

---

# **Carnivora Documentation**

***Release 0.13.2***

**Sophie Herold**

**Dec 22, 2019**



<b>1</b>	<b>Carnivora</b>	<b>1</b>
1.1	Client Software . . . . .	1
1.2	Installing Carnivora . . . . .	1
<b>2</b>	<b>API Conventions</b>	<b>3</b>
2.1	Function Naming . . . . .	3
2.2	Backend Notification Naming . . . . .	3
<b>3</b>	<b>Development</b>	<b>5</b>
3.1	Todo List . . . . .	5
<b>4</b>	<b>dns</b>	<b>7</b>
4.1	Tables . . . . .	8
4.2	Functions . . . . .	11
4.3	Domains . . . . .	24
<b>5</b>	<b>domain_reseller</b>	<b>27</b>
5.1	Tables . . . . .	28
5.2	Functions . . . . .	30
<b>6</b>	<b>email</b>	<b>41</b>
6.1	Tables . . . . .	42
6.2	Functions . . . . .	49
6.3	Domains . . . . .	69
<b>7</b>	<b>jabber</b>	<b>71</b>
7.1	Tables . . . . .	71
7.2	Functions . . . . .	73
<b>8</b>	<b>server_access</b>	<b>77</b>
8.1	Tables . . . . .	78
8.2	Functions . . . . .	79
8.3	Domains . . . . .	83
<b>9</b>	<b>web</b>	<b>85</b>
9.1	Tables . . . . .	86
9.2	Functions . . . . .	89

<b>10 backend</b>	<b>97</b>
10.1 Tables . . . . .	98
10.2 Functions . . . . .	98
10.3 Domains . . . . .	103
10.4 Roles . . . . .	103
<b>11 commons</b>	<b>105</b>
11.1 Functions . . . . .	106
11.2 Domains . . . . .	108
11.3 Sequences . . . . .	109
<b>12 system</b>	<b>111</b>
12.1 Tables . . . . .	112
12.2 Functions . . . . .	118
<b>13 user</b>	<b>125</b>
13.1 Tables . . . . .	126
13.2 Functions . . . . .	127
13.3 Domains . . . . .	130
13.4 Roles . . . . .	131
<b>14 PostgreSQL</b>	<b>133</b>
14.1 Types . . . . .	133
<b>15 YamSql</b>	<b>135</b>
15.1 Types . . . . .	135

A powerfull backend for web-service management. Written in YamSql.

The documentation can be found online at [carnivora.readthedocs.io](http://carnivora.readthedocs.io) or as sphinx source in `docs/`.

## 1.1 Client Software

**Canini** Full privileged superadmin CLI. Supports adding additional modules via config. Written in Python 3.

**Edentata** An unprivileged webinterface without superadmin capabilities. Targeting end-users and focused on usability. Supports adding additional modules via config. Written in PHP 7.

**Genconfig** A generic config producer which can use carnivora as backend. Written in Python 3.

**PgListend** Daemon that executes tasks on PostgreSQL push signals. Designed to call genconfig on database updates. Written in Python 3.

**LibInternetX** PHP library for connecting to the InterNetX domain reseller XML API. Includes CLI coupling to Carnivora.

## 1.2 Installing Carnivora

### 1.2.1 Perequisites

The setup is performed via [HamSql](#). It should be callable as `hamsql` in your shell.

Install PostgreSQL on Debian

```
apt install postgresql postgresql-contrib postgresql-plpython3
```

## 1.2.2 Configuration

You can configure accounts that can connect to the database via `/etc/carnivora/_postgresql_user/module.yaml`. The accounts generated via this config have the names `carnivora_edentata` and `carnivora_machine_example`.

```
name: _postgresql_user
description: PostgreSQL users and their privileges

roles:
-
  name: edentata
  login: true
  description: Account for edentata web frontend
  member_in:
    - userlogin
-
  name: machine_example
  description: Account for machine example
  login: true
  member_in:
    - backend
```

## 1.2.3 Running the Setup

Simplest way to execute the setup on a system with a default PostgreSQL configuration is to run

```
su postgres -c "hamsql install -s examples/setup.yaml -c postgres://postgres@/carnivora
↪"
```

Supplying the database name (here *carnivora*) via the `-c` option is mandatory. The database will be create if it is not present. Additional or deviating connection options can be provided.

### 2.1 Function Naming

**Prefix `_`** Internal functions which do not belong to API.

#### 2.1.1 Frontend API

**Prefix `del_`** Delete object from database. Returns `void`.

**Prefix `ins_`** Insert object to database. Returns `void`.

**Prefix `sel_`** Gives all object for which the user has ownership. Returns a recordset that can be used as `<table>` in a `SELECT ... FROM <table> statement`.

**Prefix `upd_`** Updates objects in database. Returns `void`.

#### 2.1.2 Backend API

**Prefix `srv_`** Gives all object designated to the connected machine. Returns a recordset.

**Prefix `fwd_`** Forwards informations from a machine to carnivora. Similar to `upd_` but for backend.

### 2.2 Backend Notification Naming

Carnivora sends push notifications (usign the PostgreSQL `NOTIFY` command) if objects are changed. Machines can connect to channels that only give notifications relevant to them.

**Channel** `carnivora/ machine name`

**Payload** `service entity name / service / subservice`

Example `mail.example.org/mail/mailbox`.

The emitted signals are documented in the schema description. The service entity name is silently omitted in those documentations.



## CHAPTER 3

---

### Development

---

#### 3.1 Todo List

---

**Todo:** Document *managed\_custom*. Unclear if this is even properly supported or checked.

---

(The [original entry](#) is located in /home/docs/checkouts/readthedocs.org/user\_builds/carnivora/checkouts/latest/docs/schemas/dns.rst, line 14.)

---

**Todo:** checks might be off

---

(The [original entry](#) is located in /home/docs/checkouts/readthedocs.org/user\_builds/carnivora/checkouts/latest/docs/schemas/dns.rst, line 1384.)

---

**Todo:** validity checks

---

(The [original entry](#) is located in /home/docs/checkouts/readthedocs.org/user\_builds/carnivora/checkouts/latest/docs/schemas/email.rst, line 2287.)

---

**Todo:** check owner and contingent

---

(The [original entry](#) is located in /home/docs/checkouts/readthedocs.org/user\_builds/carnivora/checkouts/latest/docs/schemas/web.rst, line 511.)

---

**Todo:** proper checking of format

---

(The [original entry](#) is located in /home/docs/checkouts/readthedocs.org/user\_builds/carnivora/checkouts/latest/docs/schemas\_system/con line 277.)



### DNS and Registered Domains

The entity name for **domain\_registered** services are considered the *nameservers* used for this domain. In case of the *unmanaged* subservice, a fake name or the responsible nameserver that is not managed by the system can be given.

To allow service activation, the service needs a `dns_activatable` subservice entity.

---

**Todo:** Document *managed\_custom*. Unclear if this is even properly supported or checked.

---

#### Schema Contents

- *Tables*

- `dns.custom`
- `dns.registered`
- `dns.service`

- *Functions*

- `dns._domain_order`
- `dns._is_subdomain_of`
- `dns._rdata_txtdata_valid`
- `dns.del_custom`
- `dns.del_registered`
- `dns.del_service`
- `dns.fwd_registered_status`
- `dns.ins_custom`

- `dns.ins_registered`
- `dns.ins_service`
- `dns.sel_activatable_service`
- `dns.sel_custom`
- `dns.sel_nameserver`
- `dns.sel_registered`
- `dns.sel_service`
- `dns.sel_usable_domain`
- `dns.srv_record`
- `dns.upd_custom`
- *Domains*
  - `dns.t_domain`
  - `dns.t_domain_rdata`
  - `dns.t_hostname`
  - `dns.t_rdata`
  - `dns.t_ttl`
  - `dns.t_type`

## 4.1 Tables

### 4.1.1 `dns.custom`

Direct name server entries.

#### Primary key

- `id`

#### Columns

- **type** `dns.t_type` Type (A, AAAA, CNAME, MX, SRV, TXT, ...)
- **rdata** `dns.t_rdata` fancy rdata storage
- **ttl** `NULL` | `dns.t_ttl` Time to live, NULL indicates default value
- **backend\_status** `NULL` | `backend.t_status` Status of database entry in backend. NULL: nothing pending, 'ins': entry not present on backend client, 'upd': update pending on backend client, 'del': deletion pending on backend client.

#### Default

`'ins'`

- **registered** `dns.t_hostname` Registered domain of which domain is a sub domain

References `dns.registered.domain`

On Delete: CASCADE

- **domain** *dns.t\_domain* domain of entry
- **id** *uuid* uuid serial number to identify database elements uniquely

#### Default

```
commons._uuid()
```

### 4.1.2 dns.registered

Domains registered under a public suffix.

#### Primary key

- domain

#### Foreign keys

- Reference service entity

##### Local Columns

- service\_entity\_name
- service

##### Referenced Columns

- *system.service\_entity.service\_entity\_name*
- *system.service\_entity.service*

- Reference subservice entity

##### Local Columns

- service\_entity\_name
- service
- subservice

##### Referenced Columns

- *system.subservice\_entity.service\_entity\_name*
- *system.subservice\_entity.service*
- *system.subservice\_entity.subservice*

#### Columns

- **owner** *user.t\_user* Owner

References *user.user.owner*

On Update: CASCADE

- **backend\_status** *NULL* | *backend.t\_status* Status of database entry in backend. NULL: nothing pending, 'ins': entry not present on backend client, 'upd': update pending on backend client, 'del': deletion pending on backend client.

#### Default

'ins'

- **service\_entity\_name** *dns.t\_hostname* Service entity name
- **service** *commons.t\_key* Service (e.g. email, jabber)
- **subservice** *commons.t\_key* Subservice (e.g. account, alias)
- **domain** *dns.t\_hostname* Domain
- **public\_suffix** *varchar* Public Suffix

### 4.1.3 dns.service

Name server entries based on system.service (i.e. system.service\_dns)

#### Primary key

- domain
- service

#### Foreign keys

- Reference service entity

#### Local Columns

- service\_entity\_name
- service

#### Referenced Columns

- *system.service\_entity.service\_entity\_name*
- *system.service\_entity.service*

#### Columns

- **service\_entity\_name** *dns.t\_hostname* Service entity name
- **service** *commons.t\_key* Service (e.g. email, jabber)
- **backend\_status** *NULL | backend.t\_status* Status of database entry in backend. NULL: nothing pending, 'ins': entry not present on backend client, 'upd': update pending on backend client, 'del': deletion pending on backend client.

#### Default

'ins'

- **registered** *dns.t\_hostname* Registered domain of which domain is a sub domain  
References *dns.registered.domain*
- **domain** *dns.t\_hostname* domain for which the entries should be created

## 4.2 Functions

### 4.2.1 dns.\_\_domain\_order

ORDER

**Parameters**

- p\_domain *dns.t\_domain*

**Returns** varchar[]

**Execute privilege**

- *userlogin*
- *backend*

```
RETURN commons._reverse_array(regex_split_to_array(p_domain, E'\\.'));
```

### 4.2.2 dns.\_\_is\_subdomain\_of

Checks if *p\_subdomain* is a subdomain of *p\_domain*

**Parameters**

- p\_subdomain *dns.t\_domain*
- p\_domain *varchar*

**Returns** bool

```
RETURN p_domain = p_subdomain OR
'.' || p_domain = right(p_subdomain, char_length(p_domain) + 1);
```

### 4.2.3 dns.\_\_rdata\_txtdata\_valid

Rdata txt-data valid

**Parameters**

- p\_txtdata *varchar[]*

**Returns** bool

```
RETURN ((
  SELECT DISTINCT TRUE
  FROM UNNEST(p_txtdata) AS s
  WHERE octet_length(s) > 255
) IS NULL);
```

### 4.2.4 dns.del\_custom

Delete Custom

**Parameters**

- p\_id *uuid*

**Variables defined for body**

- v\_nameserver *dns.t\_hostname*
- v\_managed *commons.t\_key*
- v\_owner *user.t\_user*

**Returns** void**Execute privilege**

- *userlogin*

```
-- begin userlogin prelude
v_owner := (SELECT t.act_as FROM "user"._get_login() AS t);
-- end userlogin prelude

UPDATE dns.custom AS t
    SET backend_status = 'del'
FROM dns.registered AS s
WHERE
    s.domain = t.registered AND

    t.id = p_id AND
    s.owner = v_owner

RETURNING s.service_entity_name, s.subservice
INTO v_nameserver, v_managed;

PERFORM backend._conditional_notify_service_entity_name(
    FOUND, v_nameserver, 'dns', v_managed
);
```

## 4.2.5 dns.del\_registered

Delete registered domain

**Parameters**

- p\_domain *dns.t\_hostname*

**Variables defined for body**

- v\_nameserver *dns.t\_hostname*
- v\_managed *commons.t\_key*
- v\_owner *user.t\_user*

**Returns** void**Execute privilege**

- *userlogin*

```
-- begin userlogin prelude
v_owner := (SELECT t.act_as FROM "user"._get_login() AS t);
-- end userlogin prelude
```

(continues on next page)



(continued from previous page)

```

UPDATE dns.registered
SET backend_status = 'del'
WHERE domain = p_domain
  AND owner = v_owner
RETURNING service_entity_name, subservice
  INTO v_nameserver, v_managed;

PERFORM backend._conditional_notify_service_entity_name(
  FOUND, v_nameserver, 'domain_registered', v_managed
);

```

### 4.2.6 dns.del\_service

deletes all service entries of a specific domain

#### Parameters

- p\_domain *dns.t\_hostname*
- p\_service *commons.t\_key*

#### Variables defined for body

- v\_nameserver *dns.t\_hostname*
- v\_managed *commons.t\_key*
- v\_owner *user.t\_user*

Returns void

#### Execute privilege

- *userlogin*

```

-- begin userlogin prelude
v_owner := (SELECT t.act_as FROM "user"._get_login() AS t);
-- end userlogin prelude

BEGIN
  -- perform DELETE to trigger potential foreign key errors
  DELETE FROM dns.service AS t
  USING dns.registered AS s
  WHERE
    s.domain = t.registered AND

    t.domain = p_domain AND
    t.service = p_service AND
    s.owner = v_owner;

  -- if not failed yet, emulate rollback of DELETE
  RAISE transaction_rollback;
EXCEPTION
  WHEN transaction_rollback THEN
    UPDATE dns.service AS t
      SET backend_status = 'del'
    FROM dns.registered AS s

```

(continues on next page)

(continued from previous page)

```
WHERE
    s.domain = t.registered AND

    t.domain = p_domain AND
    t.service = p_service AND
    s.owner = v_owner
RETURNING s.service_entity_name, s.subservice
INTO v_nameserver, v_managed;

PERFORM backend._conditional_notify_service_entity_name(
    FOUND, v_nameserver, 'dns', v_managed
);

END;
```

### 4.2.7 dns.fwd\_registered\_status

Update status

#### Parameters

- p\_domain *dns.t\_hostname*
- p\_backend\_status *backend.t\_status*
- p\_include\_inactive *boolean*

Returns void

#### Execute privilege

- *backend*

```
PERFORM backend._get_login();

UPDATE dns.registered
SET
    backend_status = p_backend_status
WHERE domain = p_domain;
```

### 4.2.8 dns.ins\_custom

Ins Custom

#### Parameters

- p\_registered *dns.t\_hostname*
- p\_domain *dns.t\_domain*
- p\_type *dns.t\_type*
- p\_rdata *dns.t\_rdata*
- p\_ttl *integer*

Variables defined for body

- v\_nameserver *dns.t\_hostname*
- v\_managed *commons.t\_key*
- v\_owner *user.t\_user*

**Returns** void

**Execute privilege**

- *userlogin*

```
-- begin userlogin prelude
v_owner := (SELECT t.act_as FROM "user"._get_login() AS t);
-- end userlogin prelude

SELECT service_entity_name, subservice INTO v_nameserver, v_managed FROM dns.
↳registered
WHERE
    domain = p_registered AND
    owner = v_owner;

IF v_nameserver IS NULL THEN
    PERFORM commons._raise_inaccessible_or_missing();
END IF;

IF v_managed IS NULL THEN
    PERFORM commons._raise_inaccessible_or_missing();
END IF;

INSERT INTO dns.custom
(registered, domain, type, rdata, ttl)
VALUES
(p_registered, p_domain, p_type, p_rdata, p_ttl);

PERFORM backend._notify_service_entity_name(v_nameserver, 'dns', v_managed);
```

## 4.2.9 dns.ins\_registered

registers new domain

**Parameters**

- p\_domain *dns.t\_hostname*
- p\_subservice *commons.t\_key*
- p\_service\_entity\_name *dns.t\_hostname*
- p\_public\_suffix *varchar*

**Variables defined for body**

- v\_owner *user.t\_user*

**Returns** void

**Execute privilege**

- *userlogin*

```
-- begin userlogin prelude
v_owner := (SELECT t.act_as FROM "user"._get_login() AS t);
-- end userlogin prelude

INSERT INTO dns.registered
(domain, public_suffix, owner, service, subservice, service_entity_name)
VALUES
(p_domain, p_public_suffix, v_owner, 'domain_registered', p_subservice, p_service_
↪entity_name);

PERFORM backend._notify_service_entity_name(p_service_entity_name, 'domain_registered
↪', p_subservice);
```

### 4.2.10 dns.ins\_service

Creates service dns entry

#### Parameters

- p\_registered *dns.t\_hostname*
- p\_domain *dns.t\_hostname*
- p\_service\_entity\_name *dns.t\_hostname*
- p\_service *commons.t\_key*

#### Variables defined for body

- v\_nameserver *dns.t\_hostname*
- v\_managed *commons.t\_key*
- v\_owner *user.t\_user*

