CanIt-Domain-PRO API Guide

for Version 10.2.5

AppRiver, LLC

19 September 2018

Contents

Chapter 1

Introduction

1.1 Overview

The CanIt REST API is a REST-based remote interface to CanIt-Domain-PRO.

The main goal of this remote API is to ease integration with other systems, by exposing access to CanIt-Domain-PRO data with a minimum of external requirements. Client access to this API can use one of the libraries described in Chapter 3. Alternatively, if you wish to access the API from a language other than Perl, Python or PHP, you can write your own wrapper using the HTTP client library of your choice.

1.2 REST Basics

REST stands for **RE**presentational **S**tate **T**ransfer. The basic idea of REST is that we use URIs to describe the objects we're manipulating. The verbs for manipulating them are the standard HTTP protocol GET/POST/PUT/DELETE operations which roughly correspond to "Retrieve", "Update", "Create" and "Delete".

URIs should not include any actions but instead just be a representation of the thing we're viewing/modifying. For example, a domain rule creation action for the domain "example.org" is implemented as:

`PUT /api/2.0/realm/base/stream/default/rule/domain/example.org`

with rule settings in the body of the PUT method.

In practice, the API is not pure REST, as it uses some action URIs to log in and create an API session and to log out.

The HTTP methods for GET, POST, PUT, and DELETE, as defined in HTTP 1.1, are used in this version of the API.

1.2.1 GET

The GET method is used to fetch a particular piece of information. It does not make any changes on the server side. For example:

```
GET /api/2.0/incident/1234
GET /api/2.0/realm/foo-com/stream/default/setting/AutoReject
```

A successful GET returns **200 OK** and the content for that resource. The content is formatted in the desired serialization format (as specified by the **Accept:** header) by the server.

In case of error, a number of HTTP errors may be returned, including **404 Not Found** for an nonexistent resource, **400 Bad Request** for poorly-formatted request information, **405 Method Not Allowed** for unsupported actions, and **500 Server Error** for processing problems on the server.

Some GET requests take query parameters. For example:

```
GET /api/2.0/log/search/0/40?subject=Test&sender=bob@example.org
```

This does not adhere strictly to the REST philosophy, but is convenient in some cases. We strongly recommend that you use one of our API client libraries which take care of properly-encoding query parameters for GET requests. This is more reliable than building up your own GET request string by hand.

1.2.2 POST

The POST method is used for creating new resources on the server when the identifying URI for that resource isn't yet known and for partial updates of existing resources. Example:

```
POST /api/2.0/realm/base/stream/default/setting/AutoReject
POST /api/2.0/incident/12345
```

The body of all POST messages must consist of `multipart/form-data` or `application/x-www-url-encoded` as defined in RFC 2388 and RFC 1867.

A POST that creates a new resource successfully returns a **201 Created** code with a Location: header pointing to the new resource.

A POST that updates an existing resource successfully returns a **204 No Content**.

In case of error, an appropriate HTTP error code is returned.

In addition to the standard HTTP codes mentioned for GET, invalid or badly-formatted POST data can return a **400 Bad Request** with a body explaining the failure in detail (such as which values supplied were invalid).

1.2.3 PUT

The PUT method is used for creating new resources on the server when the identifying URI for that resource **is** known or for wholesale replacement of a resource on the server. The data is provided in the body of the PUT. Example:

```
PUT /api/2.0/realm/foobar
```

The body of all PUT messages can consist of any supported serialization format. At present, YAML and JSON are supported, although YAML is deprecated.

A PUT that creates a new resource returns **201 Created** with a Location: header pointing to the canonical URI for that resource.

A PUT that updates an existing resource returns **204 No Content**

A failed PUT can also return the same errors as a POST.

Note: Most API calls that support PUT also support POST with the same parameters. The difference between PUT and POST is that a PUT will usually fail if the resource exists already, whereas a POST will create a new resource (if one does not exist) or update an already-existing resource.

