraphs from the update message that were not recognised as objects. Each part is separated in the ACK by a line containing several '~' characters. Within each part the objects and paragraphs are listed in the order they were processed. This is generally the order they appear in the update message unless AUTO-n nic-hdls are referenced. The line before each object contains three '-' characters. This allows for easy parsing by script.

An example of this section would look like this:

```
DETAILED EXPLANATION:


~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The following object(s) were found to have ERRORS:


---
Update FAILED: [person] dn1172-ripe         de nis
***Error: Syntax error in object

person:         de nis
address:         here
phone:          1
***Error: Syntax error in "1"
nic-hdl:        dn1172-ripe
changed:        admin@here.com 20070327
source:         ripe

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The following object(s) were processed SUCCESSFULLY:


---
```

```
Modify SUCCEEDED: [person] dn1172-ripe          de nis

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The following paragraph(s) do not look like objects
and were NOT PROCESSED:

This is a private e-mail.
It is intended only for the named recipient.


~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

There are many reasons why the operation may fail for the object, including syntax error, authorisation failure and failed policy check.

2.4.2 Notifications

There are a number of attributes that may cause notification messages to be generated:

- "notify:"
- "mnt-nfy:"
- "upd-to:"
- "irt-nfy:"
- "ref-nfy:"

The "notify:" attribute is an option in most object types, and is used when an object is successfully updated. The "notify:" attribute of the old version of the object is used if the object is being modified or deleted. The "notify:" attribute of the new version of the object is used if the object is being created. If there are multiple "notify:" attributes, an e-mail will be sent to the addresses contained in all of these, subject to the above conditions on the operation. If the update fails for any reason then no notifications will be sent to any "notify:" e-mail addresses.

The "mnt-nfy:" and "upd-to:" attributes can only be included in **mntner** objects. "upd-to:" is mandatory and "mnt-nfy:" is optional. These are used to inform users of successful updates to a maintained object or attempted updates where the authorisation failed.

When a maintained object is updated successfully a notification message will be sent to e-mail addresses contained in the "mnt-nfy:" attributes of the **mntner** objects. If the updated object has one or more "mnt-by:" attributes, notifications will be sent to all the e-mail address listed in all the "mnt-nfy:" attributes in all the referenced **mntner** objects.

When an **inet(6)num**, **route(6)** or a **domain** object is created, authentication is required from the parent object. If the parent has a "mnt-lower:" (or "mnt-routes:" or "mnt-domains:") attribute, this is the **mntner** that will need to be authenticated against. Otherwise the parent "mnt-by:" attribute be used. The e-mail address listed in all the "mnt-nfy:" attributes of all the parent's appropriate **mntner** objects will be notified.

When an update to a maintained object fails the authentication, the notifications are sent to the e-mail address contained in the "upd-to:" attributes. The rules for finding the appropriate "upd-to:" attributes are the same as for the "mnt-nfy:" above.

The "irt-nfy:" attribute is only allowed in an **irt** object and is optional. This is used when a reference to an **irt** object is added to or removed from an **inetnum** or **inet6num** object (by means of "mnt-irt:" attribute). If the **irt** object contains one or more "irt-nfy:" attribute(s), all the e-mail addresses listed in all the "irt-nfy:" attributes will be sent a notification.

The "ref-nfy:" attribute is only allowed in an **organisation** object and is optional. This is used when a reference to an **organisation** object is added to an object (by means of "org:" attribute). If the **organisation** object contains one or more "ref-nfy:" attribute(s), all the e-mail addresses listed in all the "ref-nfy:" attributes will be sent a notification.

The format of a notification message is similar to the ACK message. The first section explains why you are being sent this notification. The next section has the e-mail header or IP address details showing where the update came from. The final section shows the changes that were contained in the update message, if it was successful. If it failed for authorisation reasons, it shows the object for which a change was attempted, but not the actual change details.

A notification message is only sent to a single e-mail address. It will contain all the notification details of objects from an update message that relate to that e-mail address. If the same details need to be sent to two different e-mail addresses, then two separate e-mails will be generated by the database software. This ensures that when an update contains many objects covered by overlapping notification e-mail addresses, only the appropriate details are sent to each e-mail address.

2.5 Protecting Data

The RIPE Database provides mechanisms to protect objects and control who can make changes to them. In some cases there are also restrictions over who can create certain objects.

- Authentication is the way we determine who is attempting to make a change.
- Authorisation is how we decide whether a transaction passing a specific authentication check is allowed to perform a given operation.

Different types of objects in the database require different levels of protection. Authentication based on strong encryption is the preferred method, however, this may not always be legally available. For this reason, the server supports multiple authentication methods.

2.5.1 Authorisation Model

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

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

When submitting an update that requires authorisation, the authentication information valid for one of the authentication tokens of one of the relevant **mntner** objects should be supplied. Different methods require different authentication information, as shown below.

Authentication methods currently supported include the following:

Method	Description
MD5-PW	This scheme is based on the MD5 hash algorithm. The authentication information stored in the database is a passphrase encrypted using md5-crypt algorithm, which is a concatenation of the "1" string, the salt, and the 128-bit hash output. Because it uses an 8-character salt and an almost unlimited pass phrase, this scheme is quite stable against dictionary attacks. However, since the encrypted form is exposed it cannot be considered as a strong form of authentication. Authentication information is supplied using a "password:" pseudo-attribute. The value of this attribute is a clear-text pass phrase. It can appear anywhere in the body of the message, but not within mail headers. Line continuation is not allowed for this attribute. The attribute and the pass phrase should fit on one line. If you split the pass phrase across multiple lines this will be treated as a syntax error.