Returns void

#### Execute privilege

- *userlogin*

```
-- begin userlogin prelude
v_owner := (SELECT t.act_as FROM "user"._get_login() AS t);
-- end userlogin prelude

SELECT service_entity_name, subservice INTO v_nameserver, v_managed FROM dns.
↪registered
WHERE
    domain = p_registered AND
    owner = v_owner;

IF v_nameserver IS NULL THEN
    PERFORM commons._raise_inaccessible_or_missing();
END IF;

INSERT INTO dns.service (registered, domain, service_entity_name, service)
VALUES (p_registered, p_domain, p_service_entity_name, p_service);
```

(continues on next page)

(continued from previous page)

```
PERFORM backend._notify_service_entity_name(v_nameserver, 'dns', v_managed);
```

### 4.2.11 dns.sel\_activatable\_service

Activatable services

**Parameters** *None*

**Variables defined for body**

- v\_owner *user.t\_user*

**Returns** TABLE

**Returned columns**

- service *commons.t\_key*
- service\_entity\_name *dns.t\_hostname*

**Execute privilege**

- *userlogin*

```
-- begin userlogin prelude
v_owner := (SELECT t.act_as FROM "user"._get_login() AS t);
-- end userlogin prelude

RETURN QUERY
SELECT
    COALESCE(t.service, s.service) AS service,
    COALESCE(t.service_entity_name, s.service_entity_name) AS service_entity_name
FROM system._effective_contingent() AS t
FULL OUTER JOIN system._effective_contingent_domain() AS s
USING (service, subservice, service_entity_name, owner)
WHERE
    COALESCE(t.subservice, s.subservice) = 'dns_activatable' AND
    COALESCE(t.owner, s.owner) = v_owner

ORDER BY service, service_entity_name
;
```

### 4.2.12 dns.sel\_custom

sel custom

**Parameters** *None*

**Variables defined for body**

- v\_owner *user.t\_user*

**Returns** TABLE

**Returned columns**

- id *uuid*

- registered *dns.t\_hostname*
- domain *dns.t\_domain*
- type *dns.t\_type*
- rdata *dns.t\_rdata*
- ttl *dns.t\_ttl*
- backend\_status *backend.t\_status*

**Execute privilege**

- *userlogin*

```
-- begin userlogin prelude
v_owner := (SELECT t.act_as FROM "user"._get_login() AS t);
-- end userlogin prelude

RETURN QUERY
  SELECT
    t.id,
    t.registered,
    t.domain,
    t.type,
    t.rdata,
    t.ttl,
    t.backend_status
  FROM dns.custom AS t
  JOIN dns.registered AS s
    ON s.domain = t.registered
  WHERE
    s.owner = v_owner
  ORDER BY backend_status, registered, dns._domain_order(t.domain);
```

## 4.2.13 dns.sel\_nameserver

Select available nameservers

**Parameters** *None*

**Variables defined for body**

- v\_owner *user.t\_user*

**Returns** TABLE

**Returned columns**

- subservice *commons.t\_key*
- service\_entity\_name *dns.t\_hostname*

**Execute privilege**

- *userlogin*

```
-- begin userlogin prelude
v_owner := (SELECT t.act_as FROM "user"._get_login() AS t);
-- end userlogin prelude
```

(continues on next page)

(continued from previous page)

```

RETURN QUERY
SELECT
    COALESCE(t.subservice, s.subservice) AS subservice,
    COALESCE(t.service_entity_name, s.service_entity_name) AS service_entity_name
FROM system._effective_contingent() AS t

FULL OUTER JOIN system._effective_contingent_domain() AS s
    USING (service, subservice, service_entity_name, owner)

WHERE
    COALESCE(t.service, s.service) = 'domain_registered' AND
    COALESCE(t.owner, s.owner) = v_owner

ORDER BY subservice, service_entity_name
;

```

#### 4.2.14 dns.sel\_registered

List registered domains

Parameters *None*

Variables defined for body

- v\_owner *user.t\_user*

Returns TABLE

Returned columns

- domain *dns.t\_hostname*
- public\_suffix *varchar*
- backend\_status *backend.t\_status*
- subservice *commons.t\_key*
- service\_entity\_name *dns.t\_hostname*

Execute privilege

- *userlogin*

```

-- begin userlogin prelude
v_owner := (SELECT t.act_as FROM "user"._get_login() AS t);
-- end userlogin prelude

RETURN QUERY
    SELECT t.domain, t.public_suffix, t.backend_status, t.subservice, t.service_
    ↪entity_name
    FROM dns.registered AS t
    WHERE
        t.owner = v_owner
    ORDER BY backend_status, domain;

```

### 4.2.15 dns.sel\_service

Select service based dns entries

**Parameters** *None*

**Variables defined for body**

- v\_owner *user:t\_user*

**Returns** TABLE

**Returned columns**

- registered *dns.t\_hostname*
- domain *dns.t\_hostname*
- service *commons.t\_key*
- service\_entity\_name *dns.t\_hostname*
- backend\_status *backend.t\_status*

**Execute privilege**

- *userlogin*

```
-- begin userlogin prelude
v_owner := (SELECT t.act_as FROM "user"._get_login() AS t);
-- end userlogin prelude

RETURN QUERY
SELECT
    t.registered,
    t.domain,
    t.service,
    t.service_entity_name,
    t.backend_status
FROM dns.service AS t
JOIN dns.registered AS s
    ON s.domain = t.registered
WHERE
    s.owner = v_owner
ORDER BY backend_status, registered, dns._domain_order(t.domain), service, ↵
↵service_entity_name;
```

### 4.2.16 dns.sel\_usable\_domain

Usable domains

**Parameters**

- p\_service *commons.t\_key*
- p\_subservice *commons.t\_key*

**Variables defined for body**

- v\_owner *user:t\_user*

**Returns** TABLE



**Returned columns**

- domain *dns.t\_hostname*
- service\_entity\_name *dns.t\_hostname*

**Execute privilege**

- *userlogin*

```
-- begin userlogin prelude
v_owner := (SELECT t.act_as FROM "user"._get_login() AS t);
-- end userlogin prelude

RETURN QUERY
SELECT t.domain, t.service_entity_name FROM dns.service AS t
  JOIN dns.registered AS d
    ON d.domain = t.registered
  LEFT JOIN system._effective_contingent_domain() AS contingent_d
    ON
      contingent_d.domain = t.domain AND
      contingent_d.service = t.service AND
      contingent_d.subservice = p_subservice AND
      contingent_d.service_entity_name = t.service_entity_name AND
      contingent_d.owner = v_owner

  LEFT JOIN system._effective_contingent() AS contingent
    ON
      contingent.service = t.service AND
      contingent.subservice = p_subservice AND
      contingent.owner = v_owner AND
      d.owner = v_owner

WHERE
  t.service = p_service AND
  COALESCE(contingent_d.domain_contingent, contingent.domain_contingent, 0) > 0
ORDER BY
  t.domain
;
```

**4.2.17 dns.srv\_record**

Servers both record types combined: Raw entries and the ones assembled from records templates for services (system.service\_entity\_dns).

**Parameters**

- p\_include\_inactive *boolean*

**Returns TABLE****Returned columns**

- registered *dns.t\_hostname*
- domain *dns.t\_domain*
- type *dns.t\_type*
- rdata *dns.t\_rdata*

- ttl *dns.t\_ttl*
- backend\_status *backend.t\_status*

### Execute privilege

- *backend*

```
PERFORM backend._get_login();

RETURN QUERY
WITH

-- DELETE
d_s AS (
    DELETE FROM dns.service AS t
    USING dns.registered AS s
    WHERE
        s.domain = t.registered AND
        backend._deleted(t.backend_status) AND
        backend._machine_priviledged_service('dns', s.service_entity_name)
),

d_c AS (
    DELETE FROM dns.custom AS t
    USING dns.registered AS s
    WHERE
        s.domain = t.registered AND
        backend._deleted(t.backend_status) AND
        backend._machine_priviledged_service('dns', s.service_entity_name)
),

-- UPDATE
u_s AS (
    UPDATE dns.service AS t
        SET backend_status = NULL
    FROM dns.registered AS s
    WHERE
        s.domain = t.registered AND
        backend._machine_priviledged_service('dns', s.service_entity_name) AND
        backend._active(t.backend_status)
),

u_c AS (
    UPDATE dns.custom AS t
        SET backend_status = NULL
    FROM dns.registered AS s
    WHERE
        s.domain = t.registered AND
        backend._machine_priviledged_service('dns', s.service_entity_name) AND
        backend._active(t.backend_status)
)

SELECT
    t.registered,
    COALESCE(s.domain_prefix || t.domain, t.domain)::dns.t_domain,
    s.type,
    s.rdata,
```

(continues on next page)

(continued from previous page)

```

        s.ttl,
        t.backend_status
FROM dns.service AS t
JOIN system.service_entity_dns AS s
    USING (service, service_entity_name)
JOIN dns.registered AS u
    ON t.registered = u.domain
WHERE
    u.subservice = 'managed' AND
    backend._machine_priviledged_service('dns', u.service_entity_name) AND
    (backend._active(t.backend_status) OR p_include_inactive)

UNION ALL

SELECT
    t.registered,
    t.domain,
    t.type,
    t.rdata,
    t.ttl,
    t.backend_status
FROM dns.custom AS t
JOIN dns.registered AS u
    ON t.registered = u.domain
WHERE
    u.subservice = 'managed' AND
    backend._machine_priviledged_service('dns', u.service_entity_name) AND
    (backend._active(t.backend_status) OR p_include_inactive)
;

```

### 4.2.18 dns.upd\_custom

Ins Custom

#### Parameters

- p\_id *uuid*
- p\_rdata *dns.t\_rdata*
- p\_ttl *integer*

#### Variables defined for body

- v\_nameserver *dns.t\_hostname*
- v\_managed *commons.t\_key*
- v\_owner *user.t\_user*

Returns void

#### Execute privilege

- *userlogin*

```

-- begin userlogin prelude
v_owner := (SELECT t.act_as FROM "user"._get_login() AS t);
-- end userlogin prelude

```

(continues on next page)

(continued from previous page)

```

UPDATE dns.custom AS t
  SET
    rdata = p_rdata,
    ttl = p_ttl,
    backend_status = 'upd'
FROM dns.registered AS s

WHERE
  s.domain = t.registered AND

  t.id = p_id AND
  s.owner = v_owner
RETURNING s.service_entity_name, s.subservice INTO v_nameserver, v_managed;

PERFORM backend._notify_service_entity_name(v_nameserver, 'dns', v_managed);

```

## 4.3 Domains

### 4.3.1 dns.t\_domain

Fully qualified hostname (without trailing dot)

#### Checks

- **hostname valid regex** Hostname

```

VALUE ~ '^[a-z\d_-]{1,63}(\.[a-z\d_-]{1,63})+${}' AND
octet_length(VALUE) <= 253

```

### 4.3.2 dns.t\_domain\_rdata

Fully qualified or relative domain name. Trailing dot marks a FQDN.

---

**Todo:** checks might be off

---

#### Checks

- **invalid rdata domain** check

```

(VALUE ~ '^[a-z\d][a-z\d-]{0,62}\.' OR
VALUE ~ '^[a-z\d][a-z\d-]{0,62}\.*[a-z\d][a-z\d-]{1,63}$') AND
octet_length(VALUE) <= 253

```

### 4.3.3 dns.t\_hostname

Fully qualified hostname (without trailing dot)

#### Checks

- **hostname valid regex** Hostname

```
VALUE ~ '^[a-z\d]|[a-z\d][a-z\d-]{0,61}[a-z\d])(\.[a-z\d]|[a-z\d][a-z\d-]{0,61}[a-z\d]))+$' AND
octet_length(VALUE) <= 253
```

#### 4.3.4 dns.t\_rdata

Resource record data (Rdata)

#### 4.3.5 dns.t\_ttl

time to live

Checks

- **ttl range** Ensure that TTL is at least one minute and put maximum to 48h

```
VALUE BETWEEN 60 AND EXTRACT(EPOCH FROM INTERVAL '2 days')
```

#### 4.3.6 dns.t\_type

Resource record type

Checks

- **Invalid or unsupported resource type** Resource type (A, AAAA, CNAME, MX, SRV, TXT,...)

```
VALUE IN (
  'A',
  'AAAA',
  'CNAME',
  'MX',
  'NS',
  'SRV',
  'SSHFP',
  'TXT'
)
```



---

## domain\_reseller

---

Features for Domains Registered via a Reseller

Stores additional details for dns.registered domains. Also supports storing contact informations (handles).

**This module sends the following signals:**

- domain\_reseller/handle
- domain\_registered/managed
- domain\_registered/unmanaged

### Schema Contents

- *Tables*
  - `domain_reseller.handle`
  - `domain_reseller.registered`
- *Functions*
  - `domain_reseller.del_handle`
  - `domain_reseller.fwd_handle_id`
  - `domain_reseller.fwd_registered_status`
  - `domain_reseller.ins_handle`
  - `domain_reseller.ins_registered`
  - `domain_reseller.sel_handle`
  - `domain_reseller.sel_registered`
  - `domain_reseller.sel_reseller`
  - `domain_reseller.srv_handle`

```
- domain_reseller.srv_registered
- domain_reseller.upd_handle
- domain_reseller.upd_registered
```

## 5.1 Tables

### 5.1.1 domain\_reseller.handle

Handles (Domain Contacts)

Domain contacts that can be used as owner, admin-c, tech-c or zone-c.

#### Primary key

- alias

#### Foreign keys

- Reference service entity

##### Local Columns

- service\_entity\_name
- service

##### Referenced Columns

- *system.service\_entity.service\_entity\_name*
- *system.service\_entity.service*

- Reference subservice entity

##### Local Columns

- service\_entity\_name
- service
- subservice

##### Referenced Columns

- *system.subservice\_entity.service\_entity\_name*
- *system.subservice\_entity.service*
- *system.subservice\_entity.subservice*

#### Columns

- **service\_entity\_name** *dns.t\_hostname* Service entity name
- **service** *commons.t\_key* Service (e.g. email, jabber)
- **subservice** *commons.t\_key* Subservice (e.g. account, alias)
- **owner** *user.t\_user* Owner

References *user.user.owner*

On Update: CASCADE



- **backend\_status** *NULL* | *backend.t\_status* Status of database entry in backend. *NULL*: nothing pending, 'ins': entry not present on backend client, 'upd': update pending on backend client, 'del': deletion pending on backend client.

**Default**

'ins'

- **alias** *varchar* Free choosable alias
- **id** *NULL* | *varchar* Internal id at reseller
- **fname** *varchar* First name
- **lname** *varchar* Last name
- **address** *varchar* Address
- **pcode** *varchar* Postcode
- **city** *varchar* City
- **country** *varchar* Country
- **state** *varchar* State
- **email** *email.t\_address* Email
- **phone** *varchar* Phone
- **organization** *NULL* | *varchar* Organization
- **fax** *NULL* | *varchar* Fax
- **mobile\_phone** *NULL* | *varchar* Mobile phone

### 5.1.2 domain\_reseller.registered

Additional informations to those stored in dns.registered

**Primary key**

- domain

**Columns**

- **domain** *dns.t\_hostname* Domain  
References *dns.registered.domain*  
On Delete: CASCADE
- **registrant** *varchar* Registrant (Owner)  
References *domain\_reseller.handle.alias*
- **admin\_c** *varchar* Admin-C  
References *domain\_reseller.handle.alias*
- **tech\_c** *NULL* | *varchar* Tech-C  
References *domain\_reseller.handle.alias*
- **zone\_c** *NULL* | *varchar* Zone-C  
References *domain\_reseller.handle.alias*

- **payable** *NULL | timestamp* Payable
- **period** *NULL | integer* Renewal period (years)
- **registrar\_status** *NULL | varchar* Registrar status
- **registry\_status** *NULL | varchar* Registry status
- **last\_status** *NULL | varchar* Last update status

## 5.2 Functions

### 5.2.1 domain\_reseller.del\_handle

Deletes handle

#### Parameters

- **p\_alias** *varchar*

#### Variables defined for body

- **v\_service\_entity\_name** *dns.t\_hostname*
- **v\_owner** *user.t\_user*

**Returns** void

#### Execute privilege

- *userlogin*

```
-- begin userlogin prelude
v_owner := (SELECT t.act_as FROM "user"._get_login() AS t);
-- end userlogin prelude

BEGIN
    -- perform DELETE to trigger potential foreign key errors
    DELETE FROM domain_reseller.handle
    WHERE
        alias = p_alias AND
        owner = v_owner;

    -- if not failed yet, emulate rollback of DELETE
    RAISE transaction_rollback;
EXCEPTION
    WHEN transaction_rollback THEN
        UPDATE domain_reseller.handle
            SET backend_status = 'del'
        WHERE
            alias = p_alias AND
            owner = v_owner
        RETURNING service_entity_name INTO v_service_entity_name;

    PERFORM backend._conditional_notify_service_entity_name(
        FOUND, v_service_entity_name, 'domain_reseller', 'handle'
    );
END;
```

### 5.2.2 domain\_reseller.fwd\_handle\_id

Insert handle id

#### Parameters

- p\_alias *varchar*
- p\_id *varchar*
- p\_include\_inactive *boolean*

**Returns** void

#### Execute privilege

- *backend*

```
PERFORM backend._get_login();

UPDATE domain_reseller.handle
  SET id = p_id
  WHERE alias = p_alias;
```

### 5.2.3 domain\_reseller.fwd\_registered\_status

Update status

#### Parameters

- p\_domain *dns.t\_hostname*
- p\_payable *timestamp*
- p\_period *integer*
- p\_registrar\_status *varchar*
- p\_registry\_status *varchar*
- p\_last\_status *varchar*
- p\_include\_inactive *boolean*