1.2.4 DELETE

The DELETE method is used for removing the specified resource. Examples:

```
DELETE /api/2.0/realm/base/stream/default/rule/domain/example.com
DELETE /api/2.0/realm/somerealm
```

A successful DELETE returns **204 No Content**. An unsuccessful DELETE on a resource that did not already exist returns **404 Not Found**. It is up to the client to determine if a 404 on a DELETE is considered failure or success.

A failure to delete returns a **400 Bad Request** or **500 Server Error** depending on the reason for the failure.

1.3 Standard URIs

All API URIs are HTTP or HTTPS URIs. In canonical form, they should look like:

`http[s]://servername[:port]/`*`path-prefix`*`/api/`*`version_number`*`/rest-of-uri`

The path portion of the URI should always begin with **`/`*`path-prefix`*`/api/`*`version_number`***. For this release, the version number is **2.0**. The ***`path-prefix`*** portion depends on your CanIt-Domain-PRO installation; it is typically **`canit`**. In the rest of this document, we will *only* show URIs starting from **`/api/2.0`**, ignoring any possible ***`path-prefix`***.

Following the API version is the portion of the URI that identifies what we're accessing or modifying.

Many API URIs require a realm and stream name, or a realm and user name. For these URIs, these are specified immediately after the version number, followed by the specific identifier for the object being accessed. For example:

```
GET /api/2.0/realm/base/stream/default/settings
```

lists stream settings for the 'default' stream in the 'base' realm.

URIs that manipulate things that are not realm and stream qualified (such as incidents) have their identifying portion following directly after the version number. For example:

```
GET /api/2.0/incident/1122334455
```

retrieves information about incident 1122334455.

1.4 Data Serialization Formats

The data sent to or received from the API server can be encoded in one of two formats: YAML or JSON. Although (for historical reasons) the default format is YAML, we recommend using JSON instead. The server-side implementation of YAML serialization and deserialization is buggy and can cause problems with some inputs. We retain it only for backward compatibility.

To send data in a format other than YAML, provide the PUT data in that format with an appropriate **Content-Type:** header containing the MIME type of that format.

To receive data in a format other than YAML, the client should specify the MIME type of that format in an **Accept:** header.

If, for some reason, a client cannot specify the MIME type in an **Accept:** header, then it can force a particular format by prefixing the URL with **`/yaml`** or **`/json`**. For example:

`GET /api/2.0/realm/`*`some_realm`*`/streams`
(List all streams in the default serialization format or the format specified in an **Accept:** header.)

`GET /api/2.0/yaml/realm/`*`some_realm`*`/streams`
(List all streams, forcing the return data to be in YAML format.)

`GET /api/2.0/json/realm/`*`some_realm`*`/streams`
(List all streams, forcing the return data to be in JSON format.)

Note: POST requests *never* use YAML or JSON for *submitting* data. The request data must *always* be of type `multipart/form-data` or `application/x-www-url-encoded`. However, POST requests *return* YAML or JSON data.

1.4.1 YAML

YAML (Yet Another Markup Language) format is a text based serialization format, with a MIME type `text/x-yaml`.

Full information on YAML can be found at `http://yaml.org/spec/`, or in the documentation for YAML and YAML::Syck on CPAN.

Note: YAML is (for historical reasons) the default serialization format, but is deprecated. You should use JSON instead.

1.4.2 JSON

JSON (JavaScript Object Notation) is another text-based serialization format. Its MIME type is `application/json`. JSON is defined in RFC 4627 at http://www.ietf.org/rfc/rfc4627.txt.

Note: Versions of PHP older than 5.2.0 may not come with built-in support for JSON. If you are running an older version of PHP, see http://www.php.net/manual/en/book.json.php for information on installing the JSON extension.

1.5 Authentication

The API requires an authenticated session for most commands. Session state is stored on the server and sessions are tracked using HTTP cookies.