PGPKEY	This is a strong form of authentication. The authentication information is a signature identity pointing to a public key certificate, which is stored in a separate **key-cert** object. The user 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. Anyone can supply any public keys with any ownership information to the database. These keys can be used to protect other objects by checking that the update comes from a user who knows the corresponding secret key.
X.509	This is another strong form of authentication. It works in the same way as PGPKEY, but uses an X.509 certificate as the key.

2.5.2 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 **mntner** objects can be used to protect one object.

Only those **mntner** objects referenced by the "mnt-by:" attributes are authorised to modify or delete the object. Note that authentication checks are logically OR-ed (e.g. A or B or C). If the information required by at least one authentication token from one **mntner** object is supplied, the operation will be authorised. That means that object protection is as weak as the weakest authentication method used in the **mntner** objects referenced by an 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 at least one of the **mntner** objects referenced by one of the "mnt-by:" attributes.

If the operation is a modification and the old object already has one or more "mnt-by:" attributes, then one of the **mntner** objects referenced in one of the "mnt-by:" attributes in the old object must authenticate the change. If the old object does not have any "mnt-by:" attributes, then one of the **mntner** objects referenced in one of the "mnt-by:" attributes in the new object must authenticate the change.

If the operation is a creation then one of the **mntner** objects referenced in one of the "mnt-by:" attributes in the new object must authenticate the change.

2.5.3 Protection of aut-num Object Space

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

This parent authorisation is only required when an object is created. It is in addition to the authorisation of the individual object itself.

2.5.4 Protection of Address Space

The **inetnum** and **inet6num** objects represent address space allocations and assignments. The "mnt-lower:" attribute is used to reference a **mntner** object that authorises the creation of more specific **inetnum** or **inet6num** objects. If no "mnt-lower:" attribute is specified, one of the "mnt-by:" attributes of the parent object must be used instead.

This parent authorisation is only required when an object is created. It is in addition to the authorisation of the individual object itself.

2.5.5 Protection of Route Object Space

The **route** object creation must satisfy several authentication criteria.

It must match the authentication specified in the **aut-num** object referenced by the "origin:" attribute of the **route** object submission.

It must also match the authentication specified in an exact match **route** object, if one exists. If none exist, it must use a **route** object with the longest prefix match that is less specific than the prefix of the **route** object submission, if one exists. If no applicable **route** object exists, then an exact match **inetnum** will be used, if one exists. If one does not exist it must use the most specific **inetnum** object that is less specific than the **route** object submission. The order that these checks are tried is fixed and as stated here. The sequence is only followed until an appropriate object is found. As soon as one of the appropriate objects is found, the authentication from that object will be used. If it fails that authentication it will not continue the sequence.

Finally, the creation must be authorised by the **mntner** of the **route** object itself referenced by the "mnt-by:" attribute of the **route** object submission.

For the checks against the **aut-num**, **route** and **inetnum** objects, authorisation shall be tested using the **mntner** object referenced in the "mnt-routes:" attribute(s) first. If there are no "mnt-routes:" attributes, the "mnt-lower:" attributes are checked. If there are no "mnt-lower:" attributes, the "mnt-by:" attributes are used for the authorisation check. Again, the order that these checks are performed is fixed and as stated here. The sequence is only followed until an appropriate attribute is found. As soon as one of the appropriate attributes is found, the authentication from that attribute will be used. If it fails that authentication it will not continue the sequence.

The same sequence applies to **route6** and **inet6num** objects.

2.5.6 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 in the form "AS1:AS-BAR".

When a hierarchical name is not used, authorisation for objects such as **as-set** and **route-set** corresponds to the rules for individual objects described in Section 2.5.2, 'Protection of Individual Objects'.

If hierarchical names are used, then the creation of an object must be authorised by the object whose class primary key is named by everything to the left of the rightmost colon in the class primary key name of the object being created. Continuing the hierarchy from above to create the object "AS1:AS-BAR:AS2". This would need to be authorised by "AS1:AS-BAR". This object is considered to be the 'parent' of the object being created.

Authorisation is determined by first using any **mntner** object referenced by any "mnt-lower:" attribute in the parent object. If none exist, then any **mntner** object referenced by any "mnt-by:" attribute in the parent object will be used.

The parent authorisation is required in addition to the authorisation of the individual object itself. Parent authorisation is only required when an object is created.

The object descriptions for each object type in Section 1.2, 'Object Types' shows which objects are hierarchical.

2.5.7 Protection of Domain Object Space

Protection of the forward **domain** object space is done with the "mnt-lower:" attribute. When used in the forward **domain** object, this attribute references the **mntner** object 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. If there is no corresponding second-level **domain** object, then a top-level **domain** object protects the creation of the third-level **domain** objects, etc. Only the Database Administrator can create top-level **domain** objects (ccTLD or gTLD).

If no "mnt-lower:" attributes exist, then any one of the "mnt-by:" attributes of the parent object will be used. The parent authorisation is required in addition to the authorisation of the individual object itself. Parent authorisation is only required when an object is created. The maintainer attributes are both optional in the domain object. If no maintainer attributes exist in the parent object, then no parent authorisation is required. If the domain object being created has no "mnt-by:" attribute then its own individual authorisation is not required.

Protection of the reverse domain object space for "in-addr.arpa" domains is done with separate methods for creation, deletion and modification.

For creation, first look for an exact match **inetnum** object. If this is not found look for a less specific **inetnum** object. If either of these is found, use this as the first stage of authorisation. Authorisation can be approved by any **mntner** object referenced in any of the "mnt-domains:" attributes of the selected **inetnum** object. If no "mnt-domains:" attributes exist in the selected **inetnum** object, use any **mntner** object referenced from any "mnt-lower:" attribute in the **inetnum** object. If none of these exist, use "mnt-by:" attributes.