**Returns** void

#### Execute privilege

- *backend*

```
PERFORM backend._get_login();

UPDATE domain_reseller.registered
  SET
    payable = p_payable,
    period = p_period,
    registrar_status = p_registrar_status,
    registry_status = p_registry_status,
    last_status = p_last_status
  WHERE domain = p_domain;
```

## 5.2.4 domain\_reseller.ins\_handle

Inserts handle

### Parameters

- p\_alias *varchar*
- p\_service\_entity\_name *dns.t\_hostname*
- p\_fname *varchar*
- p\_lname *varchar*
- p\_address *varchar*
- p\_pcode *varchar*
- p\_city *varchar*
- p\_country *varchar*
- p\_state *varchar*
- p\_email *email.t\_address*
- p\_phone *varchar*
- p\_organization *varchar*
- p\_fax *varchar*
- p\_mobile\_phone *varchar*

### Variables defined for body

- v\_owner *user.t\_user*

**Returns** void

### Execute privilege

- *userlogin*

```
-- begin userlogin prelude
v_owner := (SELECT t.act_as FROM "user"._get_login() AS t);
-- end userlogin prelude

INSERT INTO domain_reseller.handle
(
  service_entity_name,
  service,
  subservice,
  owner,
  alias,
  fname,
  lname,
  address,
  pcode,
  city,
  country,
  state,
  email,
  phone,
```

(continues on next page)

(continued from previous page)

```

organization,
fax,
mobile_phone
)
VALUES
(
  p_service_entity_name,
  'domain_reseller',
  'handle',
  v_owner,
  p_alias,
  p_fname,
  p_lname,
  p_address,
  p_pcode,
  p_city,
  p_country,
  p_state,
  p_email,
  p_phone,
  p_organization,
  p_fax,
  p_mobile_phone
);

PERFORM backend._notify_service_entity_name(p_service_entity_name, 'domain_reseller',
↪ 'handle');
```

### 5.2.5 domain\_reseller.ins\_registered

Inserts details for registered domain

#### Parameters

- p\_domain *dns.t\_hostname*
- p\_registrant *varchar*
- p\_admin\_c *varchar*

#### Variables defined for body

- v\_owner *user.t\_user*

Returns void

#### Execute privilege

- userlogin

```

-- begin userlogin prelude
v_owner := (SELECT t.act_as FROM "user"._get_login() AS t);
-- end userlogin prelude

INSERT INTO domain_reseller.registered
  (domain, registrant, admin_c)
VALUES
  (p_domain, p_registrant, p_admin_c);
```

### 5.2.6 domain\_reseller.sel\_handle

Selects handles

#### Parameters

- `p_hide_foreign` *bool*

#### Variables defined for body

- `v_owner` *user:t\_user*

**Returns** SETOF domain\_reseller."handle"

#### Execute privilege

- *userlogin*

```
-- begin userlogin prelude
v_owner := (SELECT t.act_as FROM "user"._get_login() AS t);
-- end userlogin prelude

RETURN QUERY
    SELECT * FROM domain_reseller.handle
WHERE
    owner=v_owner OR (owner="user"._login_user() AND NOT p_hide_foreign)
ORDER BY backend_status, fname, lname, alias;
```

### 5.2.7 domain\_reseller.sel\_registered

Selects details for registered domains

#### Parameters *None*

#### Variables defined for body

- `v_owner` *user:t\_user*

**Returns** TABLE

#### Returned columns

- domain *dns.t\_hostname*
- registrant *varchar*
- admin\_c *varchar*
- tech\_c *varchar*
- zone\_c *varchar*
- payable *timestamp*
- period *integer*
- registrar\_status *varchar*
- registry\_status *varchar*
- last\_status *varchar*
- backend\_status *backend.t\_status*

#### Execute privilege

- *userlogin*

```
-- begin userlogin prelude
v_owner := (SELECT t.act_as FROM "user"._get_login() AS t);
-- end userlogin prelude

RETURN QUERY
  SELECT
    t.domain,
    t.registrant,
    t.admin_c,
    t.tech_c,
    t.zone_c,
    t.payable,
    t.period,
    t.registrar_status,
    t.registry_status,
    t.last_status,
    s.backend_status
  FROM domain_reseller.registered AS t
  JOIN dns.registered AS s
    USING (domain)
  WHERE
    s.owner = v_owner
  ORDER BY backend_status, domain
;
```

## 5.2.8 domain\_reseller.sel\_reseller

Selects available resellers

**Parameters** *None*

**Variables defined for body**

- v\_owner *user:t\_user*

**Returns** TABLE

**Returned columns**

- subservice *commons.t\_key*
- service\_entity\_name *dns.t\_hostname*

**Execute privilege**

- *userlogin*

```
-- begin userlogin prelude
v_owner := (SELECT t.act_as FROM "user"._get_login() AS t);
-- end userlogin prelude

RETURN QUERY
  SELECT
    COALESCE(t.subservice, s.subservice) AS subservice,
    COALESCE(t.service_entity_name, s.service_entity_name) AS service_entity_name
  FROM system._effective_contingent() AS t
```

(continues on next page)

(continued from previous page)

```

FULL OUTER JOIN system._effective_contingent_domain() AS s
USING (service, subservice, service_entity_name, owner)
WHERE
    COALESCE(t.service, s.service) = 'domain_reseller' AND
    COALESCE(t.owner, s.owner) = v_owner

ORDER BY subservice, service_entity_name
;

```

### 5.2.9 domain\_reseller.srv\_handle

Serves handles

#### Parameters

- p\_include\_inactive *boolean*

**Returns** SETOF domain\_reseller.”handle”

#### Execute privilege

- *backend*

```

PERFORM backend._get_login();

RETURN QUERY
WITH
    -- DELETE
    d AS (
        DELETE FROM domain_reseller.handle AS t
        WHERE
            backend._machine_priviledged_service(t.service, t.service_entity_name) AND
            backend._deleted(t.backend_status)
    ),
    -- UPDATE
    s AS (
        UPDATE domain_reseller.handle AS t
        SET backend_status = NULL
        WHERE
            backend._machine_priviledged_service(t.service, t.service_entity_name) AND
            backend._active(t.backend_status)
    )

SELECT * FROM domain_reseller.handle AS t
WHERE
    backend._machine_priviledged_service(t.service, t.service_entity_name) AND
    (backend._active(t.backend_status) OR p_include_inactive);

```

### 5.2.10 domain\_reseller.srv\_registered

Serves details for registered domains

#### Parameters



- `p_include_inactive` *boolean*

**Returns** TABLE

**Returned columns**

- `domain` *dns.t\_hostname*
- `registrant` *varchar*
- `registrant_id` *varchar*
- `admin_c` *varchar*
- `admin_c_id` *varchar*
- `tech_c` *varchar*
- `tech_c_id` *varchar*
- `zone_c` *varchar*
- `zone_c_id` *varchar*
- `backend_status` *backend.t\_status*

**Execute privilege**

- *backend*

```
PERFORM backend._get_login();

RETURN QUERY
SELECT
    t.domain,
    t.registrant,
    (SELECT id FROM domain_reseller.handle WHERE alias = t.registrant),
    t.admin_c,
    (SELECT id FROM domain_reseller.handle WHERE alias = t.admin_c),
    t.tech_c,
    (SELECT id FROM domain_reseller.handle WHERE alias = t.tech_c),
    t.zone_c,
    (SELECT id FROM domain_reseller.handle WHERE alias = t.zone_c),
    s.backend_status
FROM domain_reseller.registered AS t
JOIN dns.registered AS s USING (domain)
WHERE
    backend._machine_priviledged_service(s.service, s.service_entity_name) AND
    (backend._active(s.backend_status) OR p_include_inactive);
```

## 5.2.11 domain\_reseller.upd\_handle

Updates handle

**Parameters**

- `p_alias` *varchar*
- `p_address` *varchar*
- `p_pcode` *varchar*
- `p_city` *varchar*

- p\_country *varchar*
- p\_state *varchar*
- p\_email *email.t\_address*
- p\_phone *varchar*
- p\_organization *varchar*
- p\_fax *varchar*
- p\_mobile\_phone *varchar*

**Variables defined for body**

- v\_service\_entity\_name *dns.t\_hostname*
- v\_owner *user.t\_user*

**Returns** void

**Execute privilege**

- *userlogin*

```
-- begin userlogin prelude
v_owner := (SELECT t.act_as FROM "user"._get_login() AS t);
-- end userlogin prelude

UPDATE domain_reseller.handle
  SET
    backend_status = 'upd',
    address = p_address,
    pcode = p_pcode,
    city = p_city,
    country = p_country,
    state = p_state,
    email = p_email,
    phone = p_phone,
    organization = p_organization,
    fax = p_fax,
    mobile_phone = p_mobile_phone

WHERE
  alias = p_alias AND
  owner = v_owner
RETURNING service_entity_name INTO v_service_entity_name;

PERFORM backend._conditional_notify_service_entity_name(
  FOUND, v_service_entity_name, 'domain_reseller', 'handle'
);
```

## 5.2.12 domain\_reseller.upd\_registered

Updates details for registered domain

**Parameters**

- p\_domain *dns.t\_hostname*
- p\_admin\_c *varchar*

**Variables defined for body**

- v\_nameserver *dns.t\_hostname*
- v\_managed *commons.t\_key*
- v\_owner *user.t\_user*

**Returns** void**Execute privilege**

- *userlogin*

```
-- begin userlogin prelude
v_owner := (SELECT t.act_as FROM "user"._get_login() AS t);
-- end userlogin prelude

UPDATE domain_reseller.registered AS t
  SET
    admin_c = p_admin_c
FROM dns.registered AS s
WHERE
  s.domain = t.domain AND
  s.owner = v_owner AND

  t.domain = p_domain;

UPDATE dns.registered AS t
  SET backend_status = 'upd'
WHERE
  t.owner = v_owner AND
  t.domain = p_domain AND
  -- don't change domains that are in some transition status
  (t.backend_status = 'upd' OR t.backend_status IS NULL)
RETURNING t.service_entity_name, t.subservice
  INTO v_nameserver, v_managed;

PERFORM backend._conditional_notify_service_entity_name(
  FOUND, v_nameserver, 'domain_registered', v_managed
);
```



### Email and Mailing lists

**This module sends the following signals:**

- email/alias
- email/list
- email/mailbox
- email/redirection

#### Schema Contents

- *Tables*

- `email.address`
- `email.alias`
- `email.list`
- `email.list_subscriber`
- `email.mailbox`
- `email.redirection`

- *Functions*

- `email._address`
- `email._address_valid`
- `email.del_alias`
- `email.del_list`
- `email.del_list_subscriber`

- *email.del\_mailbox*
- *email.del\_redirection*
- *email.ins\_alias*
- *email.ins\_list*
- *email.ins\_list\_subscriber*
- *email.ins\_mailbox*
- *email.ins\_redirection*
- *email.sel\_alias*
- *email.sel\_list*
- *email.sel\_list\_subscriber*
- *email.sel\_mailbox*
- *email.sel\_redirection*
- *email.srv\_alias*
- *email.srv\_list*
- *email.srv\_list\_subscriber*
- *email.srv\_mailbox*
- *email.srv\_redirection*
- *email.upd\_list*
- *email.upd\_mailbox*
- *Domains*
  - *email.t\_localpart*
  - *email.t\_address*

## 6.1 Tables

### 6.1.1 `email.address`

Collection of all known addresses

#### Primary key

- localpart
- domain

#### Foreign keys

- reference dns (service)

#### Local Columns

- domain
- service

- service\_entity\_name

**Referenced Columns**

- *dns.service.domain*
- *dns.service.service*
- *dns.service.service\_entity\_name*

- Reference subservice entity

**Local Columns**

- service\_entity\_name
- service
- subservice

**Referenced Columns**

- *system.subservice\_entity.service\_entity\_name*
- *system.subservice\_entity.service*
- *system.subservice\_entity.subservice*

**Columns**

- **domain** *dns.t\_hostname* Domain name
- **service** *commons.t\_key* Service
- **service\_entity\_name** *dns.t\_hostname* ent. name
- **subservice** *commons.t\_key* Subservice (e.g. account, alias)
- **localpart** *email.t\_localpart* Local part

### 6.1.2 email.alias

Aliases for e-mail mailboxes, owner is determined by mailbox.owner

**Primary key**

- localpart
- domain

**Foreign keys**

- reference dns (service)

**Local Columns**

- domain
- service
- service\_entity\_name

**Referenced Columns**

- *dns.service.domain*
- *dns.service.service*
- *dns.service.service\_entity\_name*

- Reference subservice entity

**Local Columns**

- service\_entity\_name
- service
- subservice

**Referenced Columns**

- *system.subservice\_entity.service\_entity\_name*
- *system.subservice\_entity.service*
- *system.subservice\_entity.subservice*

- reference to a mailbox

**Local Columns**

- mailbox\_localpart
- mailbox\_domain

**Referenced Columns**

- *email.mailbox.localpart*
- *email.mailbox.domain*

**Columns**

- **domain** *dns.t\_hostname* Domain name
- **service** *commons.t\_key* Service
- **service\_entity\_name** *dns.t\_hostname* ent. name
- **subservice** *commons.t\_key* Subservice (e.g. account, alias)
- **backend\_status** *NULL | backend.t\_status* Status of database entry in backend. NULL: nothing pending, 'ins': entry not present on backend client, 'upd': update pending on backend client, 'del': deletion pending on backend client.

**Default**

'ins'

- **localpart** *email.t\_localpart* Local part
- **mailbox\_localpart** *email.t\_localpart* Mailbox to which the mails will be delivered
- **mailbox\_domain** *dns.t\_hostname* Mailbox to which the mails will be delivered

### 6.1.3 email.list

Mailing lists

**Primary key**

- localpart
- domain

**Foreign keys**



- reference dns (service)

#### Local Columns

- domain
- service
- service\_entity\_name

#### Referenced Columns

- *dns.service.domain*
- *dns.service.service*
- *dns.service.service\_entity\_name*

- Reference subservice entity

#### Local Columns

- service\_entity\_name
- service
- subservice

#### Referenced Columns

- *system.subservice\_entity.service\_entity\_name*
- *system.subservice\_entity.service*
- *system.subservice\_entity.subservice*

### Columns

- **domain** *dns.t\_hostname* Domain name
- **service** *commons.t\_key* Service
- **service\_entity\_name** *dns.t\_hostname* ent. name
- **subservice** *commons.t\_key* Subservice (e.g. account, alias)
- **owner** *user.t\_user* Owner

References *user.user.owner*

On Update: CASCADE

- **backend\_status** *NULL* | *backend.t\_status* Status of database entry in backend. NULL: nothing pending, 'ins': entry not present on backend client, 'upd': update pending on backend client, 'del': deletion pending on backend client.

#### Default

```
'ins'
```

- **option** *jsonb* Free options in JSON format

#### Default

```
'{}'
```

- **localpart** *email.t\_localpart* Local part of the email list address
- **admin** *email.t\_address* Email address of the list admin

- **options** *NULL | jsonb* Arbitrary options

### 6.1.4 email.list\_subscriber

list subscribers

#### Primary key

- address
- list\_localpart
- list\_domain

#### Foreign keys

- reference to a list

#### Local Columns

- list\_localpart
- list\_domain

#### Referenced Columns

- *email.list.localpart*
- *email.list.domain*

#### Columns

- **backend\_status** *NULL | backend.t\_status* Status of database entry in backend. NULL: nothing pending, 'ins': entry not present on backend client, 'upd': update pending on backend client, 'del': deletion pending on backend client.

#### Default

```
'ins'
```

- **option** *jsonb* Free options in JSON format

#### Default

```
'{}'
```

- **address** *email.t\_address* Subscribers address
- **list\_localpart** *email.t\_localpart* List
- **list\_domain** *dns.t\_hostname* List

### 6.1.5 email.mailbox

E-mail mailboxes correspond to something a mail user can login into. Basically a mailbox represents a mailbox. A mailbox is bound to a specific address. Further addresses can be linked to mailboxes via aliases.

#### Primary key

- localpart
- domain

#### Foreign keys

- reference dns (service)

#### Local Columns

- domain
- service
- service\_entity\_name

#### Referenced Columns

- *dns.service.domain*
- *dns.service.service*
- *dns.service.service\_entity\_name*

- Reference subservice entity

#### Local Columns

- service\_entity\_name
- service
- subservice

#### Referenced Columns

- *system.subservice\_entity.service\_entity\_name*
- *system.subservice\_entity.service*
- *system.subservice\_entity.subservice*

### Columns

- **domain** *dns.t\_hostname* Domain name
- **service** *commons.t\_key* Service
- **service\_entity\_name** *dns.t\_hostname* ent. name
- **subservice** *commons.t\_key* Subservice (e.g. account, alias)
- **owner** *user.t\_user* Owner

References *user.user.owner*

On Update: CASCADE

- **backend\_status** *NULL* | *backend.t\_status* Status of database entry in backend. NULL: nothing pending, 'ins': entry not present on backend client, 'upd': update pending on backend client, 'del': deletion pending on backend client.

#### Default

```
'ins'
```

- **option** *jsonb* Free options in JSON format

#### Default

```
'{}'
```

- **uid** *integer* Unix user identifier

#### Default

```
nextval('commons.uid')
```

- **localpart** *email.t\_localpart* Local part
- **password** *commons.t\_password* Unix shadow crypt format
- **quota** *NULL | int* Quota for mailbox in MiB

### 6.1.6 email.redirection

Redirections

#### Primary key

- localpart
- domain

#### Foreign keys

- reference dns (service)

##### Local Columns

- domain
- service
- service\_entity\_name

##### Referenced Columns

- *dns.service.domain*
- *dns.service.service*
- *dns.service.service\_entity\_name*

- Reference subservice entity

##### Local Columns

- service\_entity\_name
- service
- subservice

##### Referenced Columns

- *system.subservice\_entity.service\_entity\_name*
- *system.subservice\_entity.service*
- *system.subservice\_entity.subservice*

#### Columns

- **domain** *dns.t\_hostname* Domain name
- **service** *commons.t\_key* Service
- **service\_entity\_name** *dns.t\_hostname* ent. name
- **subservice** *commons.t\_key* Subservice (e.g. account, alias)

- **owner** *user.t\_user* Owner

References *user.user.owner*

On Update: CASCADE

- **backend\_status** *NULL* | *backend.t\_status* Status of database entry in backend. NULL: nothing pending, 'ins': entry not present on backend client, 'upd': update pending on backend client, 'del': deletion pending on backend client.