The API commands for creating and deleting sessions are described in the reference section.

1.6 Special Values in URIs

CanIt-Domain-PRO treats the special value `@@` specially in three places:

1. If you supply a realm name of `@@` in the URI, the API *usually* uses the realm of the user who has authenticated to the API server. However, for the site administrator *only*, the special value `@@` in connection with Address Mappings, Authentication Mappings, Domain Mappings and Verification Servers makes the API use the realm as determined by doing a Realm Mapping lookup on the domain or address in question.
2. If you supply a stream name of `@@` in the URI, the API uses the home stream of the user who authenticated to the API server.
3. If you supply a stream name of `*` in the URI and are the CanIt-Domain-PRO administrator or a realm administrator, then *some* API functions return values for all streams in the realm. Such functions are said to *permit the wildcard stream*. Functions that do not permit the wildcard stream return an error if it is used.
4. If you supply a user name of `@@` in the URI, the API uses the user-name of the authenticated user.

CanIt-Domain-PRO treats the special value `*` specially if it is encountered where a realm name is expected. For the six cases:

```
GET /api/2.0/realm/*/address_mappings
GET /api/2.0/realm/*/authentication_mappings
GET /api/2.0/realm/*/domain_mappings
GET /api/2.0/realm/*/domains_seen
GET /api/2.0/realm/*/users
GET /api/2.0/realm/*/verification_servers
```

CanIt-Domain-PRO retrieves all of the items in the authenticated user's realm *and* all sub-realms. For the site administrator, this boils down to a site-wide listing of all items in all realms.

An API call that does not permit the use of the wildcard realm returns an error if it is used.

1.7 Response Codes

Each HTTP request to the REST API returns a HTTP response, with an appropriate code and message, and, depending on the request, some response data.

For more information, see the HTTP Specification

1.7.1 Success Codes

200 OK

The request was successful. In this API, this code is usually returned in response to a successful GET, with the desired information in the body of the response.

201 Created

The request succeeded in creating the desired resource. The Location: parameter contains the canonical location of the new resource.

204 No Content

The request succeeded, but the server does not wish to return any new content.

1.7.2 Client Failure Codes

400 Bad Request

The request failed, due to invalid data provided by the client. The body of the request may contain further information, either in machine-readable (see below) or human-readable format. The client should **not** repeat the request without modifications

400 Bad Request (bad data)

The request failed, due to invalid data provided by the client. The body of the request contains details on which data fields were invalid, and why. This body data is a hash of the missing or invalid fields, encoded in the serialization format preferred by the client (YAML by default).

Note: Unfortunately, as this usage extends HTTP, there is no way to distinguish this more verbose 400 error from a standard 400 error by numeric code alone.

403 Forbidden

The client is forbidden to access that resource.

404 Not Found

No resource was found for that URI.

405 Method Not Allowed

The client used a method (GET, POST, PUT, etc) not permitted for the resource identified by that URI.

1.7.3 Server Failure Codes

500 Server Error

An unexpected problem occurred on the server side.

Chapter 2

API Reference

This section provides a full reference for all API functionality available to integrators. Note that where we show JSON examples, we have reformatted the JSON for readability. The JSON actually produced by the API is typically all on one line with no extraneous spaces for readability.

2.1 Authenticating to the API

2.1.1 Logging in

To authenticate to the API server:

`POST /api/2.0/login`

with POST variables **`user`** and **`password`** containing the username and password.

A successful login will return **204 No Content**, with the header containing a cookie value to be used for subsequent requests.

2.1.2 Logging out

To log out of an API session:

`POST /api/2.0/logout`

with an empty body. The session associated with the cookie provided will be destroyed, and that cookie will no longer be usable.

Properly logging out of an API session, while not completely necessary, is highly recommended.

2.1.3 Switching Users

Note: This API call is available only to realm administrators.