If there is no **inetnum** object found, or if the authorisation from the **inetnum** object fails, then look for a parent **domain** object that is directly above the **domain** object to be created. If none is found then the authorisation fails. If it is found then follow the same process to check the authorisation as above. Use the "mnt-lower:" attributes if present, otherwise check the "mnt-by:" attributes.

This is different to all the other authentication processes. Normally when an appropriate object is found to check authorisation against the search sequence ends. In this case, the sequence continues if the authorisation is checked and fails.

For modification and deletion, any **mntner** referenced in a "mnt-by" attribute of the object can authorise the update. Where a deletion fails, **mntners** referenced in a "mnt-domains", "mnt-lower" or "mnt-by" attribute of the corresponding **inetnum** object can authorise the update. They will be checked in this order. The first attribute type found will be taken as the only one to use.

2.5.8 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 modifying an object-member. This "member-of:" attribute can be used in **route(6)**, **aut-num** and **inet-rtr** objects. 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**. The **set** owner must validate the membership claim of an object with a "member-of:" attribute. It does that by matching a "mnt-by:" attribute of the object with one of the maintainers in a "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(6)**, **aut-num** or **inet-rtr**), the operation fails. If a **set** object has no "mbrs-by-ref:" attributes, the **set** is defined explicitly by the "members:" attributes in the **set** object. In this case no other object can validate a claim to be a member of this **set**.

2.5.9 Referencing an irt Object

The **irt** object can be referenced in **inetnum** and **inet6num** objects. This reference is made by adding an optional "mnt-irt:" attribute to the **inet(6)num** object with the name of the **irt** object. Adding a reference to an **irt** object requires authorisation from the **irt** object. Authorisation can be approved by any of the **mntner** objects referenced in any of the "auth:" attributes of the **irt** object.

Authorisation from the **irt** object is only required when a "mnt-irt:" attribute is added to a referencing object, either on creation or by modification. Deletion of the referencing object or any modification that does not add an "mnt-irt:" attribute does not require authorisation from the **irt** object. Removing the "mnt-irt:" attribute does not require authorisation from the **irt** object either. The **irt** authorisation is required in addition to the authorisation of the individual object itself.

2.5.10 Referencing an Organisation Object

The **organisation** object can be referenced in any object. This reference is made by adding an optional "org:" attribute to the object being updated, along with the name of the **organisation** object. Adding a reference to an **organisation** object requires authorisation from the **organisation** object itself. Authorisation can be approved by any of the **mntner** objects referenced in any of the "mnt-ref:" attributes of the **organisation** object.

Authorisation from the **organisation** object is only required when an "org:" attribute is added to a referencing object, either on creation or by modification. This is in addition to the authorisation of the referencing object itself. Deletion of the referencing object or any modification that does not add an "org:" attribute (including removing an existing "org:" attribute) does not require authorisation from the **organisation** object.

3.0 Mirroring of the RIPE Database

Near Real Time Mirroring (NRTM) is a mechanism that allows servers other than the primary server for a given database, to maintain an up-to-date copy of all the data in the database. It will receive any modifications as they are processed by the primary server.

Permission must be obtained from the Database Administration to be allowed to mirror the RIPE Database. For Data Protection reasons, there are very strict criteria which must be satisfied before this permission is granted. Even when your reasons are acceptable, other ways of satisfying your requirements will be explored before mirroring is considered. The mirroring process described below is not accessible until this permission has been granted. For further information contact the Customer Services by e-mailing ripe-dbm@ripe.net

When permission to mirror has been granted, periodically (defined by configuration, usually around every 15 minutes) the remote server will connect 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. After the defined period a new connection is made and a new request sent.

In RIPE Database series 3.x, the NRTM server listens on a port that is different from the whois port (43).

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

The old (series 2.x) database software supported a NRTM protocol Version 1. With this version of the protocol, 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 in series 3.x of the database software (though this option is still supported). A modification is now treated 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 line-break characters (\n\n) and then the corresponding object. This will either be the object as it was before deletion or the object as it should appear after being created or modified. If an object already existing in the database follows the "ADD" string, the database software will treat this as a modification.

You can request NRTM by using the "-g" flag on a query. The arguments to this query flag are:

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

where

- <source> is the string that identifies the database that is being mirrored (for example RIPE).
- *<NRTM_protocol_version_#>* is the version of the mirroring protocol that the <source> supports (this should be 2 for the source RIPE).
- <first> is the lowest serial number requested.
- <last> is the most recent serial number requested. If the keyword "LAST" is used, this tells the main server to send all updates up to the most recent one available at the time of the query. The main server will never send the last object it processed to the mirror server, or display that it is available for mirroring. This protects secondary mirror servers in case the last update causes database corruption and a server crash. If the main server crashes, it automatically starts in a mode that does not allow for further mirroring updates. The updates that may have caused the crash are not sent to the mirror server.

A client may request a persistent connection by including the "-k" query flag with a mirroring request ("-g" query flag). In this case, the last argument is ignored and the server supplies the new objects as soon as they are processed. The client is responsible for closing the connection. A persistent connection for mirroring does not time out, but the connection may break if there is a server error.

You can use the "-q sources" flag with the mirror server to retrieve information regarding available mirroring possibilities. Please see Section 2.12, 'Other Server Features' in the RIPE Database Query Reference Manual for more details.

The main server never returns the latest serial when serving a mirroring request. The latest serial is not displayed with the "-q sources" query either.

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
```