#### Default

```
'ins'
```

- **localpart** *email.t\_localpart* Local part
- **destination** *email.t\_address* External address to which the mails will be delivered

## 6.2 Functions

### 6.2.1 email.\_address

List all addresses

**Parameters** *None*

**Returns** TABLE

**Returned columns**

- localpart *email.t\_localpart*
- domain *dns.t\_hostname*
- owner *user.t\_user*
- subservice *commons.t\_key*

```
RETURN QUERY (
  SELECT t.localpart, t.domain, t.owner, t.subservice FROM email.mailbox AS t
  UNION ALL
  SELECT t.localpart, t.domain, t.owner, t.subservice FROM email.redirection AS t
  UNION ALL
  SELECT t.localpart, t.domain, s.owner, t.subservice FROM email.alias AS t
  LEFT JOIN email.mailbox AS s
    ON
      t.mailbox_localpart = s.localpart AND
      t.mailbox_domain = s.domain
  UNION ALL
  SELECT t.localpart, t.domain, t.owner, t.subservice FROM email.list AS t
);
```

### 6.2.2 email.\_address\_valid

x

**Parameters**

- p\_localpart *email.t\_localpart*

- p\_domain *dns.t\_hostname*

**Returns** void

```
IF (
    SELECT TRUE FROM email._address()
    WHERE
        localpart = p_localpart AND
        domain = p_domain
) THEN
    RAISE 'Email address already exists.'
    USING DETAIL = '$carnivora:email:address_already_exists$';
END IF;
```

### 6.2.3 email.del\_alias

Delete Alias

**Parameters**

- p\_localpart *email.t\_localpart*
- p\_domain *dns.t\_hostname*
- p\_mailbox\_localpart *email.t\_localpart*
- p\_mailbox\_domain *dns.t\_hostname*

**Variables defined for body**

- v\_owner *usert.user*

**Returns** void

**Execute privilege**

- *userlogin*

```
-- begin userlogin prelude
v_owner := (SELECT t.act_as FROM "user"._get_login() AS t);
-- end userlogin prelude

UPDATE email.alias AS t
    SET backend_status = 'del'
FROM email.mailbox AS s
WHERE
    -- JOIN
    t.mailbox_localpart = s.localpart AND
    t.mailbox_domain = s.domain AND

    t.localpart = p_localpart AND
    t.domain = p_domain AND
    s.localpart = p_mailbox_localpart AND
    s.domain = p_mailbox_domain AND

    s.owner = v_owner;

PERFORM backend._conditional_notify(FOUND, 'email', 'alias', p_domain);
```

### 6.2.4 email.del\_list

Delete mailing list

#### Parameters

- p\_domain *dns.t\_hostname*
- p\_localpart *email.t\_localpart*

#### Variables defined for body

- v\_owner *user.t\_user*

Returns void

#### Execute privilege

- *userlogin*

```
-- begin userlogin prelude
v_owner := (SELECT t.act_as FROM "user"._get_login() AS t);
-- end userlogin prelude

DELETE FROM email.list
WHERE
    domain = p_domain AND
    localpart = p_localpart AND
    owner = v_owner;

PERFORM backend._conditional_notify(FOUND, 'email', 'list', p_domain);
```

### 6.2.5 email.del\_list\_subscriber

del

#### Parameters

- p\_list\_localpart *email.t\_localpart*
- p\_list\_domain *dns.t\_hostname*
- p\_address *email.t\_address*

#### Variables defined for body

- v\_owner *user.t\_user*

Returns void

#### Execute privilege

- *userlogin*

```
-- begin userlogin prelude
v_owner := (SELECT t.act_as FROM "user"._get_login() AS t);
-- end userlogin prelude

UPDATE email.list_subscriber AS t
SET backend_status = 'del'
```

(continues on next page)

(continued from previous page)

```

FROM email.list AS s
WHERE
    s.localpart = t.list_localpart AND
    s.domain = t.list_domain AND
    s.owner = v_owner AND

    t.list_localpart = p_list_localpart AND
    t.list_domain = p_list_domain AND
    t.address = p_address;

PERFORM backend._conditional_notify(FOUND, 'email', 'list', p_list_domain);

```

## 6.2.6 email.del\_mailbox

Delete mailbox

### Parameters

- p\_localpart *email.t\_localpart*
- p\_domain *dns.t\_hostname*

### Variables defined for body

- v\_owner *user.t\_user*

Returns void

### Execute privilege

- *userlogin*

```

-- begin userlogin prelude
v_owner := (SELECT t.act_as FROM "user"._get_login() AS t);
-- end userlogin prelude

UPDATE email.mailbox
    SET backend_status = 'del'
WHERE
    localpart = p_localpart AND
    domain = p_domain AND
    owner = v_owner;

PERFORM backend._conditional_notify(FOUND, 'email', 'mailbox', p_domain);

```

## 6.2.7 email.del\_redirection

Delete redirection

### Parameters

- p\_localpart *email.t\_localpart*
- p\_domain *dns.t\_hostname*

### Variables defined for body



- v\_owner *user.t\_user*

**Returns** void

**Execute privilege**

- *userlogin*

```
-- begin userlogin prelude
v_owner := (SELECT t.act_as FROM "user"._get_login() AS t);
-- end userlogin prelude

UPDATE email.redirection
  SET backend_status = 'del'
  WHERE
    localpart = p_localpart AND
    domain = p_domain AND
    owner = v_owner;

PERFORM backend._conditional_notify(FOUND, 'email', 'redirection', p_domain);
```

## 6.2.8 email.ins\_alias

Create e-mail aliases

**Parameters**

- p\_localpart *email.t\_localpart*
- p\_domain *dns.t\_hostname*
- p\_mailbox\_localpart *email.t\_localpart*
- p\_mailbox\_domain *dns.t\_hostname*

**Variables defined for body**

- v\_subservice *commons.t\_key* (default: 'alias')
- v\_num\_total *int*
- v\_num\_domain *int*
- v\_owner *user.t\_user*

**Returns** void

**Execute privilege**

- *userlogin*

```
-- begin userlogin prelude
v_owner := (SELECT t.act_as FROM "user"._get_login() AS t);
-- end userlogin prelude

PERFORM email._address_valid(p_localpart, p_domain);

v_num_total := (SELECT COUNT(*) FROM email._address() AS t WHERE t.owner=v_owner AND
↳ t.subservice=v_subservice);
v_num_domain := (SELECT COUNT(*) FROM email._address() AS t WHERE t.owner=v_owner AND
↳ t.subservice=v_subservice AND t.domain = p_domain);
```

(continues on next page)

(continued from previous page)

```

PERFORM system._contingent_ensure(
    p_owner:=v_owner,
    p_domain:=p_domain,
    p_service:='email',
    p_subservice:=v_subservice,
    p_current_quantity_total:=v_num_total,
    p_current_quantity_domain:=v_num_domain);

PERFORM email._address_valid(p_localpart, p_domain);
LOCK TABLE email.mailbox;

PERFORM commons._raise_inaccessible_or_missing(
EXISTS(
    SELECT TRUE FROM email.mailbox
    WHERE
        domain=p_mailbox_domain AND
        localpart=p_mailbox_localpart AND
        owner=v_owner AND
        backend._active(backend_status)
));

INSERT INTO email.alias
    (service, subservice, localpart, domain, mailbox_localpart, mailbox_domain, ↵
    ↪service_entity_name)
VALUES
    ('email', 'alias', p_localpart, p_domain, p_mailbox_localpart, p_mailbox_domain,
    (SELECT service_entity_name FROM dns.service WHERE service='email' AND domain = p_
    ↪domain));

PERFORM backend._notify_domain('email', 'alias', p_domain);

```

### 6.2.9 email.ins\_list

Creates a mailing list

#### Parameters

- p\_localpart *email.t\_localpart*
- p\_domain *dns.t\_hostname*
- p\_admin *email.t\_address*

#### Variables defined for body

- v\_subservice *commons.t\_key* (default: 'list')
- v\_num\_total *int*
- v\_num\_domain *int*
- v\_owner *user.t\_user*

**Returns** void

#### Execute privilege

- *userlogin*

```

-- begin userlogin prelude
v_owner := (SELECT t.act_as FROM "user"._get_login() AS t);
-- end userlogin prelude

PERFORM email._address_valid(p_localpart, p_domain);

v_num_total := (SELECT COUNT(*) FROM email._address() AS t WHERE t.owner=v_owner AND
↳t.subservice=v_subservice);
v_num_domain := (SELECT COUNT(*) FROM email._address() AS t WHERE t.owner=v_owner AND
↳t.subservice=v_subservice AND t.domain = p_domain);

PERFORM system._contingent_ensure(
    p_owner:=v_owner,
    p_domain:=p_domain,
    p_service:='email',
    p_subservice:=v_subservice,
    p_current_quantity_total:=v_num_total,
    p_current_quantity_domain:=v_num_domain);

INSERT INTO email.list
    (service, subservice, localpart, domain, owner, admin, service_entity_name) VALUES
    ('email', 'list', p_localpart, p_domain, v_owner, p_admin,
    (SELECT service_entity_name FROM dns.service WHERE service='email' AND domain = p_
↳domain));

PERFORM backend._notify_domain('email', 'list', p_domain);

```

### 6.2.10 email.ins\_list\_subscriber

Adds a subscriber to a mailing list

#### Parameters

- p\_address *email.t\_address*
- p\_list\_localpart *email.t\_localpart*
- p\_list\_domain *dns.t\_hostname*

#### Variables defined for body

- v\_owner *user.t\_user*

Returns void

#### Execute privilege

- *userlogin*

```

-- begin userlogin prelude
v_owner := (SELECT t.act_as FROM "user"._get_login() AS t);
-- end userlogin prelude

PERFORM commons._raise_inaccessible_or_missing(
    EXISTS (
        SELECT TRUE FROM email.list
        WHERE

```

(continues on next page)

(continued from previous page)

```

        localpart = p_list_localpart AND
        domain = p_list_domain AND
        owner = v_owner
    )
);

INSERT INTO email.list_subscriber
(address, list_localpart, list_domain)
VALUES
(p_address, p_list_localpart, p_list_domain);

PERFORM backend._notify_domain('email', 'list', p_list_domain);

```

### 6.2.11 email.ins\_mailbox

Creates an email box

#### Parameters

- p\_localpart *email.t\_localpart*
- p\_domain *dns.t\_hostname*
- p\_password *commons.t\_password\_plaintext*

#### Variables defined for body

- v\_subservice *commons.t\_key* (default: 'mailbox')
- v\_num\_total *int*
- v\_num\_domain *int*
- v\_owner *user.t\_user*

Returns void

#### Execute privilege

- *userlogin*

```

-- begin userlogin prelude
v_owner := (SELECT t.act_as FROM "user"._get_login() AS t);
-- end userlogin prelude

PERFORM email._address_valid(p_localpart, p_domain);

v_num_total := (SELECT COUNT(*) FROM email._address() AS t WHERE t.owner=v_owner AND
↳t.subservice=v_subservice);
v_num_domain := (SELECT COUNT(*) FROM email._address() AS t WHERE t.owner=v_owner AND
↳t.subservice=v_subservice AND t.domain = p_domain);

PERFORM system._contingent_ensure(
    p_owner:=v_owner,
    p_domain:=p_domain,
    p_service:='email',
    p_subservice:=v_subservice,
    p_current_quantity_total:=v_num_total,
    p_current_quantity_domain:=v_num_domain);

```

(continues on next page)

(continued from previous page)

```

PERFORM email._address_valid(p_localpart, p_domain);

INSERT INTO email.mailbox
    (service, subservice, localpart, domain, owner, password, service_entity_name)
    ↪ VALUES
    ('email', 'mailbox', p_localpart, p_domain, v_owner, commons._hash_password(p_
    ↪ password),
    (SELECT service_entity_name FROM dns.service WHERE service='email' AND domain = p_
    ↪ domain)
    );

PERFORM backend._notify_domain('email', 'mailbox', p_domain);

```

## 6.2.12 email.ins\_redirection

Creates a redirection

### Parameters

- p\_localpart *email.t\_localpart*
- p\_domain *dns.t\_hostname*
- p\_destination *email.t\_address*

### Variables defined for body

- v\_subservice *commons.t\_key* (default: 'redirection')
- v\_num\_total *int*
- v\_num\_domain *int*
- v\_owner *user.t\_user*

**Returns** void

### Execute privilege

- *userlogin*

```

-- begin userlogin prelude
v_owner := (SELECT t.act_as FROM "user"._get_login() AS t);
-- end userlogin prelude

PERFORM email._address_valid(p_localpart, p_domain);

v_num_total := (SELECT COUNT(*) FROM email._address() AS t WHERE t.owner=v_owner AND
    ↪ t.subservice=v_subservice);
v_num_domain := (SELECT COUNT(*) FROM email._address() AS t WHERE t.owner=v_owner AND
    ↪ t.subservice=v_subservice AND t.domain = p_domain);

PERFORM system._contingent_ensure(
    p_owner:=v_owner,
    p_domain:=p_domain,
    p_service:='email',
    p_subservice:=v_subservice,

```

(continues on next page)

(continued from previous page)

```

    p_current_quantity_total:=v_num_total,
    p_current_quantity_domain:=v_num_domain);

PERFORM email._address_valid(p_localpart, p_domain);

INSERT INTO email.redirection
    (service, subservice, localpart, domain, destination, owner, service_entity_name)
↪VALUES
    ('email', 'redirection', p_localpart, p_domain, p_destination, v_owner,
    (SELECT service_entity_name FROM dns.service WHERE service='email' AND domain = p_
↪domain));

PERFORM backend._notify_domain('email', 'redirection', p_domain);

```

### 6.2.13 email.sel\_alias

Select aliases

**Parameters** *None*

**Variables defined for body**

- v\_owner *user.t\_user*

**Returns** TABLE

**Returned columns**

- localpart *email.t\_localpart*
- domain *dns.t\_hostname*
- mailbox\_localpart *email.t\_localpart*
- mailbox\_domain *dns.t\_hostname*
- backend\_status *backend.t\_status*

**Execute privilege**

- *userlogin*

```

-- begin userlogin prelude
v_owner := (SELECT t.act_as FROM "user"._get_login() AS t);
-- end userlogin prelude

RETURN QUERY
SELECT
    t.localpart,
    t.domain,
    t.mailbox_localpart,
    t.mailbox_domain,
    t.backend_status
FROM email.alias AS t

INNER JOIN email.mailbox AS s
    ON

```

(continues on next page)

(continued from previous page)

```

        t.mailbox_localpart = s.localpart AND
        t.mailbox_domain = s.domain
WHERE s.owner = v_owner

ORDER BY t.backend_status, t.localpart, t.domain;

```

### 6.2.14 email.sel\_list

List all lists

**Parameters** *None*

**Variables defined for body**

- v\_owner *user.t\_user*

**Returns** TABLE

**Returned columns**

- domain *dns.t\_hostname*
- localpart *email.t\_localpart*
- owner *user.t\_user*
- admin *email.t\_address*
- backend\_status *backend.t\_status*
- option *jsonb*
- num\_subscribers *bigint*

**Execute privilege**

- *userlogin*

```

-- begin userlogin prelude
v_owner := (SELECT t.act_as FROM "user"._get_login() AS t);
-- end userlogin prelude

RETURN QUERY
SELECT
    t.domain,
    t.localpart,
    t.owner,
    t.admin,
    t.backend_status,
    t.option,
    (SELECT COUNT(*) FROM email.list_subscriber AS s
     WHERE s.list_localpart=t.localpart AND s.list_domain=t.domain)
FROM
    email.list AS t
WHERE
    t.owner = v_owner
ORDER BY t.backend_status, t.localpart, t.domain
;

```

## 6.2.15 email.sel\_list\_subscriber

a

**Parameters** *None*

**Variables defined for body**

- v\_owner *user:t\_user*

**Returns** TABLE

**Returned columns**

- address *email.t\_address*
- list\_localpart *email.t\_localpart*
- list\_domain *dns.t\_hostname*
- backend\_status *backend.t\_status*

**Execute privilege**

- *userlogin*

```
-- begin userlogin prelude
v_owner := (SELECT t.act_as FROM "user"._get_login() AS t);
-- end userlogin prelude

RETURN QUERY
  SELECT
    t.address,
    t.list_localpart,
    t.list_domain,
    t.backend_status
  FROM email.list_subscriber AS t
  JOIN email.list AS s
  ON
    t.list_localpart = s.localpart AND
    t.list_domain = s.domain
  WHERE
    s.owner = v_owner
  ORDER BY list_localpart, list_domain, backend_status, address
;
```

## 6.2.16 email.sel\_mailbox

List all mailboxes

**Parameters** *None*

**Variables defined for body**

- v\_owner *user:t\_user*

**Returns** TABLE

**Returned columns**

- domain *dns.t\_hostname*



- localpart *email.t\_localpart*
- owner *user.t\_user*
- quota *int*
- backend\_status *backend.t\_status*

**Execute privilege**

- *userlogin*

```
-- begin userlogin prelude
v_owner := (SELECT t.act_as FROM "user"._get_login() AS t);
-- end userlogin prelude

RETURN QUERY
SELECT
  t.domain,
  t.localpart,
  t.owner,
  t.quota,
  t.backend_status
FROM
  email.mailbox AS t
WHERE
  t.owner = v_owner
ORDER BY backend_status, localpart, domain
;
```