Once logged in to the API, you can switch to another user. This is useful, for example, if you wish to perform actions using the API on behalf of another user but you wish to have CanIt-Domain-PRO enforce permissions so that your API calls are unable to do anything the target user would not be allowed to do.

Note: Once you switch users, there is no going back. You have to log out of the API and re-login to become the original user. Note also that if the user-ID you switch into lacks API permission, you won't be able to do anything after switching to that user!

To switch users:

`POST /api/2.0/switch_user`

The POST body should contain:

- **`su_userid`** — the user-ID you wish to become.
- **`su_stream`** — (optional) the stream to consider your "home stream" after switching users.
- **`su_realm`** — (optional) the realm in which the specified user exists. A realm administrator can become a user in any of the realms in the tree rooted at his or her realm. If you omit the realm, CanIt-Domain-PRO assumes the current realm. In particular, it does *not* attempt to deduce a realm based on the user-ID.

If the **`switch_user`** call succeeds, it returns a "204 No Content" result. If it fails, it returns an appropriate error code.

If the original logged-in user did not have write-access, then CanIt-Domain-PRO remains in read-only mode regardless of which user the original user switches to.

2.2 Basic Information

The following API call returns basic information about the CanIt installation:

`GET /api/2.0/info`

It returns a serialized hash reference containing:

product_name The name of the product (CanIt-PRO or CanIt-Domain-PRO)

product_version The version number of the product.

has_realms Returns 1 for CanIt-Domain-PRO and 0 for CanIt-PRO.

2.3 Time Zones

The following API call returns the list of time zones that CanIt-Domain-PRO supports:

`GET /api/2.0/time_zones`

The returned value is a hash whose keys are the internal time zone names and corresponding values are a slightly more readable description of the time zone. If you want to set a stream's time zone, then you need to set the stream setting **`TimeZone`** to be one of the hash keys returned by the **`GET /api/2.0/time_zones`** call. See Section 2.20.11.

2.4 Address Mappings

2.4.1 Retrieving an Address Mapping

To retrieve a single address mapping:

`GET /api/2.0/realm/`*`some_realm`*`/address_mapping/`*`addr@example.com`*

This will return a serialized hash reference containing:

address The address itself.

stream The name of the stream for this address.

realm The name of the realm for this address.

cached 1 or 0, indicating whether or not this address mapping is a cached entry or a manually-entered one.

cache_date Creation time of cached entry as number of seconds from the UNIX epoch.

In JSON format, it should be similar to:

```
{"address": "addr@example.com",
 "stream": "some_stream",
 "cached": 0,
 "cache_date": 0,
 "realm": base}
```

2.4.2 Listing Address Mappings

To list all address mappings:

`GET /api/2.0/realm/`*`some_realm`*`/address_mappings`

This will produce a list of serialized hashes. Each hash will contain the same information as an individual GET (see above).

In JSON format, it should be similar to:

```
[
  {"address": "user1@example.com",
   "stream": "user1",
   "cached": 0,
   "cache_date": 0,
   "realm": "base},
  {"address": "user2@example.com",
   "stream": "user2",
   "cached": 1,
   "cache_date": 1185221865,
    "realm": base}
  ]
```

2.4.3 Creating an Address Mapping

To create a new address mapping:

`PUT /api/2.0/realm/`*`some_realm`*`/address_mapping/`*`addr@example.com`*

The body of the PUT request should be a serialized hash containing:

stream The name of the stream for this address.

cached (optional) 1 or 0, indicating whether or not this address mapping is a cached entry or a manually-entered one. If absent, 0 is assumed.

2.4.4 Deleting an Address Mapping

To delete an address mapping:

`DELETE /api/2.0/realm/`*`some_realm`*`/address_mapping/`*`addr@example.com`*

2.5 Running Address-to-Stream Mapping

To see which stream an address will map to:

`GET /api/2.0/address_to_stream/`*`addr@example.com`*

This will return the name of the stream to which *addr@example.com* will be streamed. Note that this may involve running an LDAP lookup, etc. depending on how streaming is configured.