4.0 Using the RIPE Database Efficiently

Most of this manual provides technical details of how the RIPE Database works. This section provides advice on how to use the RIPE Database efficiently, and hopefully make your life easier. Future releases of this document may include additional advice from users.

4.1 Using the Role Object

The **person** and **role** objects are often said to be interchangeable:

- They share the same name space in the database
- The nic-hdls are only unique across the two object types combined
- A **person** object can be used everywhere that a **role** object can be used.

But these two objects have very different functions. A **person** object holds personal details about an individual. A **role** object should describe a business function or operational unit and will reference the individual people responsible for this activity.

Although it is possible to do everything you want with the RIPE Database without ever using a **role** object, large scale changes can be costly. By careful planning, you can make future changes very easy to handle. The principle is the same if you have 10 objects or 10,000 objects in the database. However, problems most commonly occur when dealing with a very large number of objects.

Many organisations create a large number of objects which directly reference a specific **person** object, and find themselves in trouble if this person leaves the company. The organisation may be responsible for many objects of different types, possibly with several different **mntner** objects protecting them, and finding them and getting all the authorisations right to change the references can easily become a problem.

A few basic principles will help to avoid this situation. Only use a **person** object as a holder of personal information, and only reference a **person** object in **role** objects. Reference the **role** objects in all the other objects where contact data is required. If the person responsible for a role changes, then it is simply necessary to modify a few **role** objects to reference a different **person** object. All references to the **role** objects remain valid.