**6.2.17 email.sel\_redirection**

Lists all redirections

**Parameters** *None*

**Variables defined for body**

- v\_owner *user:t\_user*

**Returns** TABLE

**Returned columns**

- domain *dns.t\_hostname*
- localpart *email.t\_localpart*
- destination *email.t\_address*
- backend\_status *backend.t\_status*

**Execute privilege**

- *userlogin*

```
-- begin userlogin prelude
v_owner := (SELECT t.act_as FROM "user"._get_login() AS t);
-- end userlogin prelude
```

(continues on next page)

(continued from previous page)

```

RETURN QUERY
SELECT
  t.domain,
  t.localpart,
  t.destination,
  t.backend_status
FROM
  email.redirection AS t
WHERE
  t.owner = v_owner
ORDER BY t.backend_status, t.localpart, t.domain;

```

## 6.2.18 email.srv\_alias

Lists all email aliases

### Parameters

- `p_include_inactive` *boolean*

Returns TABLE

### Returned columns

- `localpart` *email.t\_localpart*
- `domain` *dns.t\_hostname*
- `mailbox_localpart` *email.t\_localpart*
- `mailbox_domain` *dns.t\_hostname*
- `backend_status` *backend.t\_status*

### Execute privilege

- *backend*

```

PERFORM backend._get_login();

RETURN QUERY
WITH
  -- DELETE
  d AS (
    DELETE FROM email.alias AS t
    WHERE
      backend._deleted(t.backend_status) AND
      backend._machine_priviledged(t.service, t.domain)
  ),
  -- UPDATE
  s AS (
    UPDATE email.alias AS t
    SET backend_status = NULL
    WHERE
      backend._machine_priviledged(t.service, t.domain) AND
      backend._active(t.backend_status)
  )

```

(continues on next page)

(continued from previous page)

```

)

-- SELECT
SELECT
    t.localpart,
    t.domain,
    t.mailbox_localpart,
    t.mailbox_domain,
    t.backend_status
FROM email.alias AS t

WHERE
    backend._machine_priviledged(t.service, t.domain) AND
    (backend._active(t.backend_status) OR p_include_inactive);

```

## 6.2.19 email.srv\_list

Lists all mailinglists

### Parameters

- `p_include_inactive` *boolean*

Returns TABLE

### Returned columns

- `localpart` *email.t\_localpart*
- `domain` *dns.t\_hostname*
- `admin` *email.t\_address*
- `option` *jsonb*
- `backend_status` *backend.t\_status*

### Execute privilege

- *backend*

```

PERFORM backend._get_login();

RETURN QUERY
WITH
    -- DELETE
    d AS (
        DELETE FROM email.list AS t
        WHERE
            backend._deleted(t.backend_status) AND
            backend._machine_priviledged(t.service, t.domain)
    ),
    -- UPDATE
    s AS (
        UPDATE email.list AS t
        SET backend_status = NULL
    )

```

(continues on next page)

(continued from previous page)

```

WHERE
    backend._machine_priviledged(t.service, t.domain) AND
    backend._active(t.backend_status)
)

-- SELECT
SELECT
    t.localpart,
    t.domain,
    t.admin,
    t.option,
    t.backend_status
FROM email.list AS t

WHERE
    backend._machine_priviledged(t.service, t.domain) AND
    (backend._active(t.backend_status) OR p_include_inactive);

```

## 6.2.20 email.srv\_list\_subscriber

Lists all mailinglist subscribers

### Parameters

- `p_include_inactive` *boolean*

Returns TABLE

### Returned columns

- `localpart` *email.t\_localpart*
- `domain` *dns.t\_hostname*
- `address` *email.t\_address*
- `backend_status` *backend.t\_status*

### Execute privilege

- *backend*

```

PERFORM backend._get_login();

RETURN QUERY
WITH
    -- DELETE
    d AS (
        DELETE FROM email.list_subscriber AS t
        USING email.list AS l
        WHERE
            t.list_domain = l.domain AND
            t.list_localpart = l.localpart AND

            backend._deleted(t.backend_status) AND
            backend._machine_priviledged(l.service, l.domain)
    )

```

(continues on next page)

(continued from previous page)

```

),

-- UPDATE
s AS (
    UPDATE email.list_subscriber AS t
        SET backend_status = NULL
    FROM email.list AS l
    WHERE
        t.list_domain = l.domain AND
        t.list_localpart = l.localpart AND

        backend._machine_priviledged(l.service, l.domain) AND
        backend._active(t.backend_status)
)

-- SELECT
SELECT
    t.list_localpart,
    t.list_domain,
    t.address,
    t.backend_status
FROM email.list_subscriber AS t

JOIN email.list AS l ON
    t.list_domain = l.domain AND
    t.list_localpart = l.localpart

WHERE
    backend._machine_priviledged(l.service, l.domain) AND
    (backend._active(t.backend_status) OR p_include_inactive);

```

### 6.2.21 email.srv\_mailbox

Lists all mailboxes

#### Parameters

- `p_include_inactive` *boolean*

**Returns** TABLE

#### Returned columns

- `localpart` *email.t\_localpart*
- `domain` *dns.t\_hostname*
- `password` *commons.t\_password*
- `uid` *integer*
- `quota` *integer*
- `option` *jsonb*
- `backend_status` *backend.t\_status*

#### Execute privilege

- *backend*

```
PERFORM backend._get_login();

RETURN QUERY
WITH
    -- DELETE
    d AS (
        DELETE FROM email.mailbox AS t
        WHERE
            backend._deleted(t.backend_status) AND
            backend._machine_priviledged(t.service, t.domain)
    ),
    -- UPDATE
    s AS (
        UPDATE email.mailbox AS t
        SET backend_status = NULL
        WHERE
            backend._machine_priviledged(t.service, t.domain) AND
            backend._active(t.backend_status)
    )

    -- SELECT
    SELECT
        t.localpart,
        t.domain,
        t.password,
        t.uid,
        t.quota,
        t.option,
        t.backend_status
    FROM email.mailbox AS t

    WHERE
        backend._machine_priviledged(t.service, t.domain) AND
        (backend._active(t.backend_status) OR p_include_inactive);
```

## 6.2.22 email.srv\_redirection

Lists all mailinglists

### Parameters

- `p_include_inactive` *boolean*

Returns TABLE

### Returned columns

- `localpart` *email.t\_localpart*
- `domain` *dns.t\_hostname*
- `destination` *email.t\_address*
- `backend_status` *backend.t\_status*

Execute privilege

- *backend*

```

PERFORM backend._get_login();

RETURN QUERY
WITH

    -- DELETE
    d AS (
        DELETE FROM email.redirection AS t
        WHERE
            backend._deleted(t.backend_status) AND
            backend._machine_priviledged(t.service, t.domain)
    ),

    -- UPDATE
    s AS (
        UPDATE email.redirection AS t
        SET backend_status = NULL
        WHERE
            backend._machine_priviledged(t.service, t.domain) AND
            backend._active(t.backend_status)
    )

    -- SELECT
    SELECT
        t.localpart,
        t.domain,
        t.destination,
        t.backend_status
    FROM email.redirection AS t

    WHERE
        backend._machine_priviledged(t.service, t.domain) AND
        (backend._active(t.backend_status) OR p_include_inactive);

```

### 6.2.23 email.upd\_list

Change list admin

#### Parameters

- p\_localpart *email.t\_localpart*
- p\_domain *dns.t\_hostname*
- p\_admin *email.t\_address*

#### Variables defined for body

- v\_owner *user.t\_user*

Returns void

#### Execute privilege

- *userlogin*

```
-- begin userlogin prelude
v_owner := (SELECT t.act_as FROM "user"._get_login() AS t);
-- end userlogin prelude

UPDATE email.list
  SET
    admin = p_admin,
    backend_status = 'upd'
WHERE
  localpart = p_localpart AND
  domain = p_domain AND
  owner = v_owner AND
  backend._active(backend_status);

PERFORM backend._conditional_notify(FOUND, 'email', 'list', p_domain);
```

## 6.2.24 email.upd\_mailbox

Change mailbox password

### Parameters

- p\_localpart *email.t\_localpart*
- p\_domain *dns.t\_hostname*
- p\_password *commons.t\_password\_plaintext*

### Variables defined for body

- v\_owner *user.t\_user*

**Returns** void

### Execute privilege

- *userlogin*

```
-- begin userlogin prelude
v_owner := (SELECT t.act_as FROM "user"._get_login() AS t);
-- end userlogin prelude

UPDATE email.mailbox
  SET
    password = commons._hash_password(p_password),
    backend_status = 'upd'
WHERE
  localpart = p_localpart AND
  domain = p_domain AND
  owner = v_owner AND
  backend._active(backend_status);

PERFORM backend._conditional_notify(FOUND, 'email', 'mailbox', p_domain);
```



## 6.3 Domains

### 6.3.1 email.t\_localpart

Local part of an email address, the thing in front of the @

#### Checks

- **valid\_characters** Only allow lower-case addresses

```
VALUE ~ '^[a-z0-9.\-]+$'
```

- **no\_starting\_dot** b

```
left(VALUE, 1) <> '.'
```

- **no\_ending\_dot** c

```
right(VALUE, 1) <> '.'
```

### 6.3.2 email.t\_address

Email address

---

**Todo:** validity checks

---



Jabber (XMPP)

**This module sends the following signals:**

- jabber/account

### Schema Contents

- *Tables*
  - *jabber.account*
- *Functions*
  - *jabber.del\_account*
  - *jabber.ins\_account*
  - *jabber.sel\_account*
  - *jabber.srv\_account*
  - *jabber.upd\_account*

## 7.1 Tables

### 7.1.1 jabber.account

Jabber accounts

**Primary key**

- node
- domain

### Foreign keys

- reference dns (service)

#### Local Columns

- domain
- service
- service\_entity\_name

#### Referenced Columns

- *dns.service.domain*
- *dns.service.service*
- *dns.service.service\_entity\_name*

- Reference subservice entity

#### Local Columns

- service\_entity\_name
- service
- subservice

#### Referenced Columns

- *system.subservice\_entity.service\_entity\_name*
- *system.subservice\_entity.service*
- *system.subservice\_entity.subservice*

### Columns

- **domain** *dns.t\_hostname* Domain name
- **service** *commons.t\_key* Service
- **service\_entity\_name** *dns.t\_hostname* ent. name
- **subservice** *commons.t\_key* Subservice (e.g. account, alias)
- **owner** *user.t\_user* Owner

References *user.user.owner*

On Update: CASCADE

- **backend\_status** *NULL* | *backend.t\_status* Status of database entry in backend. *NULL*: nothing pending, 'ins': entry not present on backend client, 'upd': update pending on backend client, 'del': deletion pending on backend client.

#### Default

`'ins'`

- **node** *email.t\_localpart* part in front of the @ in account name
- **password** *commons.t\_password* Unix shadow crypt format

## 7.2 Functions

### 7.2.1 jabber.del\_account

Delete jabber account

#### Parameters

- p\_node *email.t\_localpart*
- p\_domain *dns.t\_hostname*

#### Variables defined for body

- v\_owner *user.t\_user*

**Returns** void

#### Execute privilege

- *userlogin*

```
-- begin userlogin prelude
v_owner := (SELECT t.act_as FROM "user"._get_login() AS t);
-- end userlogin prelude

UPDATE jabber.account
SET backend_status = 'del'
WHERE
  node = p_node AND
  domain = p_domain AND
  owner = v_owner;

PERFORM backend._conditional_notify(FOUND, 'jabber', 'account', p_domain);
```

### 7.2.2 jabber.ins\_account

Insert jabber account

#### Parameters

- p\_node *email.t\_localpart*
- p\_domain *dns.t\_hostname*
- p\_password *commons.t\_password\_plaintext*

#### Variables defined for body

- v\_num\_total *integer*
- v\_num\_domain *integer*
- v\_owner *user.t\_user*

**Returns** void

#### Execute privilege

- *userlogin*

```

-- begin userlogin prelude
v_owner := (SELECT t.act_as FROM "user"._get_login() AS t);
-- end userlogin prelude

v_num_total := (SELECT COUNT(*) FROM jabber.account AS t WHERE t.owner=v_owner);
v_num_domain := (SELECT COUNT(*) FROM jabber.account AS t WHERE t.owner=v_owner AND t.
↪domain = p_domain);

PERFORM system._contingent_ensure(
    p_owner:=v_owner,
    p_domain:=p_domain,
    p_service:='jabber',
    p_subservice:='account',
    p_current_quantity_total:=v_num_total,
    p_current_quantity_domain:=v_num_domain);

INSERT INTO jabber.account
    (service, subservice, node, domain, owner, password, service_entity_name) VALUES
    ('jabber', 'account', p_node, p_domain, v_owner, commons._hash_password(p_
↪password),
    (SELECT service_entity_name FROM dns.service WHERE service='jabber' AND domain =
↪p_domain));

PERFORM backend._notify_domain('jabber', 'account', p_domain);

```

### 7.2.3 jabber.sel\_account

Select jabber accounts

**Parameters** *None*

**Variables defined for body**

- v\_owner *user.t\_user*

**Returns** TABLE

**Returned columns**

- node *email.t\_localpart*
- domain *dns.t\_hostname*
- backend\_status *backend.t\_status*

**Execute privilege**

- *userlogin*

```

-- begin userlogin prelude
v_owner := (SELECT t.act_as FROM "user"._get_login() AS t);
-- end userlogin prelude

RETURN QUERY
    SELECT
        t.node,
        t.domain,

```

(continues on next page)

(continued from previous page)

```

        t.backend_status
FROM jabber.account AS t
WHERE
    t.owner = v_owner
ORDER BY t.backend_status, t.node, t.domain;
```

## 7.2.4 jabber.srv\_account

Lists all jabber accounts

### Parameters

- `p_include_inactive` *boolean*

Returns TABLE

### Returned columns

- node *email.t\_localpart*
- domain *dns.t\_hostname*
- password *commons.t\_password*
- backend\_status *backend.t\_status*

### Execute privilege

- *backend*

```

PERFORM backend._get_login();

RETURN QUERY
WITH

    -- DELETE
    d AS (
        DELETE FROM jabber.account AS t
        WHERE
            backend._deleted(t.backend_status) AND
            backend._machine_priviledged(t.service, t.domain)
    ),

    -- UPDATE
    s AS (
        UPDATE jabber.account AS t
        SET backend_status = NULL
        WHERE
            backend._machine_priviledged(t.service, t.domain) AND
            backend._active(t.backend_status)
    )

    -- SELECT
SELECT
    t.node,
    t.domain,
    t.password,
```

(continues on next page)

(continued from previous page)

```
t.backend_status
FROM jabber.account AS t

WHERE
    backend._machine_privileged(t.service, t.domain) AND
    (backend._active(t.backend_status) OR p_include_inactive);
```

## 7.2.5 jabber.upd\_account

Change jabber account password

### Parameters

- p\_node *email.t\_localpart*
- p\_domain *dns.t\_hostname*
- p\_password *commons.t\_password\_plaintext*

### Variables defined for body

- v\_owner *user.t\_user*

Returns void

### Execute privilege

- *userlogin*

```
-- begin userlogin prelude
v_owner := (SELECT t.act_as FROM "user"._get_login() AS t);
-- end userlogin prelude

UPDATE jabber.account
    SET
        password = commons._hash_password(p_password)
WHERE
    node = p_node AND
    domain = p_domain AND
    owner = v_owner;

PERFORM backend._conditional_notify(FOUND, 'jabber', 'account', p_domain);
```



---

### server\_access

---

#### Server Access

Explicit passwd entries for shell accounts and sftp.

**This module sends the following signals:**

- server\_access/sftp
- server\_access/ssh

#### Schema Contents

- *Tables*
  - `server_access.user`
- *Functions*
  - `server_access.del_user`
  - `server_access.ins_user`
  - `server_access.sel_user`
  - `server_access.srv_user`
  - `server_access.upd_user`
- *Domains*
  - `server_access.t_user`

## 8.1 Tables

### 8.1.1 `server_access.user`

unix user

#### Primary key

- `user`

#### Foreign keys

- Reference service entity

##### Local Columns

- `service_entity_name`
- `service`

##### Referenced Columns

- `system.service_entity.service_entity_name`
- `system.service_entity.service`

- Reference subservice entity

##### Local Columns

- `service_entity_name`
- `service`
- `subservice`

##### Referenced Columns

- `system.subservice_entity.service_entity_name`
- `system.subservice_entity.service`
- `system.subservice_entity.subservice`

#### Columns

- **`service_entity_name`** *`dns.t.hostname`* Service entity name
- **`service`** *`commons.t.key`* Service (e.g. email, jabber)
- **`subservice`** *`commons.t.key`* Subservice (e.g. account, alias)
- **`backend_status`** *`NULL`* | *`backend.t.status`* Status of database entry in backend. `NULL`: nothing pending, `'ins'`: entry not present on backend client, `'upd'`: update pending on backend client, `'del'`: deletion pending on backend client.