The return value will be a serialized hash containing:

stream The name of the stream.

realm The name of the realm.

Note that this API call may return an error if (for example) a back-end LDAP server is non-responsive.

Some errors (such as a nonexistent address) result in a successful return, but the returned hash contains only an **error** key whose value contains an error message. Be sure to check the return value of this call before using it!

This API call is available only to realm administrators. If an address maps to a realm that the calling user does not own, this call returns a Permission Denied error.

2.6 Authentication Mappings

2.6.1 Retrieving an Authentication Mapping

To retrieve an authentication mapping for a domain:

`GET /api/2.0/realm/`*`some_realm`*`/auth_mapping/`*`example.com`*

This will return a serialized hash reference containing:

domain The domain being mapped.

key The key for the mapping method used.

comment The comment attached to the user-lookup method.

realm The realm for this domain and method.

lookup_realm The realm in which the User Lookup Settings (if any) exist

In JSON format, it should be similar to:

```
{"domain": "example.com",
 "key": "IMAP",
 "comment": "Some comment was added here"}
```

2.6.2 Listing Authentication Mappings

To list all authentication mappings:

`GET /api/2.0/realm/`*`some_realm`*`/auth_mappings`

This will produce a list of serialized hashes. Each hash will contain the same information as an individual GET (see above).

In JSON format, it should be similar to:

```
[
  {"domain": "example.com",
    "key": "IMAP",
    "comment": "A comment"},
  {"domain": "otherexample.com",
    "key": "MyLDAPMappingMethod",
    "comment": "Another comment"}
]
```

2.6.3 Creating an Authentication Mapping

To create an authentication mapping:

PUT /api/2.0/realm/*some_realm*/auth_mapping/*example.com*

The body of the PUT request should be a serialized hash containing:

key The key identifying a valid authentication mapping method.

lookup_realm (Optional) The realm in which the associated User Lookup exists. The realm must be an ancestor of the current realm and the User Lookup must be marked "Inheritable". If this parameter is omitted, the lookup realm defaults to the current realm.

If the specified **key** is not found in the lookup realm, the API call fails.

2.6.4 Deleting an Authentication Mapping

To delete an authentication mapping:

DELETE /api/2.0/realm/*some_realm*/auth_mapping/*example.com*

2.7 Domain Mappings

2.7.1 Retrieving a Domain Mapping

To retrieve a domain mapping:

GET /api/2.0/realm/*some_realm*/domain_mapping/*example.com*

This will return a serialized hash reference containing:

domain The domain being mapped.

key The key for the mapping method used.

realm The realm for this domain and method.

lookup_realm The realm in which the User Lookup Settings (if any) exist

In JSON

```
{"domain": "example.com",
 "key": "Program"}
```

2.7.2 Listing Domain Mappings

To list all domain mappings:

GET /api/2.0/realm/*some_realm*/domain_mappings

This will produce a list of serialized hashes. Each hash will contain the same information as an individual GET (see above).

In JSON format, it should be similar to:

```
[{"domain": "example.com",
  "key": "Program"},
 {"domain": "otherexample.com",
  "key": "MyLDAPMappingMethod"}]
```

2.7.3 Creating a Domain Mapping

To create a domain mapping:

`PUT /api/2.0/realm/`*`some_realm`*`/domain_mapping/`*`example.com`*

The body of the PUT request should be a serialized hash containing:

key The key identifying a valid domain mapping method.

lookup_realm (Optional) The realm in which the associated User Lookup exists. The realm must be an ancestor of the current realm and the User Lookup must be marked “Inheritable”. If this parameter is omitted, the lookup realm defaults to the current realm.

If the specified **key** is not found in the lookup realm, the API call fails.