Even if you have only a handful of objects in the database, it is good practice to do this. Your business may grow, and human nature means you will not go back and change things until you have to do so. This is how these objects were intended to be used, but as this practice was never enforced, half the database still makes direct references to **person** objects.

4.2 Using the Organisation Object

The **organisation** object was introduced long after the database was designed. When this object was introduced it was intended to be used in a certain way, but again, this practice was never enforced, and many database users have not adopted it. It is worth knowing how the **organistion** object can make life easier in some situations.

Consider all users as entities, whether they are multinational companies, universities or individuals. To use the RIPE Database each of these entities needs a set of data objects that represent their business model. They will need:

- People who can be contacted
- Who have defined roles in the business
- Which are responsible for Internet resources
- That need authentication tokens to protect them
- Which may need public keys

This set of objects represents the organisation of the entity. When the entity is an individual who has been assigned some PI space, they may need several objects in the database. Multinational companies may have many thousands of objects.

The **organisation** object was introduced as a way of keeping track of these sets of objects. The idea is to put the organisational identity of the entity at the centre by defining its **organisation** object. The organisation's business model can then be mapped out by creating the objects from the list above as appropriate. Each of these objects can be directly 'tagged' with a reference to the **organisation** object using the "org:" attribute. Or for a simplified model, tag the **mntner** objects using the "org:" attribute in the **mntner** object. If all objects are maintained, then there is an indirect reference back to the **organisation** object through the **mntner** objects.

Some multinational companies may have a devolved business model with different parts of the organisation responsible for different parts of their network. In this situation additional **organisation** objects can be created. These objects can reference the main

organisation object through their own "org:" attribute. This allows users to keep track of the entire company's data or the parts delegated to different sections of the company.

If this is done, it is easy to 'see' the map of all the data for an entity. Any bulk changes to data are very much simplified. Tools can be written and deployed more easily. New ideas can be rolled out quickly across an entire data set. Concepts like abuse handling could be re-visited. This could be applied with a default, centralised abuse handler in the **organisation** object, as well as more localised, optional ones in the **mntner** objects or the individual objects themselves.

4.3 Abuse Handling

The **irt** (Internet Response Team) object was originally introduced to identify teams for handling serious network problems like DOS attacks. It has since been adapted to not only fulfil that role, but to also try to handle abuse complaints at all levels.

Several objects can optionally include an "abuse-mailbox:" attribute. This includes the **irt** object. The **irt** object can be referenced from **inetnum** and **inet6num** objects. When an **irt** object, including an "abuse-mailbox:" attribute, is referenced from an **inet(6)num** object, this defines the abuse handler for this address space. It also defines the abuse handler for some of the more specific address space to that specified by the **inet(6)num** object referencing the **irt** object.

There is a query flag ("-c") which will return the **irt** object, if one exists, for any specified **inet(6)num** object. There is also another query flag ("-b") that will find the **irt** object, extract the "abuse-mailbox:" attribute and return brief details including the contact address from the **irt** object and others. For details of how these queries work see Section 2.4, 'Abuse Contacts' in the "RIPE Database Query Reference Manual" [18].

Appendices

A1. Object Attributes

This appendix has NOT been updated in this draft document. There are some corrections and updates to be included in the next draft.

Shown below are the syntax definitions of the object attributes that the RIPE Database supports.

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>)*<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]
`<domain-name>`	Domain name as specified in RFC 1034 [7] without trailing dot ("."). The total length should not exceed 255 characters (octets) `<dlabel>(\.<dlabel>)*`
`<e-mail>`	e-mail address specification as defined in RFC 2822 [8],
`<filter>`	Please see RFC 2622 [1]
`<freeform>`	A sequence of ASCII characters
`<inet-rtr-name>`	Specifies the name of an inet-rtr object. It is a `<domain-name>`.
`<ipv4-address>`	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}`
`<ipv6-address>`	`<quad>(:<quad>){7,7}`
`<ipv6-address-prefix>`	`<ipv6-address>/integer` (between 0 and 128)
`<ipv6-filter>`	Please see RPSLng [14]
`<irt-name>`	Specifies the name of an **irt** object. It is an `<object-name>` starting with "IRT-" prefix reserved for this object class.
`<mntner-name>`	`<object-name>`
`<nic-handle>`	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}([1-9]<digit>{0,5})?(-<alpha>`
	`([a-zA-Z0-9_-]{0,7}<alnum>))?)|`
	`(AUTO-<digit>+(<alpha>{2,4})?)`

`<object-name>`	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 **rtr-set** names. Names starting with "fltr-" are reserved for **filter-set** names. Names starting with "prng-" are reserved for **peering-set** names. Names starting with "irt-" are reserved for **irt** object names. This is the RIPE Database extension.