#### Default

'ins'

- **`owner`** *`user.t.user`* Owner  
References *`user.user.owner`*  
On Update: CASCADE
- **`uid`** *`integer`* Unix user identifier

**Default**

```
nextval('commons.uid')
```

- **user** *server\_access.t\_user* User
- **password** *NULL* | *commons.t\_password* Unix shadow crypt format

## 8.2 Functions

### 8.2.1 server\_access.del\_user

delete

**Parameters**

- p\_user *server\_access.t\_user*
- p\_service\_entity\_name *dns.t\_hostname*

**Variables defined for body**

- v\_subservice *commons.t\_key*
- v\_owner *user.t\_user*

**Returns** void

**Execute privilege**

- *userlogin*

```
-- begin userlogin prelude
v_owner := (SELECT t.act_as FROM "user"._get_login() AS t);
-- end userlogin prelude

BEGIN
    -- perform DELETE to trigger potential foreign key errors
    DELETE FROM server_access.user
    WHERE
        "user" = p_user AND
        service_entity_name = p_service_entity_name AND
        owner = v_owner;

    -- if not failed yet, emulate rollback of DELETE
    RAISE transaction_rollback;
EXCEPTION
    WHEN transaction_rollback THEN
        UPDATE server_access.user
            SET backend_status = 'del'
        WHERE
            "user" = p_user AND
            service_entity_name = p_service_entity_name AND
            owner = v_owner
        RETURNING subservice INTO v_subservice;

    PERFORM backend._conditional_notify_service_entity_name(
        FOUND, p_service_entity_name, 'server_access', v_subservice
```

(continues on next page)

(continued from previous page)

```
);  
END;
```

## 8.2.2 `server_access.ins_user`

ins user

### Parameters

- `p_user` *server\_access.t\_user*
- `p_service_entity_name` *dns.t\_hostname*
- `p_subservice` *commons.t\_key*
- `p_password` *commons.t\_password\_plaintext*

### Variables defined for body

- `v_password` *commons.t\_password*
- `v_owner` *user.t\_user*

Returns void

### Execute privilege

- *userlogin*

```
-- begin userlogin prelude  
v_owner := (SELECT t.act_as FROM "user"._get_login() AS t);  
-- end userlogin prelude  
  
IF p_password IS NULL THEN  
    v_password := NULL;  
ELSE  
    v_password := commons._hash_password(p_password);  
END IF;  
  
INSERT INTO server_access.user  
    (service, subservice, service_entity_name, "user", password, owner)  
VALUES  
    ('server_access', p_subservice, p_service_entity_name, p_user, v_password, v_  
↪owner);  
  
PERFORM backend._notify_service_entity_name(p_service_entity_name, 'server_access', p_  
↪subservice);
```

## 8.2.3 `server_access.sel_user`

sel user

Parameters *None*

### Variables defined for body

- `v_owner` *user.t\_user*

**Returns TABLE****Returned columns**

- user *server\_access.t\_user*
- password\_login *boolean*
- service *commons.t\_key*
- subservice *commons.t\_key*
- service\_entity\_name *dns.t\_hostname*
- backend\_status *backend.t\_status*

**Execute privilege**

- *userlogin*

```
-- begin userlogin prelude
v_owner := (SELECT t.act_as FROM "user"._get_login() AS t);
-- end userlogin prelude

RETURN QUERY
  SELECT
    t.user,
    t.password IS NOT NULL,
    t.service,
    t.subservice,
    t.service_entity_name,
    t.backend_status
  FROM
    server_access.user AS t
  WHERE
    owner = v_owner
  ORDER BY backend_status, "user"
;
```

**8.2.4 server\_access.srv\_user**

backend server\_access.user

**Parameters**

- p\_include\_inactive *boolean*

**Returns TABLE****Returned columns**

- user *server\_access.t\_user*
- password *commons.t\_password*
- service *commons.t\_key*
- subservice *commons.t\_key*
- service\_entity\_name *dns.t\_hostname*
- backend\_status *backend.t\_status*

- uid *int*

### Execute privilege

- *backend*

```
PERFORM backend._get_login();

RETURN QUERY
WITH
    -- DELETE
    d AS (
        DELETE FROM server_access.user AS t
        WHERE
            backend._deleted(t.backend_status) AND
            backend._machine_priviledged_service(t.service, t.service_entity_name)
    ),
    -- UPDATE
    s AS (
        UPDATE server_access.user AS t
        SET backend_status = NULL
        WHERE
            backend._machine_priviledged_service(t.service, t.service_entity_name) AND
            backend._active(t.backend_status)
    )
    -- SELECT
    SELECT
        t.user,
        t.password,
        t.service,
        t.subservice,
        t.service_entity_name,
        t.backend_status,
        t.uid
    FROM server_access.user AS t
    WHERE
        backend._machine_priviledged_service(t.service, t.service_entity_name) AND
        (backend._active(t.backend_status) OR p_include_inactive);
```

## 8.2.5 server\_access.upd\_user

passwd user

### Parameters

- p\_user *server\_access.t\_user*
- p\_service\_entity\_name *dns.t\_hostname*
- p\_password *commons.t\_password\_plaintext*

### Variables defined for body

- v\_password *commons.t\_password* (default: NULL)

- v\_subservice *commons.t\_key*
- v\_owner *user.t\_user*

**Returns** void

**Execute privilege**

- *userlogin*

```
-- begin userlogin prelude
v_owner := (SELECT t.act_as FROM "user"._get_login() AS t);
-- end userlogin prelude

IF p_password IS NOT NULL THEN
    v_password := commons._hash_password(p_password);
END IF;

UPDATE server_access.user
SET
    password = v_password,
    backend_status = 'upd'
WHERE
    "user" = p_user AND
    service_entity_name = p_service_entity_name AND
    owner = v_owner
RETURNING subservice INTO v_subservice;

PERFORM backend._conditional_notify_service_entity_name(
    FOUND, p_service_entity_name, 'server_access', v_subservice
);
```

## 8.3 Domains

### 8.3.1 server\_access.t\_user

Unix user. This type only allows a subset of those names allowed by POSIX.

**Checks**

- **valid\_characters** Only allow lower-case characters.

```
VALUE ~ '^[a-z0-9\_]+$'
```

- **no\_repeated\_hyphens** Reserve double hyphens as a separator for system generated users.

```
NOT (VALUE LIKE '%--%')
```

- **no\_starting\_hyphen** No hyphens at the beginning: [http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1\\_chap03.html#tag\\_03\\_431](http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap03.html#tag_03_431)

```
left (VALUE, 1) <> '-'
```





#### Websites

**This module sends the following signals:**

- web/alias
- web/site

#### Schema Contents

- *Tables*
  - `web.alias`
  - `web.site`
- *Functions*
  - `web.del_alias`
  - `web.del_site`
  - `web.ins_alias`
  - `web.ins_site`
  - `web.sel_alias`
  - `web.sel_site`
  - `web.srv_alias`
  - `web.srv_site`

## 9.1 Tables

### 9.1.1 `web.alias`

Aliases

#### Primary key

- domain
- site\_port

#### Foreign keys

- reference dns (service)

##### Local Columns

- domain
- service
- service\_entity\_name

##### Referenced Columns

- *dns.service.domain*
- *dns.service.service*
- *dns.service.service\_entity\_name*

- Reference subservice entity

##### Local Columns

- service\_entity\_name
- service
- subservice

##### Referenced Columns

- *system.subservice\_entity.service\_entity\_name*
- *system.subservice\_entity.service*
- *system.subservice\_entity.subservice*

- site

##### Local Columns

- site
- service\_entity\_name
- site\_port

##### Referenced Columns

- *web.site.domain*
- *web.site.service\_entity\_name*
- *web.site.port*

- dns

**Local Columns**

- domain
- service
- service\_entity\_name

**Referenced Columns**

- *dns.service.domain*
- *dns.service.service*
- *dns.service.service\_entity\_name*

**Columns**

- **domain** *dns.t\_hostname* Domain name
- **service** *commons.t\_key* Service
- **service\_entity\_name** *dns.t\_hostname* ent. name
- **subservice** *commons.t\_key* Subservice (e.g. account, alias)
- **backend\_status** *NULL* | *backend.t\_status* Status of database entry in backend. *NULL*: nothing pending, 'ins': entry not present on backend client, 'upd': update pending on backend client, 'del': deletion pending on backend client.

**Default**

'ins'

- **site** *dns.t\_hostname* Site
- **site\_port** *commons.t\_port* port

**Default**

80

**9.1.2 web.site**

Website

**Primary key**

- domain
- port

**Foreign keys**

- reference dns (service)

**Local Columns**

- domain
- service
- service\_entity\_name

**Referenced Columns**

- *dns.service.domain*

- *dns.service.service*
- *dns.service.service\_entity\_name*

- Reference subservice entity

#### Local Columns

- *service\_entity\_name*
- *service*
- *subservice*

#### Referenced Columns

- *system.subservice\_entity.service\_entity\_name*
- *system.subservice\_entity.service*
- *system.subservice\_entity.subservice*

- *server\_access*

#### Local Columns

- *user*
- *service\_entity\_name*
- *owner*

#### Referenced Columns

- *server\_access.user.user*
- *server\_access.user.service\_entity\_name*
- *server\_access.user.owner*

### Columns

- **domain** *dns.t\_hostname* Domain name
- **service** *commons.t\_key* Service
- **service\_entity\_name** *dns.t\_hostname* ent. name
- **subservice** *commons.t\_key* Subservice (e.g. account, alias)
- **backend\_status** *NULL | backend.t\_status* Status of database entry in backend. *NULL*: nothing pending, 'ins': entry not present on backend client, 'upd': update pending on backend client, 'del': deletion pending on backend client.

#### Default

```
'ins'
```

- **option** *jsonb* Free options in JSON format

#### Default

```
' {} '
```

- **owner** *user.t\_user* Owner

References *user.user.owner*

On Update: CASCADE

- **port** *commons.t\_port* Port
- **user** *server\_access.t\_user* Server account under which the htdocs reside
- **https** *bool* HTTPS

## 9.2 Functions

### 9.2.1 web.del\_alias

del

#### Parameters

- p\_domain *dns.t\_hostname*
- p\_site\_port *commons.t\_port*

#### Variables defined for body

- v\_owner *user.t\_user*

**Returns** void

#### Execute privilege

- *userlogin*

```
-- begin userlogin prelude
v_owner := (SELECT t.act_as FROM "user"._get_login() AS t);
-- end userlogin prelude

UPDATE web.alias AS t
    SET backend_status = 'del'
FROM web.site AS s, server_access.user AS u
WHERE
    -- JOIN web.site
    s.domain = t.site AND

    -- JOIN server_access.user
    u.service_entity_name = t.service_entity_name AND
    u.user = s.user AND

    u.owner = v_owner AND
    t.domain = p_domain AND
    t.site_port = p_site_port;

PERFORM backend._conditional_notify(FOUND, 'web', 'alias', p_domain);
```

### 9.2.2 web.del\_site

del

#### Parameters

- p\_domain *dns.t\_hostname*
- p\_port *commons.t\_port*

**Variables defined for body**

- v\_owner *user:t\_user*

**Returns** void**Execute privilege**

- *userlogin*

```
-- begin userlogin prelude
v_owner := (SELECT t.act_as FROM "user"._get_login() AS t);
-- end userlogin prelude

UPDATE web.site AS t
    SET backend_status = 'del'
FROM server_access.user AS s
WHERE
    -- JOIN server_access.user
    s.user = t.user AND
    s.service_entity_name = t.service_entity_name AND

    t.domain = p_domain AND
    t.port = p_port AND
    s.owner = v_owner;

PERFORM backend._conditional_notify(FOUND, 'web', 'site', p_domain);
```

## 9.2.3 web.ins\_alias

Insert alias

**Parameters**

- p\_domain *dns.t\_hostname*
- p\_site *dns.t\_hostname*
- p\_site\_port *commons.t\_port*

**Variables defined for body**

- v\_owner *user:t\_user*

**Returns** void**Execute privilege**

- *userlogin*

```
-- begin userlogin prelude
v_owner := (SELECT t.act_as FROM "user"._get_login() AS t);
-- end userlogin prelude

PERFORM commons._raise_inaccessible_or_missing(
    EXISTS (
        SELECT TRUE FROM web.site AS t
        JOIN server_access.user AS s
            USING ("user", service_entity_name)
```

(continues on next page)

(continued from previous page)

```

        WHERE
            t.domain = p_site AND
            t.port = p_site_port AND
            s.owner = v_owner
    )
);

INSERT INTO web.alias
    (domain, service, subservice, site, site_port, service_entity_name)
VALUES
    (
        p_domain,
        'web',
        'alias',
        p_site,
        p_site_port,
        (SELECT service_entity_name FROM web.site WHERE domain = p_site AND port = p_
↪site_port)
    );

PERFORM backend._notify_domain('web', 'alias', p_domain);

```

## 9.2.4 web.ins\_site

Insert site

---

**Todo:** check owner and contingent

---

### Parameters

- p\_domain *dns.t\_hostname*
- p\_port *commons.t\_port*
- p\_https *bool*
- p\_user *server\_access.t\_user*
- p\_service\_entity\_name *dns.t\_hostname*

### Variables defined for body

- v\_owner *user.t\_user*

**Returns** void

### Execute privilege

- *userlogin*

```

-- begin userlogin prelude
v_owner := (SELECT t.act_as FROM "user"._get_login() AS t);
-- end userlogin prelude

PERFORM system._contingent_ensure(
    p_owner:=v_owner,

```

(continues on next page)

(continued from previous page)

```

p_domain:=p_domain,
p_service:='web',
p_subservice:='site',
p_current_quantity_total:=
  (SELECT COUNT(*) FROM web.site WHERE owner=v_owner)::int,
p_current_quantity_domain:=
  (SELECT COUNT(*) FROM web.site WHERE owner=v_owner AND domain = p_domain)::int
);

INSERT INTO web.site
  (domain, service, subservice, port, https, "user", service_entity_name, owner)
VALUES
  (p_domain, 'web', 'site', p_port, p_https, p_user, p_service_entity_name, v_
↪owner);

PERFORM backend._notify_domain('web', 'site', p_domain);

```

### 9.2.5 web.sel\_alias

Select alias

Parameters *None*

Variables defined for body

- v\_owner *user.t\_user*

Returns TABLE

Returned columns

- domain *dns.t\_hostname*
- site *dns.t\_hostname*
- site\_port *commons.t\_port*
- backend\_status *backend.t\_status*

Execute privilege

- *userlogin*

```

-- begin userlogin prelude
v_owner := (SELECT t.act_as FROM "user"._get_login() AS t);
-- end userlogin prelude

RETURN QUERY
  SELECT
    t.domain,
    t.site,
    t.site_port,
    t.backend_status
  FROM web.alias AS t

  JOIN web.site AS u
    ON
      u.domain = t.site AND

```

(continues on next page)



(continued from previous page)

```

        u.port = t.site_port

JOIN server_access.user AS s
  ON
    u.user = s.user AND
    s.service_entity_name = t.service_entity_name

WHERE s.owner = v_owner
ORDER BY t.backend_status, t.domain;

```

### 9.2.6 web.sel\_site

Owner defined via server\_access

**Parameters** *None*

**Variables defined for body**

- v\_owner *user:t\_user*

**Returns** TABLE

**Returned columns**

- service *commons.t\_key*
- subservice *commons.t\_key*
- domain *dns.t\_hostname*
- port *commons.t\_port*
- user *server\_access.t\_user*
- service\_entity\_name *dns.t\_hostname*
- https *bool*
- backend\_status *backend.t\_status*
- option *jsonb*

**Execute privilege**

- *userlogin*

```

-- begin userlogin prelude
v_owner := (SELECT t.act_as FROM "user"._get_login() AS t);
-- end userlogin prelude

```

**RETURN QUERY**

```

SELECT
    t.service,
    t.subservice,
    t.domain,
    t.port,
    t.user,
    t.service_entity_name,
    t.https,
    t.backend_status,

```

(continues on next page)

(continued from previous page)

```

    t.option
FROM web.site AS t
JOIN server_access.user AS s
    USING ("user", service_entity_name)
WHERE
    s.owner = v_owner
ORDER BY t.backend_status, t.domain, t.port;

```

## 9.2.7 web.srv\_alias

backend web.alias

### Parameters

- p\_include\_inactive *boolean*

Returns TABLE

### Returned columns

- domain *dns.t\_hostname*
- site *dns.t\_hostname*
- site\_port *commons.t\_port*
- backend\_status *backend.t\_status*

### Execute privilege

- *backend*

```

PERFORM backend._get_login();

RETURN QUERY
WITH
    -- DELETE
    d AS (
        DELETE FROM web.alias AS t
        WHERE
            backend._deleted(t.backend_status) AND
            backend._machine_priviledged(t.service, t.domain)
    ),
    -- UPDATE
    s AS (
        UPDATE web.alias AS t
        SET backend_status = NULL
        WHERE
            backend._machine_priviledged(t.service, t.domain) AND
            backend._active(t.backend_status)
    )

    -- SELECT
    SELECT
        t.domain,

```

(continues on next page)

(continued from previous page)

```

    t.site,
    t.site_port,
    t.backend_status
FROM web.alias AS t

WHERE
    backend._machine_priviledged(t.service, t.domain) AND
    (backend._active(t.backend_status) OR p_include_inactive);

```

### 9.2.8 web.srv\_site

backend web.site

#### Parameters

- p\_include\_inactive *boolean*

Returns TABLE

#### Returned columns

- domain *dns.t\_hostname*
- port *commons.t\_port*
- user *server\_access.t\_user*
- service\_entity\_name *dns.t\_hostname*
- https *bool*
- subservice *commons.t\_key*
- option *jsonb*
- backend\_status *backend.t\_status*

#### Execute privilege

- *backend*

```

PERFORM backend._get_login();

RETURN QUERY
WITH

    -- DELETE
    d AS (
        DELETE FROM web.site AS t
        WHERE
            backend._deleted(t.backend_status) AND
            backend._machine_priviledged(t.service, t.domain)
    ),

    -- UPDATE
    s AS (
        UPDATE web.site AS t
        SET backend_status = NULL
        WHERE

```

(continues on next page)

(continued from previous page)

```
        backend._machine_priviledged(t.service, t.domain) AND
        backend._active(t.backend_status)
    )

-- SELECT
SELECT
    t.domain,
    t.port,
    t.user,
    t.service_entity_name,
    t.https,
    t.subservice,
    t.option,
    t.backend_status
FROM web.site AS t

WHERE
    backend._machine_priviledged(t.service, t.domain) AND
    (backend._active(t.backend_status) OR p_include_inactive)

ORDER BY t.domain, t.port;
```

### Carnivora Backend

The backend module provides everything required for the backend API. The backend API delivers content required for building configs etc. to clients, called machines.