2.7.4 Deleting a Domain Mapping

To delete a domain mapping:

`DELETE /api/2.0/realm/`*`some_realm`*`/domain_mapping/`*`example.com`*

2.8 Known Networks

Note: The Known Networks API function is available only to the CanIt-Domain-PRO site administrator.

2.8.1 Listing Known Networks

To retrieve a list of Known Networks settings:

`GET /api/2.0/known_networks`

The return value is an array of hashes. Each hash has a **cidr** key containing the network CIDR (or the literal string **SMTP-AUTH**). Each hash also contains a number of key/value pairs indicating the Known Networks settings.

The **domains** entry in each Known Networks hash is an array of domains associated with the Known Network. Each entry in the **domains** array is a hash that contains two key/value pairs: **domain** whose value is the domain name, and **force_to_stream** whose value is the domain-specific “Force To Stream” value. If there is no domain-specific “Force To Stream” then the value of the **force_to_stream** key is the empty string.

If no domains are associated with the network, the **domains** array is an empty array.

2.8.2 Retrieving a Known Network entry

To retrieve a particular Known Network entry:

`GET /api/2.0/known_network/`*`network/mask`*

In the URL, *network* is the network part such as 127.0.0.1 or ::1. The */mask* is optional and specifies the netmask (for example, /32 or /128.)

2.8.3 Creating or Updating a Known Network entry

To create a Known Network entry:

`PUT /api/2.0/known_network/`*`network/mask`*

To update an entry, use POST instead of PUT. The only difference between the two is that PUT will return an error if the specified network already exists.

The body of the PUT or POST should contain zero or more of the following parameters:

- **skip_rbl** with a value of 0 or 1. Corresponds to "Skip RBL Lookups" in the Known Networks page.
- **skip_spam_scan** with a value of 0 or 1. Corresponds to "Skip Spam Scan"
- **skip_virus_scan** with a value of 0 or 1. Corresponds to "Skip Virus Scan"
- **skip_ext** with a value of 0 or 1. Corresponds to "Skip Extension Rules"
- **skip_mime** with a value of 0 or 1. Corresponds to "Skip MIME-Type Rules"
- **skip_size** with a value of 0 or 1. Corresponds to "Skip Size Limit Checks"
- **skip_blacklist** with a value of 0 or 1. Corresponds to "Prohibit Block Rules"
- **skip_greylist** with a value of 0 or 1. Corresponds to "Skip Greylisting"
- **skip_spf** with a value of 0 or 1. Corresponds to "Skip SPF Checks"
- **skip_delay** with a value of 0 or 1. Corresponds to "Skip Delay Rules"
- **skip_strip** with a value of 0 or 1. Corresponds to "Skip Attachment Stripping"
- **skip_dictionary_attacks** with a value of 0 or 1. Corresponds to "Omit from Dictionary Attack Detection"
- **hold_locally** with a value of 0 or 1. Corresponds to "Friendly Host"
- **parse_received** with a value of 0 or 1. Corresponds to "Parse Received Headers"
- **auto_whitelist** with a value of 0 or 1. Corresponds to "Auto-Allow Recipients"
- **allow_relaying** with a value of 0 or 1. Corresponds to "Allow Relaying". Available only on CanIt-Domain-PRO appliances or Red Hat systems with the appliance RPMs installed.

- **relay_unlisted_domains** with a value of 0 or 1. Corresponds to "Relay Unlisted Domains"
- **rcpt_hourly_limit** with an integer value. Corresponds to "Per-Sender Recipient Rate Limit"
- **ip_rcpt_hourly_limit** with an integer value. Corresponds to "Per-IP Recipient Rate Limit"
- **force_to_stream** with a string value. Corresponds to "Force To Stream"
- **comment** with a string value. Corresponds to "Comment"

2.8.4 Deleting a Known Network entry

To delete a Known Network entry:

`DELETE /api/2.0/known_network/`*`network`*`/`*`mask`*

The system will not allow you to delete the entries for 127.0.0.1/32 or ::1/128.