`<org-id>`	The 'ORG-' string followed by 2 to 4 characters, followed by up to 5 digits followed by a source specification. The first digit must not be "0". Source specification starts with "-" followed by source name up to 9-character length.
`<organisation-name>`	Is a list of a most 12 words, each at most 64 characters in length. Words can contain alphanumeric characters, asterisk, plus and minus signs, forward slash and backslash, dash, quotes, at sign, commas, dots, underscores, ampersands, exclamation marks, colons, semicolons, brackets and square brackets.
`<peering>`	Please see RFC 2622 [1]
`<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] and RPSLng [14]
<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: <freeform>
Full postal address of a contact.

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

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

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

Alias: <domain-name>
Specifies a canonical DNS name for the router.

As-block: <as-number> - <as-number>
Specifies the range of ASNs that the **as-block** object represents. Please see [2] 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**
MD5		This scheme is based on the MD5 hash algorithm and provides stronger authentication than CRYP-PW. The authentication information stored in the database is a pass phrase encrypted using md5-crypt algorithm, which is a concatenation of the "1" string, the salt, and the 128-bit hash output. Because it uses 8-character salt and an almost unlimited pass phrase, this scheme is more stable against dictionary attacks. However, since the encrypted form is exposed it cannot be considered as a strong form of authentication.
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.
X5509-<id>		Strongest scheme of authentication. <id> is the auto-generated ID of the X509 certificate 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 for a PGP key or an X509 certificate. 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 or the certificate from your browser and prepending each line of the key with the string "certif:". In the second case, line continuation should be used to represent the key. All the lines of the exported key must be included. For PGP, this includes the begin and end markers and the empty line which separates the header from the key body. For X509 certificates, this includes the BEGIN CERTIFICATE and END CERTIFICATE lines.

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> ...]]

or: [ATOMIC] [[<ipv6-filter>] [protocol <protocol> <ipv6-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> or <ipv6-filter> is a policy expression.
Please refer to RFC 2622 [1] and RPSLng [14] for more information.

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

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

descr: <freeform>
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.

e-mail: <e-mail>
Specifies an e-mail address of a person, role, organisation or irt team.

encryption: PGPKEY-<id>
References a **key-cert** object representing a CSIRT public key used to encrypt correspondence sent to the CSIRT. <Id> is the D of the PGP public key in eight digit hexadecimal format without "0x" prefix.

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 [1] for more information.

export-comps: <filter> or <ipv6-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] and RPSLng [14] 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. Please refer to RFC 2726 [9] for detailed description of this attribute.

form: **FORM** <string>
Specifies the identifier of a registered poem object.

holes: list of <address-prefix> or <ipv6-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] and RPSLng [14] 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 addresses. 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. In route objects, the router expression can contain only IPv4 expressions, and in route6 object it can only cotain IPv6 expressions. Please refer to RFC 2622 [1] and RPSLng [14] for more information.

interface: <ipv4-address> or <ipv6-address> masklen <masklen>
<integer> [action <action>]

[tunnel <remote-endpoint-address>,<encapsulation>]

Specifies a multiprotocol interface address within an Internet router. Please refer to RPSLng [14] for more information.

irt: <irt-name>
A unique identifier of an **irt** object. The name should start with the prefix "IRT-", reserved for this type of object.

irt-nfy: <e-mail>
Specifies the e-mail address to be notified when a reference to the **irt** object is added or removed.

key-cert: PGPKEY-<id>
Defines the public key stored in the database. <Id> is the ID of the PGP 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.

method: <generated>
Defines the type of the public key. Currently the only methods supported are "PGP" and "X509". Please refer to RFC 2726 [9] for detailed description of this attribute.

member-of: list of <set-name>
This attribute can be used in the **route**, **route6, 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-domains: list of <mntner-name>

Specifies the identifier of a registered **mntner** object used for reverse **domain** authorization. Protects **domain** objects. The authentication method of this **maintainer** object will be used for any encompassing reverse **domain** object.

mnt-irt: list of <irt-name>
May appear in an **inetnum** or **inet6num** object. It points to an existing **irt** object representing CSIRT that handles security incidents for the address space specified by the **inetnum** or **inet6num** object.