#### Schema Contents

- *Tables*
  - *backend.auth*
  - *backend.machine*
- *Functions*
  - *backend.\_active*
  - *backend.\_conditional\_notify*
  - *backend.\_conditional\_notify\_service\_entity\_name*
  - *backend.\_deleted*
  - *backend.\_get\_login*
  - *backend.\_login\_machine*
  - *backend.\_machine\_priviledged*
  - *backend.\_machine\_priviledged\_service*
  - *backend.\_notify*
  - *backend.\_notify\_domain*
  - *backend.\_notify\_service\_entity\_name*
- *Domains*

- *backend.t\_status*
- *Roles*
  - *backend*

## 10.1 Tables

### 10.1.1 backend.auth

Grants rights to backend API clients based on SQL roles.

#### Primary key

- role

#### Columns

- **option** *jsonb* Free options in JSON format

#### Default

```
'{}'
```

- **role** *commons.t\_key* Grantee for right to access the backend data for the defined machine. A role is basically a user or a user group on the SQL server.
- **machine** *dns.t\_hostname* Machine for which the rights are granted.

References *backend.machine.name*

On Delete: CASCADE

### 10.1.2 backend.machine

Physical or virtual machines that hosts services.

#### Primary key

- name

#### Columns

- **option** *jsonb* Free options in JSON format

#### Default

```
'{}'
```

- **name** *dns.t\_hostname* Machine name

## 10.2 Functions

### 10.2.1 backend.\_active

Is not 'del'

**Parameters**

- backend\_status *backend.t\_status*

**Returns** boolean

```
RETURN backend_status IS NULL OR (backend_status <> 'del' AND backend_status <> 'old
↳');
```

**10.2.2 backend.\_conditional\_notify**

Notifies if first argument is true. Throws inaccessible otherwise.

**Parameters**

- p\_condition *boolean*
- p\_service *commons.t\_key*
- p\_subservice *commons.t\_key*
- p\_domain *dns.t\_hostname*

**Returns** void

```
IF p_condition THEN
    PERFORM backend._notify_domain(p_service, p_subservice, p_domain);
ELSE
    PERFORM commons._raise_inaccessible_or_missing();
END IF;
```

**10.2.3 backend.\_conditional\_notify\_service\_entity\_name**

Notifies if first argument is true. Throws inaccessible otherwise.

**Parameters**

- p\_condition *boolean*
- p\_service\_entity\_name *dns.t\_hostname*
- p\_service *commons.t\_key*
- p\_subservice *commons.t\_key*

**Returns** void

```
IF p_condition THEN
    PERFORM backend._notify_service_entity_name(p_service_entity_name, p_service, p_
↳subservice);
ELSE
    PERFORM commons._raise_inaccessible_or_missing();
END IF;
```

**10.2.4 backend.\_deleted**

Is 'del'

**Parameters**

- backend\_status *backend.t\_status*

**Returns** boolean

```
RETURN backend_status IS NOT NULL AND backend_status = 'del';
```

### 10.2.5 backend.\_get\_login

Shows informations for the current backend login. Throws an error if the current user is not a grantee for a machine.

**Parameters** *None*

**Returns** TABLE

**Returned columns**

- machine *dns.t\_hostname*

```
IF (SELECT TRUE FROM "backend"."auth"
    WHERE "role"=session_user)
THEN
    RETURN QUERY SELECT backend.auth.machine FROM backend.auth
        WHERE "role"=session_user;
ELSE
    RAISE 'Connected role `%` is not a grantee for a machine.', session_user;
END IF;
```

### 10.2.6 backend.\_login\_machine

Shows machine for the current backend login.

**Parameters** *None*

**Returns** dns.t\_hostname

```
RETURN (SELECT machine FROM backend._get_login());
```

### 10.2.7 backend.\_machine\_priviledged

Checks if a currently connected machine is priviledged to obtain data for a certain service for a certain domain name.

**Warning:** The parameter `p_domain` must be a domain, which means an entry in the column `dns.service.domain`. It must not be confused with a `service_entity_name`.

**Parameters**

- p\_service *commons.t\_key*
- p\_domain *dns.t\_hostname*

**Returns** boolean



```

RETURN COALESCE (
(
  SELECT TRUE FROM system.service_entity_machine AS t
  JOIN dns.service AS s
  ON
    s.service = p_service AND
    s.domain = p_domain

  WHERE
    t.service = p_service AND
    t.service_entity_name = s.service_entity_name AND
    t.machine_name = backend._login_machine()
)
, FALSE);

```

### 10.2.8 backend.\_machine\_priviledged\_service

Checks if a currently connected machine is priviledged to obtain data for a certain service for a certain servicee name.

**Warning:** The parameter `p_service_entity_name` must be the name of a service entity. It must not be confused with a domain.

#### Parameters

- `p_service` *commons.t\_key*
- `p_service_entity_name` *dns.t\_hostname*

**Returns** boolean

```

RETURN COALESCE (
(
  SELECT TRUE FROM system.service_entity_machine AS t
  WHERE
    t.service = p_service AND
    t.service_entity_name = p_service_entity_name AND
    t.machine_name = backend._login_machine()
)
, FALSE);

```

### 10.2.9 backend.\_notify

Informs a machine about changes. To listen to signals use

```
LISTEN "carnivora/machine.name.example"
```

on the machine. The payload has the form `<service_entity_name>/<service>/<subservice>`. For example `mail.domain.example/email/mailbox` for a mailbox related update.

#### Parameters

- `p_machine` *dns.t\_hostname*
- `p_service_entity_name` *dns.t\_hostname*
- `p_service` *commons.t\_key*

- p\_subservice *commons.t\_key*

**Returns** void

**PERFORM**

```
pg_notify(  
    'carnivora/' || p_machine,  
    p_service_entity_name || '/' || p_service || '/' || p_subservice  
);
```

## 10.2.10 backend.\_notify\_domain

Informs all machines about changes.

**Warning:** The parameter p\_domain must be a domain, which means an entry in the column dns.service.domain. It must not be confused with a service\_entity\_name.

**Parameters**

- p\_service *commons.t\_key*
- p\_subservice *commons.t\_key*
- p\_domain *dns.t\_hostname*

**Returns** void

**PERFORM**

```
backend._notify(machine_name, s.service_entity_name, p_service, p_subservice)
```

```
FROM system.service_entity_machine AS t
```

```
JOIN dns.service AS s
```

```
ON
```

```
    s.service = p_service AND
```

```
    s.domain = p_domain
```

```
WHERE
```

```
    t.service = p_service AND
```

```
    t.service_entity_name = s.service_entity_name
```

```
;
```

## 10.2.11 backend.\_notify\_service\_entity\_name

Informs all machines about changes.

**Warning:** The parameter p\_service\_entity\_name must be a service name. It must not be confused with a domain.

**Parameters**

- p\_service\_entity\_name *dns.t\_hostname*
- p\_service *commons.t\_key*
- p\_subservice *commons.t\_key*

**Returns** void

```
PERFORM
    backend._notify(machine_name, p_service_entity_name, p_service, p_subservice)

FROM system.service_entity_machine AS t
    WHERE
        t.service = p_service AND
        t.service_entity_name = p_service_entity_name
;
```

## 10.3 Domains

### 10.3.1 backend.t\_status

Backend status

## 10.4 Roles

### 10.4.1 backend

vms

**Login** *Disabled*



### Carnivora Commons

Usefull templates, functions and domains.

#### Schema Contents

- *Functions*

- `commons._hash_password`
- `commons._idn`
- `commons._jsonb_to_array`
- `commons._passwords_equal`
- `commons._raise_inaccessible_or_missing`
- `commons._reverse_array`
- `commons._uuid`

- *Domains*

- `commons.t_port`
- `commons.t_password`
- `commons.t_password_plaintext`
- `commons.t_key`
- `commons.t_hexvarchar`

- *Sequences*

- `commons.uid`

## 11.1 Functions

### 11.1.1 `commons._hash_password`

SHA512 hash of the password with 16 characters random salt. The returned format is the traditional 'crypt(3)' format.

**Parameters**

- `p_password` *commons.t\_password\_plaintext*

**Language** plpython3u

**Returns** `commons.t_password`

```
import crypt

return crypt.crypt(p_password, crypt.METHOD_SHA512)
```

### 11.1.2 `commons._idn`

Converts a unicode domain name to IDN (ASCII)

Currently using IDNA2003.

**Parameters**

- `p_domain` *varchar*

**Language** plpython3u

**Returns** *varchar*

**Execute privilege**

- *userlogin*
- *backend*

```
if p_domain is None:
    return None

if p_domain.lower() != p_domain:
    raise plpy.Error('Only lower case IDNs are allowed and can be handled.')

return p_domain.encode('idna').decode()
```

### 11.1.3 `commons._jsonb_to_array`

Converts a JSONB array to a PostgreSQL text[] array

**Parameters**

- `p_jsonb` *jsonb*

**Returns** `text[]`

```
RETURN ARRAY(SELECT jsonb_array_elements_text(p_jsonb));
```

### 11.1.4 commons.\_passwords\_equal

Compares a plaintext password with an arbitrary 'crypt(3)' hashed password.

Uses <<https://docs.python.org/3/library/hmac.html>>

#### Parameters

- p\_password\_plaintext *commons.t\_password\_plaintext*
- p\_password\_hash *commons.t\_password*

**Language** plpython3u

**Returns** boolean

```
import crypt
from hmac import compare_digest as compare_hash

# Giving crypt.crypt the full hash as second argument fixes the use of the
# right salt and algorithm. Using compare_hash to avoid timing attacks.
return compare_hash(crypt.crypt(p_password_plaintext, p_password_
↪hash))
```

### 11.1.5 commons.\_raise\_inaccessible\_or\_missing

Raised whenever a operation on an object failes because it is not owned by the user or it is not found.

#### Parameters

- p\_raise *boolean*
- Controls if the exception is raised

**Returns** void

```
IF NOT COALESCE(p_raise, FALSE) THEN
    RAISE 'Object inaccessible or missing'
    USING DETAIL = '$carnivora:commons:inaccessible_or_missing$';
END IF;
```

### 11.1.6 commons.\_reverse\_array

Copied from <[https://wiki.postgresql.org/wiki/Array\\_reverse](https://wiki.postgresql.org/wiki/Array_reverse)>

#### Parameters

- p\_array *anyarray*

**Language** sql

**Returns** anyarray

**Execute privilege**

- *userlogin*
- *backend*

```
SELECT
  ARRAY (
    SELECT $1[i]
    FROM generate_subscripts($1,1) AS s(i)
    ORDER BY i DESC
  );
```

### 11.1.7 commons.\_uuid

Returns a random uuid

**Parameters** *None*

**Returns** uuid

```
RETURN public.uuid_generate_v4();
```

## 11.2 Domains

### 11.2.1 commons.t\_port

Port

**Checks**

- **invalid\_port** Only allow port values

```
VALUE BETWEEN 0 AND 65535
```

### 11.2.2 commons.t\_password

unix hash thingy

---

**Todo:** proper checking of format

---

**Checks**

- **crypt (3) password format** Only allows SHA512 strings.

```
VALUE ~ '^\$6\$[.\-/a-zA-Z0-9]{8,16}\$[.\-/a-zA-Z0-9]{86}\$'
```

### 11.2.3 commons.t\_password\_plaintext

Password in plaintext

**Checks**

- **minimum password length 8** Ensures that passwords at least have 8 chars

```
character_length(VALUE) >= 8
```



### 11.2.4 commons.t\_key

Key

### 11.2.5 commons.t\_hexvarchar

Varchar only with HEX values

#### Checks

- **invalid characters** Only allows numbers and chars a-f for hex representation

VALUE ~ '^[0-9a-f]\*\$'

## 11.3 Sequences

### 11.3.1 commons.uid

Unix user id



### Carnivora System

Manages services, service entities and contingents.

#### Schema Contents

- *Tables*

- `system.inherit_contingent`
- `system.service`
- `system.service_entity`
- `system.service_entity_dns`
- `system.service_entity_machine`
- `system.subservice`
- `system.subservice_entity`
- `system.subservice_entity_contingent`
- `system.subservice_entity_domain_contingent`

- *Functions*

- `system._contingent_ensure`
- `system._effective_contingent`
- `system._effective_contingent_domain`
- `system._inherit_contingent_donor`
- `system._setup_register_service`
- `system._setup_register_subservice`

```
- system.sel_inherit_contingent
- system.sel_usable_host
```

## 12.1 Tables

### 12.1.1 system.inherit\_contingent

Contingents inherited from other users.

Precedence is unambiguous via primary key.

#### Primary key

- owner
- priority

#### Columns

- **owner** *user.t\_user* Owner  
References *user.user.owner*  
On Delete: CASCADE  
On Update: CASCADE
- **donor** *user.t\_user* Donor of contingent  
References *user.user.owner*  
On Delete: CASCADE  
On Update: CASCADE
- **priority** *int* Priority, higher values take precedence

### 12.1.2 system.service

Services

Just a list of services that exist. Modules do register their services here. Use `system._setup_register_service(<module>, <service>)` to insert into this table.

#### Primary key

- service

#### Columns

- **option** *jsonb* Free options in JSON format

#### Default

```
'{}'
```

- **service** *commons.t\_key* Service name
- **module** *commons.t\_key* Module name, just to keep track who uses this name

### 12.1.3 system.service\_entity

#### Service Entity

Names under which services are made available. For example (mail.example.org, email) could be a mail-server system referred to as mail.example.org by carnivora. Such a system can consist of multiple physical or virtual machines. The corresponding machines are listed in system.service\_entity\_machine. A core feature of services is the definition of ‘templates’ for dns records which have to be present for every domain that uses this service. Such ‘templates’ can be defined in system.service\_dns. Domain names can be enabled for services in dns.service. Service enabled domains are automatically equipped with the required dns entries according to the existing ‘templates’.

The service\_entity\_name might be exposed to users as the address of this service. For example as SMTP or SSH server etc. The exact interpretation of the service\_entity\_name depends on the module and the frontend.

#### Primary key

- service\_entity\_name
- service

#### Columns

- **option** *jsonb* Free options in JSON format

#### Default

```
'{}'
```

- **service\_entity\_name** *dns.t\_hostname* Host name
- **service** *commons.t\_key* email, ssh, ...

References *system.service.service*

### 12.1.4 system.service\_entity\_dns

#### Service Entity DNS

Resource records that have to be present to use a service. The records in this table can be understood as ‘templates’. The table does not contain a name (domain) for the records. Rather for every domain that uses this service, all appropriate records are created for this domain based on this table. The assignment from domain to services can be found in dns.service.

#### Primary key

- id

#### Foreign keys

- Reference service entity

#### Local Columns

- service\_entity\_name
- service

#### Referenced Columns

- *system.service\_entity.service\_entity\_name*
- *system.service\_entity.service*

#### Columns

- **service\_entity\_name** *dns.t\_hostname* Service entity name
- **service** *commons.t\_key* Service (e.g. email, jabber)
- **type** *dns.t\_type* Type (A, AAAA, CNAME, MX, SRV, TXT, ...)
- **rdata** *dns.t\_rdata* fancy rdata storage
- **ttdl** *NULL* | *dns.t\_ttl* Time to live, NULL indicates default value
- **option** *jsonb* Free options in JSON format

**Default**

```
'{}'
```

- **id** *uuid* uuid serial number to identify database elements uniquely

**Default**

```
commons._uuid()
```

- **domain\_prefix** *NULL* | *varchar* Domain prefix

### 12.1.5 system.service\_entity\_machine

#### Service Entity Machine

List of machines that provide a certain service. This information is used to provide these machines access to the data they need to provide the service. See also the module ‘backend’.

**Primary key**

- machine\_name
- service\_entity\_name
- service

**Foreign keys**

- Reference service entity

**Local Columns**

- service\_entity\_name
- service

**Referenced Columns**

- *system.service\_entity.service\_entity\_name*
- *system.service\_entity.service*

**Columns**

- **service\_entity\_name** *dns.t\_hostname* Service entity name
- **service** *commons.t\_key* Service (e.g. email, jabber)
- **option** *jsonb* Free options in JSON format

**Default**

```
'{'}
```

- **machine\_name** *dns.t\_hostname* Assigns machine

References *backend.machine.name*

## 12.1.6 system.subservice

Subservices

### Primary key

- service
- subservice

### Columns

- **service** *commons.t\_key* Service  
References *system.service.service*
- **subservice** *commons.t\_key* Subservice (concretization the service)

## 12.1.7 system.subservice\_entity

Subservice Entity

Names under which subservices are made available.

See also: Table *system.service\_entity*

### Primary key

- service\_entity\_name
- service
- subservice

### Foreign keys

- service ent

#### Local Columns

- service\_entity\_name
- service

#### Referenced Columns

- *system.service\_entity.service\_entity\_name*
- *system.service\_entity.service*

- subservice

#### Local Columns

- service
- subservice

#### Referenced Columns

- *system.subservice.service*
- *system.subservice.subservice*

## Columns

- **option** *jsonb* Free options in JSON format

### Default

```
' {} '
```

- **service\_entity\_name** *dns.t\_hostname* Service entity name
- **service** *commons.t\_key* Service name
- **subservice** *commons.t\_key* account, alias, ...