2.8.5 Associating a Domain with a Known Network

To associate a domain with a Known Network, use:

`POST /api/2.0/known_network/`*`network`*`/`*`mask`*`/domain`

with the POST variable **`domain`** containing the domain to add to the network. You may optionally supply a **`force_to_stream`** POST variable specifying a domain-specific "Force To Stream" value.

2.8.6 Removing an Associated Domain from a Known Network

To remove an associated domain from a Known Network, use:

`DELETE /api/2.0/known_network/`*`network`*`/`*`mask`*`/domain/`*`domain_name`*

where *domain_name* is the domain you no longer wish to associate with *network/mask*.

2.9 System Check

Note: The System Check API function is available only to the CanIt-Domain-PRO site administrator.

To retrieve a list of System Check results:

`GET /api/2.0/system_check`

The return value is an array of hashes. Each hash has the following members:

- **hostname** — the host on which the check was performed.
- **test_name** — the name of the test.
- **test_ok** — 0 if the test failed or 1 if it passed.
- **when_checked** — the time (as a UNIX timestamp) when the check last ran.

- **message** — a human-readable message describing the test.

2.10 Domain Recipient Verification

The following API call returns a list of domains known to CanIt-Domain-PRO with some extra information about each domain:

`GET /api/2.0/domain_recipient_verification`

Note: The domain_recipient_verification API call is available only to the CanIt-Domain-PRO site administrator.

This call returns an array of hashes. Each hash has the following entries:

- **domain** — the name of a domain.
- **rejects_bad** — 1 if the domain correctly rejects invalid recipients; 0 otherwise.
- **mx_points_here** — 1 if CanIt-Domain-PRO believes the domain's MX records to point to the CanIt-Domain-PRO cluster; 0 otherwise.
- **last_checked** — the time (as a UNIX timestamp) when the **rejects_bad** and **mx_points_here** values were last checked.

2.11 Anomalies

Note: The System Check API function is available only to CanIt-Domain-PRO users with **root** permission.

To retrieve a list of anomalies:

`GET /api/2.0/realm/`*`some_realm`*`/anomalies`

The return value is an array of hashes. Each hash has the following members:

- **family** — the general family of the anomaly.
- **detail** — specific details such as domain name.
- **message** — a human-readable explanation of the message
- **date** — the time (as a UNIX timestamp) when the anomaly last occurred.
- **hostname** — the name of the host on which the anomaly occurred.
- **count** — how many times the anomaly has occurred.
- **realm** — the realm affected by the anomaly.

The **`anomalies`** API call returns all anomalies for the specified realm *and* the subtree rooted at that realm.

2.12 Domain Routing

<u>**Note:**</u> The Domain Routing API functions are available only on appliance installations. These include our Debian-based appliances and RPM-based distributions that were installed with the **`--appliance-rpms`** flag.

2.12.1 Retrieving a Domain Route

To retrieve a single domain route:

`GET /api/2.0/domain_route/`*`example.com`*

This will return a serialized hash reference containing:

domain String containing the domain

port TCP port to which mail will be routed

server_list Serialized array reference containing one or more server names or IP addresses

treat_as_mx 0 if the server list should be treated as A records; 1 if they should be treated as MX records.

queue_threshold The number of queued messages required to elicit a warning about queued mail.

msg_age_threshold The age (in hours) a queued message must reach to elicit a warning about queued mail.

notify_address The email address to which queued-mail warnings are sent.

In JSON format, it should be similar to:

```
{"domain": "example.com",
 "port": 25,
 "server_list": ["192.168.10.1", "192.168.10.2"]}
```

2.12.2 Listing Domain Routes

To list all domain routes:

`GET /api/2.0/domain_routes`

This will produce a list of serialized hashes. Each hash will contain the same information as an individual GET (see above).