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, route6** 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-ref: list of <mntner-name>

Specifies the **maintainer** objects that are entitled to add references to the **organisation** object from other objects.

mnt-routes: <mnt-name> [{ list of <address-prefix-range> } | ANY]
May appear in an **aut-num**, **inetnum**, **inet6num**, **route** or **route6** object. This attribute references a **maintainer** object which is used in determining authorisation for the creation of **route** and route6 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. The address prefix range can contain only IPv4 prefix ranges in **inetnum** and **route** objects, only IPv6 prefix ranges in **inet6num** and **route6** objects, and it can contain both IPv4 and IPv6 prefix ranges in **aut-num** objects. Please refer to RFC-2622 [1] and RPSLng [14] for more information.

mp-default: `to <peering> [action <action>] [networks <filter>]`

Specifies default multiprotocol routing policies. Please refer to RPSLng [4] for more information.

mp-export:

```
[protocol <protocol-1>] [into <protocol-1>]
afi <afi-list>
to <peering-1> [action <action-1>]
    .
    .
    .
to <peering-N> [action <action-N>]
announce <filter>
```

Specifies a multiprotocol export policy expression. Please refer to RPSLng [14] for more information.

mp-filter:

Defines the set's multiprotocol policy filter. Please refer to RPSLng [14] for more information.

mp-import: `[protocol <protocol-1>] [into <protocol-1>]`

```
afi <afi-list>
from <peering-1> [action <action-1>]
    .
    .
    .
from <peering-N> [action <action-N>]
accept (<filter>|<filter> except <importexpression>|
        <filter> refine <importexpression>)
```

Specifies multiprotocol import policy expression. Please refer to RPSLng [14] for more information.

mp-members: `afi <afi-list> list of <address-prefix-range> or`

`<route-set-name> or`

`<route-set-name><range-operator>`

Lists the multiprotocol members of the set. Refer to RPSLng [14] for more information.

mp-peer: <protocol> afi <afi> <ipv4- or ipv6- address> <options>

| <protocol> <inet-rtr-name> <options>

| <protocol> <rtr-set-name> <options>

| <protocol> <peering-set-name> <options>

Specifies the details of any (interior or exterior) multiprotocol router peerings. Please refer to RPSLng [14] for more information.

mp-peering: afi <afi> <peering>

Defines a multiprotocol peering that can be used for importing or exporting routes. Please see RPSLng [14] 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.

org: <org-id>

This attribute may appear in any object type. It points to an existing **organisation** object representing the entity that holds the resource, (in the cases where the whois database object represents an Internet resource). In other objects, it can be used to specify the business relations. The value of this attribute is the ID of the **organisation** object. It is mandatory in the **inetnum** and **inet6num** objects with `ALLOCATED-BY-RIR`, `ALLOCATED PA`, `ALLOCATED PI` and `ALLOCATED UNSPECIFIED` values. It is optional in all other objects.

The `"org:"` attribute is single-valued in the **inetnum**, **inet6num** and **aut-num** objects, and it is multi-valued in all other objects. The `"org:"` attribute is used to specify the holder of a resource in **inetnum**, **inet6num** and **aut-num** objects, thus it must be single-valued in them. In other objects, it specifies business relations (such as a **person** object, where it can be used to specify whom the person works for) it can be multiple (in the **person** object example, a person might work for several companies).

org-name: <organisation-name>

Specifies the name of the organisation that this **organisation** object represents in the whois database. This is an ASCII-only text attribute. This restriction is because the attribute is a look-up key and the whois protocol does not allow specifying character sets in queries. The user can put the name of the organisation in non-ASCII character sets in the `"descr:"` attribute if required.

org-type: IANA, RIR, NIR, LIR or NON-REGISTRY

Specifies the type of the organisation. The possible values are `IANA` for Internet Assigned Numbers Authority, `RIR` for Regional Internet Registries, `NIR` for National Internet Registries, `LIR` for Local Internet Registries, and `NON-REGISTRY` for all other organisations. Note that in the RIPE NCC service region there are no National Internet Registries, therefore in the RIPE Database there will not be any **organisation** object with this value in the `"org-type:"` attribute.

organisation: <org-id>

Specifies the ID of an organisation object. When creating an object, one has to specify an "AUTO" ID by setting the value of the attribute to

"AUTO-1" or "AUTO-1<letterCombination>, so the database will assign the ID automatically.

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. Please refer to RFC 2726 [9] for detailed description of this attribute.

peer:		<protocol><ipv4-address><options>
		<protocol><inet-rtr-name><options>
		<protocol><rtr-set-name><options>
		<protocol><peering-set-name><options>

May appear in an **inet-rtr** object. Specifies a protocol peering with another router. Please refer to RFC 2622 [1] for more information.