## 12.1.8 system.subservice\_entity\_contingent

Subservice entity contingent

### Primary key

- service
- subservice
- service\_entity\_name
- owner

### Foreign keys

- Reference service entity

#### Local Columns

- service\_entity\_name
- service

#### Referenced Columns

- *system.service\_entity.service\_entity\_name*
- *system.service\_entity.service*

- Reference subservice entity

#### Local Columns

- service\_entity\_name
- service
- subservice

#### Referenced Columns

- *system.subservice\_entity.service\_entity\_name*
- *system.subservice\_entity.service*
- *system.subservice\_entity.subservice*

## Columns



- **service\_entity\_name** *dns.t\_hostname* Service entity name
- **service** *commons.t\_key* Service (e.g. email, jabber)
- **subservice** *commons.t\_key* Subservice (e.g. account, alias)
- **owner** *user.t\_user* Owner  
References *user.user.owner*  
On Delete: CASCADE  
On Update: CASCADE
- **domain\_contingent** *integer* Limit per domain
- **total\_contingent** *integer* Limit on the total

### 12.1.9 system.subservice\_entity\_domain\_contingent

Subservice entity per domain contingent

#### Primary key

- service
- subservice
- service\_entity\_name
- domain
- owner

#### Foreign keys

- Reference service entity

##### Local Columns

- service\_entity\_name
- service

##### Referenced Columns

- *system.service\_entity.service\_entity\_name*
- *system.service\_entity.service*

- Reference subservice entity

##### Local Columns

- service\_entity\_name
- service
- subservice

##### Referenced Columns

- *system.subservice\_entity.service\_entity\_name*
- *system.subservice\_entity.service*
- *system.subservice\_entity.subservice*

#### Columns

- **service\_entity\_name** *dns.t\_hostname* Service entity name
- **service** *commons.t\_key* Service (e.g. email, jabber)
- **subservice** *commons.t\_key* Subservice (e.g. account, alias)
- **owner** *user.t\_user* Owner  
References *user.user.owner*  
On Delete: CASCADE  
On Update: CASCADE
- **domain** *dns.t\_hostname* Specific domain for which the access is granted
- **domain\_contingent** *integer* Limit per domain

## 12.2 Functions

### 12.2.1 system.\_contingent\_ensure

Throws exceptions if the contingent is exceeded

#### Parameters

- p\_owner *user.t\_user*
- p\_service *commons.t\_key*
- p\_subservice *commons.t\_key*
- p\_domain *dns.t\_hostname*
- p\_current\_quantity\_total *integer*
- p\_current\_quantity\_domain *integer*

#### Variables defined for body

- v\_total\_contingent *integer*
- v\_domain\_contingent *integer*
- v\_domain\_contingent\_default *integer*
- v\_domain\_contingent\_specific *integer*
- v\_service\_entity\_name *dns.t\_hostname*
- v\_domain\_owner *user.t\_user*

**Returns** void

```
IF p_owner IS NULL
THEN
    RAISE 'Owner argument must not be NULL.';
END IF;

SELECT
    t.service_entity_name,
    s.owner
INTO
    v_service_entity_name,
```

(continues on next page)

(continued from previous page)

```

        v_domain_owner
FROM dns.service AS t
JOIN dns.registered AS s
    ON s.domain = t.registered

WHERE
    t.domain = p_domain AND
    t.service = p_service;

-- check dns.service entry
IF v_domain_owner IS NULL
THEN
    RAISE 'Contingent check impossible, since dns.service entry missing.'
    USING
        DETAIL = '$carnivora:system:no_contingent$',
        HINT = (p_owner, p_service, p_domain);
END IF;

SELECT domain_contingent, total_contingent
    INTO v_domain_contingent_default, v_total_contingent
FROM system._effective_contingent()
WHERE
    service = p_service AND
    subservice = p_subservice AND
    service_entity_name = v_service_entity_name AND
    owner = p_owner
;

SELECT domain_contingent
    INTO v_domain_contingent_specific
FROM system._effective_contingent_domain()
WHERE
    service = p_service AND
    subservice = p_subservice AND
    service_entity_name = v_service_entity_name AND
    owner = p_owner
;

v_domain_contingent :=
    COALESCE(v_domain_contingent_default, v_domain_contingent_specific);

IF
    v_total_contingent IS NULL AND
    v_domain_contingent IS NULL
THEN
    RAISE 'You do not have a contingent'
    USING
        DETAIL = '$carnivora:system:no_contingent$',
        HINT = (p_owner, p_service, v_service_entity_name);
END IF;

IF v_domain_contingent IS NULL AND p_owner <> v_domain_owner
THEN
    RAISE 'You are not the owner of the registered domain'
    USING
        DETAIL = '$carnivora:system:contingent_not_owner$',
        HINT = (p_owner, p_service, v_service_entity_name);

```

(continues on next page)

(continued from previous page)

```

END IF;

IF v_total_contingent <= p_current_quantity_total
THEN
    RAISE 'Total contingent exceeded'
    USING
        DETAIL = '$carnivora:system:contingent_total_exceeded$',
        HINT = (p_owner, p_service, p_domain, v_total_contingent);
END IF;

IF v_domain_contingent <= p_current_quantity_domain
THEN
    RAISE 'Domain contingent exceeded'
    USING
        DETAIL = '$carnivora:system:contingent_domain_exceeded$',
        HINT = (p_owner, p_service, p_domain, v_domain_contingent);
END IF;

```

## 12.2.2 system.\_effective\_contingent

contingent

Parameters *None*

Returns TABLE

Returned columns

- service *commons.t\_key*
- subservice *commons.t\_key*
- service\_entity\_name *dns.t\_hostname*
- owner *user.t\_user*
- domain\_contingent *int*
- total\_contingent *int*

```

RETURN QUERY
SELECT
    DISTINCT ON
        (contingent.service, contingent.subservice, contingent.service_entity_name, usr.
        ↪owner)
        contingent.service,
        contingent.subservice,
        contingent.service_entity_name,
        usr.owner,
        contingent.domain_contingent,
        contingent.total_contingent
FROM system.subservice_entity_contingent AS contingent

CROSS JOIN "user"."user" AS usr

JOIN system._inherit_contingent_donor(usr.owner) AS des
    ON des.donor = contingent.owner

```

(continues on next page)

(continued from previous page)

```
ORDER BY
    contingent.service,
    contingent.subservice,
    contingent.service_entity_name,
    usr.owner,
    des.priority_list DESC;
```

### 12.2.3 system.\_effective\_contingent\_domain

contingent

**Parameters** *None*

**Returns** TABLE

**Returned columns**

- service *commons.t\_key*
- subservice *commons.t\_key*
- service\_entity\_name *dns.t\_hostname*
- domain *dns.t\_hostname*
- owner *user.t\_user*
- domain\_contingent *int*

```
RETURN QUERY
SELECT
    DISTINCT ON
        (contingent.service, contingent.subservice, contingent.service_entity_name,
        ↪contingent.domain, usr.owner)
    contingent.service,
    contingent.subservice,
    contingent.service_entity_name,
    contingent.domain,
    usr.owner,
    contingent.domain_contingent
FROM system.subservice_entity_domain_contingent AS contingent

CROSS JOIN "user"."user" AS usr

JOIN system._inherit_contingent_donor(usr.owner) AS des
    ON des.donor = contingent.owner

ORDER BY
    contingent.service,
    contingent.subservice,
    contingent.service_entity_name,
    contingent.domain,
    usr.owner,
    des.priority_list DESC;
```

### 12.2.4 `system._inherit_contingent_donor`

Returns all contingent donors for a given user with their priority.

#### Parameters

- `p_owner` *user:t\_user*

Returns TABLE

#### Returned columns

- `donor` *user:t\_user* User from which contingents are inherited
- `priority_list` *integer[]*

```
RETURN QUERY
WITH RECURSIVE contingent_donor(donor, priority_list, cycle_detector) AS
(
    -- cast to varchar, since arrays of t_user are not defined
    SELECT p_owner, ARRAY[]::integer[], ARRAY[CAST(p_owner AS varchar)]

    UNION

    SELECT
        curr.donor,
        prev.priority_list || curr.priority,
        cycle_detector || CAST(curr.donor AS varchar)
    FROM system.inherit_contingent AS curr
    JOIN contingent_donor AS prev
    ON
        prev.donor = curr.owner AND
        curr.donor <> ALL (prev.cycle_detector)
)
SELECT
    contingent_donor.donor,
    array_append(contingent_donor.priority_list, NULL)
FROM contingent_donor
-- Appending the NULL changes the ordering between arrays with different size
ORDER BY array_append(contingent_donor.priority_list, NULL) DESC;
```

### 12.2.5 `system._setup_register_service`

Allows modules to register their services during setup. Returns the total number of service names registered for this module.

#### Parameters

- `p_module` *commons.t\_key*
- `p_service` *commons.t\_key*

Returns void

```
INSERT INTO system.service
(module, service)
SELECT p_module, p_service
WHERE NOT EXISTS (
    SELECT service FROM system.service
```

(continues on next page)

(continued from previous page)

```
WHERE module=p_module AND service=p_service
);
```

## 12.2.6 system.\_setup\_register\_subservice

Allows modules to register their services during setup. Returns the total number of service names registered for this module.

### Parameters

- p\_service *commons.t\_key*
- p\_subservice *commons.t\_key*

**Returns** void

```
INSERT INTO system.subservice
(service, subservice)
SELECT p_service, p_subservice
WHERE NOT EXISTS (
  SELECT service FROM system.subservice
  WHERE service=p_service AND subservice=p_subservice
);
```

## 12.2.7 system.sel\_inherit\_contingent

Select inherit contingent

**Parameters** *None*

**Variables defined for body**

- v\_owner *user.t\_user*

**Returns** TABLE

**Returned columns**

- owner *user.t\_user*
- donor *user.t\_user*
- priority *int*

**Execute privilege**

- *userlogin*

```
-- begin userlogin prelude
v_owner := (SELECT t.act_as FROM "user"._get_login() AS t);
-- end userlogin prelude

RETURN QUERY
  SELECT t.owner, t.donor, t.priority
  FROM system.inherit_contingent AS t
  ORDER BY t.owner, t.priority;
```

## 12.2.8 `system.sel_usable_host`

Usable hosts

### Parameters

- `p_service` *commons.t\_key*

### Variables defined for body

- `v_owner` *user.t\_user*

Returns TABLE

### Returned columns

- `subservice` *commons.t\_key*
- `service_entity_name` *dns.t\_hostname*

### Execute privilege

- *userlogin*

```
-- begin userlogin prelude
v_owner := (SELECT t.act_as FROM "user"._get_login() AS t);
-- end userlogin prelude

RETURN QUERY
    SELECT t.subservice, t.service_entity_name FROM system._effective_contingent() AS t
    ↪t
    WHERE
        owner = v_owner AND
        t.service = p_service AND
        t.total_contingent > 0
    ORDER BY
        t.service_entity_name
    ;
```



Carnivora Users: Users own things objects in the DB, and they can login into frontends (edentata)

### Schema Contents

- *Tables*

- `user.deputy`
- `user.session`
- `user.user`

- *Functions*

- `user._get_login`
- `user._login_user`
- `user._session_id`
- `user.del_login`
- `user.ins_deputy`
- `user.ins_login`
- `user.sel_deputy`
- `user.upd_user`

- *Domains*

- `user.t_user`

- *Roles*

- `userlogin`
- `system`

## 13.1 Tables

### 13.1.1 `user.deputy`

Deputies for users

#### Primary key

- `deputy`
- `represented`

#### Columns

- **`deputy`** *`user.t_user`* Deputy  
References *`user.user.owner`*  
On Delete: CASCADE  
On Update: CASCADE
- **`represented`** *`user.t_user`* User for which the deputy can act  
References *`user.user.owner`*  
On Delete: CASCADE  
On Update: CASCADE

### 13.1.2 `user.session`

User login sessions

#### Primary key

- `id`

#### Columns

- **`owner`** *`user.t_user`* Owner  
References *`user.user.owner`*  
On Delete: CASCADE  
On Update: CASCADE
- **`id`** *`varchar`* Session id

#### Default

```
"user"._session_id()
```

- **`act_as`** *`user.t_user`* Act as
- **`started`** *`timestamp`* Session started at this time

#### Default

```
CURRENT_TIMESTAMP
```

### 13.1.3 user.user

Users

Users with password set to `NULL` can be used as groups.

**Primary key**

- owner

**Columns**

- **option** *jsonb* Free options in JSON format

**Default**

```
'{}'
```

- **owner** *user.t\_user* User name, login name
- **password** `NULL` | *commons.t\_password* Unix shadow crypt format, `NULL` value disables login
- **contact\_email** `NULL` | *email.t\_address* Optional contact email address, can be used as login name

## 13.2 Functions

### 13.2.1 user.\_get\_login

Shows informations for the current user login. Throws an exception if no login is associated to the current database connection.

**Parameters** *None*

**Returns** TABLE

**Returned columns**

- owner *user.t\_user*
- act\_as *user.t\_user*

```
IF (SELECT TRUE FROM "user"."session"
    WHERE "id"="user"._session_id())
THEN
    RETURN QUERY SELECT t.owner, t.act_as FROM "user"."session" AS t
    WHERE "id"="user"._session_id();
ELSE
    RAISE 'Database connection is not associated to a user login.'
    USING HINT := 'Use user.ins_login(...) first.';
END IF;
```

### 13.2.2 user.\_login\_user

Shows informations for the current user login. Throws an exception if no login is associated to the current database connection.

**Parameters** *None*

**Returns** *user.t\_user*

```
RETURN (SELECT owner FROM "user"._get_login());
```

### 13.2.3 user.\_session\_id

Gives an id for the database connection that is unique over all database connections. It is used to identify user logins.

Not sure if this stays unique with distributed infrastructure!

**Parameters** *None*

**Returns** varchar

```
RETURN pg_backend_pid()::varchar;
```

### 13.2.4 user.del\_login

Try to logout

**Parameters** *None*

**Returns** void

**Execute privilege**

- *userlogin*

```
DELETE FROM "user".session WHERE id = "user"._session_id();

IF NOT FOUND THEN
    RAISE 'Carnivora: user logout failed, not logged in'
    USING DETAIL = '$carnivora:user:logout_failed$';
END IF;
```

### 13.2.5 user.ins\_deputy

Act as deputy

**Parameters**

- p\_act\_as *user:t\_user*

**Variables defined for body**

- v\_owner *user:t\_user*

**Returns** void

**Execute privilege**

- *userlogin*

```
-- begin userlogin prelude
v_owner := (SELECT t.act_as FROM "user"._get_login() AS t);
-- end userlogin prelude

UPDATE "user".session AS t
```

(continues on next page)

(continued from previous page)

```

SET act_as = p_act_as
FROM "user".deputy AS s
WHERE
    s.deputy = t.owner AND
    s.represented = p_act_as AND
    t.id = "user"._session_id() AND
    t.owner = v_owner;

IF NOT FOUND THEN
    RAISE 'Acting as deputy failed.'
    USING DETAIL := '$carnivora:user:deputy_failed$';
END IF;

```

### 13.2.6 user.ins\_login

Try to bind database connection to new user session.

#### Parameters

- p\_login *varchar*
- p\_password *commons.t\_password\_plaintext*

#### Variables defined for body

- v\_login\_owner *user.t\_user*

#### Returns TABLE

#### Returned columns

- user *user.t\_user*

#### Execute privilege

- *userlogin*

```

SELECT owner INTO v_login_owner FROM "user"."user" AS t
WHERE
    p_login IS NOT NULL AND
    t.password IS NOT NULL AND
    lower(p_login) IN (owner, contact_email) AND
    commons._passwords_equal(p_password, t.password);

IF v_login_owner IS NOT NULL THEN
    INSERT INTO "user"."session" (owner, act_as) VALUES (v_login_owner, v_login_owner);
    RETURN QUERY SELECT v_login_owner;
ELSE
    RAISE 'Carnivora: invalid user login'
    USING DETAIL = '$carnivora:user:login_invalid$';
END IF;

```

### 13.2.7 user.sel\_deputy

sel deputy

Parameters *None*

**Variables defined for body**

- v\_owner *user.t\_user*

**Returns** TABLE**Returned columns**

- represented *user.t\_user*

**Execute privilege**

- *userlogin*

```
-- begin userlogin prelude
v_owner := (SELECT t.act_as FROM "user"._get_login() AS t);
-- end userlogin prelude

RETURN QUERY
  SELECT t.represented FROM "user".deputy AS t
  WHERE t.deputy = "user"._login_user()
  ORDER BY t.represented;
```

## 13.2.8 user.upd\_user

change user passwd

**Parameters**

- p\_password *commons.t\_password\_plaintext*

**Variables defined for body**

- v\_owner *user.t\_user*

**Returns** void**Execute privilege**

- *userlogin*

```
-- begin userlogin prelude
v_owner := (SELECT t.act_as FROM "user"._get_login() AS t);
-- end userlogin prelude

UPDATE "user".user
  SET password = commons._hash_password(p_password)
WHERE
  owner = "user"._login_user();
```

## 13.3 Domains

### 13.3.1 user.t\_user

Username

**Checks**

- **valid\_characters** Only lower-case letters, numbers and .-\_

VALUE ~ '^[a-z0-9.\-\_]+\$'

## 13.4 Roles

### 13.4.1 userlogin

Do user actions via this group

**Login** *Disabled*

### 13.4.2 system

Highly priviledged user

**Login** *Disabled*





### 14.1 Types

#### 14.1.1 `anyarray`

#### 14.1.2 `bigint`

#### 14.1.3 `boolean`

#### 14.1.4 `integer`

- `int`

#### 14.1.5 `jsonb`

- <https://www.postgresql.org/docs/current/static/datatype-json.html>

#### 14.1.6 `timestamp`

#### 14.1.7 `uuid`

- <https://www.postgresql.org/docs/current/static/uuid-osp.html>

#### 14.1.8 `varchar`

- <https://www.postgresql.org/docs/current/static/datatype-character.html>



## 15.1 Types

### 15.1.1 `serial`

Managed via HamSql