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.

poem: POEM <string>
Specifies the title of a poem.

poetic-form: FORM <string>
Specifies the poem type.

ref-nfy: <e-mail>
Specifies the e-mail address to be notified when a reference to the **organisation** object is added or removed. An e-mail address as defined in RFC 2822 [8].

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. For more information, please see Section 2.7, 'Referral Mechanism for Domains' of the "RIPE Database Query Reference Manual" [18].

<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. It is not currently used by the database software.

remarks: <freeform>
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. It is for documentation only and has no effect on reverse delegation.

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

route6: <ipv6-address>/<prefix-length>
Specifies an IPv6 prefix. The <prefix length> is an integer in the range from 0 to 128. This is the prefix of the interAS route. Together with the "origin:" attribute, constitutes a primary key of the **route6** 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.

signature: PGPKEY-<id>
References a **key-cert** object representing a CSIRT public key used by the team to sign their correspondence. <Id> is the ID of the PGP public key in 8-digit hexadecimal format without "0x" prefix.

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> must have one of these values:

- ALLOCATED PA
- ALLOCATED PI
- ALLOCATED UNSPECIFIED
- ASSIGNED PA
- ASSIGNED PI
- LIR-PARTITIONED PA
- LIR-PARTITIONED PI
- NOT-SET

Please refer to the RIPE document "IPv4 Address Allocation and Assignment Policies in the RIPE NCC Service Region" for further information. Please refer to [10] regarding usage of the LIR-PARTITIONED status value.
For **inet6num**, <status> can have one of the following values:

- ALLOCATED-BY-RIR - For allocations made by an RIR to an LIR.
- ALLOCATED-BY-LIR - For allocations made by an LIR or an LIR's downstream customer to another downstream organisation.
- ASSIGNED - For assignments made to End User sites.

Please refer to [13] regarding usage of the status value for **inet6num** objects.

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

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. Copyright Information

A2.1 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 at the time of publishing this manual 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

A2.2 RIPE NCC Copyright

Acknowledgements

The authors wish to acknowledge the effort done by the original developers of the version 3.0 of the RIPE Database at the RIPE NCC: Daniele Arena, Marek Bukowy, Engin Gunduz, Roman Karpiuk, Shane Kerr, A.M.R. Magee, Chris Ottrey and Filippo Portera. Those who have continued its development include Can Bican, Katie Petruska, Denis Walker, Tiago Anteo and Agoston Horvath.

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

[5] RIPE Database - Getting Started

[6] IRRToolset, see http://www.isc.org/sw/IRRToolSet/

[7] P. Mockapetris, "Domain names - Concepts and Facilities", RFC 1034, November 1987

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

[9] J. Zsako, "PGP Authentication for RIPE Database Updates", RFC 2726, December 1999

[10]) N. Nimpuno and A.Robachevsky, "New Value of the "status:" Attribute for Inetnum Objects (LIR-PARTITIONED)", ripe-239, June 2002

[11] A. Cormack, D. Stikvoort, W. Woeber and A. Robachevsky, "IRT Object in the RIPE Database" ripe-254, July 2002

[12] K. Harrenstien, M.K. Stahl and E.J. Feinler, "NICNAME/WHOIS", RFC 954, October 1985

[13] J.S.L. Damas and L. Vegoda, "New Values of the "status:" Attribute for **inet6num** Objects", ripe-243, August 2002

[14] L. Blunk, J. Damas, F. Parent and A. Robachevsky, Routing Policy Specification Language next generation (RPSLng), work in progress, August 2004

[15] C. Bican, RIPE-43 presentation on Webupdates, December 2002, http://www.ripe.net/ripe/meetings/ripe-43/presentations/ripe43-database-syncupdates/index.html

[16] RIPE NCC service region, http://www.ripe.net/membership/maps/index.html

[17] The IANA ccTLD Database contains a full list of the ccTLD administrators, http://www.iana.org/cctld/cctld-whois.htm

[18] RIPE Database Query Reference Manual

[19] RIPE document store, http://www.ripe.net/ripe/docs/

[20] Protocol details for syncupdates, see http://www.ripe.net/db/syncupdates/syncupdates-manual.html

[21] Sample clients for syncupdates, http://www.ripe.net/db/syncupdates/

[22] Webupdates cgi, https://www.ripe.net/fcgi-bin/webupdates.pl

[23] J. Callas, L. Donnerhacke, H. Finney and R. Thayer, "OpenPGP Message Format", RFC 2440, November 1998

[24] R. Fielding, J. Gettys, J. Mogul, H. Frystyk and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2068, January 1997