In JSON format, it should be similar to:

```
[{"domain": "example.com",
  "port": 25,
     "server_list": ["192.168.10.1", "192.168.10.2"]},
    {"domain": "otherexample.com",
     "port": 2525,
     "server_list": ["10.0.0.1", "10.0.5.122"]}]
```

2.12.3 Adding a domain route

To add a domain route:

PUT /api/2.0/domain_route/*example.com*

with a body (in JSON) of:

```
{"server_list": ["server1.example.com", "server2.example.com"]}
```

If there's only one server, it still needs to be in list form. The body of the PUT may contain the following keys in addition to the required **server_list**:

- **treat_as_mx** (boolean) If true, the server name(s) will be treated as MX records and delivery will take place to the designated MX hosts.
- **port** (integer) Tells CanIt-Domain-PRO to route to a non-standard port instead of the usual port 25. Only the CanIt-Domain-PRO site administrator can specify non-standard ports below 1024.
- **queue_threshold** The number of queued messages required to elicit a warning about queued mail.
- **msg_age_threshold** The age (in hours) a queued message must reach to elicit a warning about queued mail.
- **notify_address** The email address to which queued-mail warnings are sent.

2.12.4 Updating a Domain Route

To update a domain route:

POST /api/2.0/domain_route/*example.com*

The POST form data must contain the key **server_list**, which must have as a value a string containing a comma-separated list of servers to use. It may contain the **treat_as_mx** and **port** keys described in the previous section.

2.12.5 Activating domain routes

To activate all domain routes (this means to generate Sendmail's "mailertable" and "access" file on all cluster members):

PUT /api/2.0/domain_routes/activate

with a JSON body of

```
{"value": 1}
```

This should be done after you're finished adding or removing domain routes, rather than for each individual domain route entry.

2.12.6 Deleting a Domain Route

To delete a domain route:

`DELETE /api/2.0/domain_route/`*`example.com`*

2.13 DKIM Key Pairs

CanIt-Domain-PRO has a set of API calls for managing DKIM key pairs used to DKIM-sign outbound mail.

Note: Only realm administrators can use the DKIM-related API calls.

2.13.1 Listing All DKIM Keys

To list all DKIM keys associated with the domains in a realm, use:

`GET /api/2.0/realm/`*`some_realm`*`/dkim_keys`

This will return an array, each of whose elements is a hash containing the following keys:

domain The domain name to which the DKIM key pair applies.

selector The DKIM selector, currently hard-coded at **`canit`**.

public_key The Base-64-encoded public key.

active 1 if the key is active; 0 if it is not.

Note that CanIt-Domain-PRO *never* reveals the DKIM private key.

2.13.2 Retrieving a DKIM Key Associated with a Specific Domain

To retrieve a specific DKIM key, use:

`GET /api/2.0/realm/`*`some_realm`*`/dkim_key/`*`domain`*`/`*`selector`*

where *domain* is the name of the domain whose key you wish to retrieve and *selector* is the selector. If you omit *selector*, a default selector of **`canit`** is used.

The return value is a hash with the same **domain**, **selector**, **active** and **public_key** elements as described previously.

2.13.3 Creating a DKIM Key Pair

To generate a DKIM key pair for a domain, use either:

`POST /api/2.0/realm/`*`some_realm`*`/dkim_key/`*`domain`*`/`*`selector`*

or

`PUT /api/2.0/realm/`*`some_realm`*`/dkim_key/`*`domain`*`/`*`selector`*

with a POST/PUT variable **value** set to 1. This will cause CanIt-Domain-PRO to create a DKIM key pair for the domain. The difference between POST and PUT is that if a domain already has a DKIM key pair, POST will not change it and will return success, whereas PUT will return an error. Note that a newly-created DKIM key-pair is NOT active. Also, if you omit *selector*, it defaults to **`canit`**.

2.13.4 Activating a DKIM Key Pair

To activate a DKIM key pair, use:

`POST /api/2.0/realm/`*`some_realm`*`/dkim_key/`*`domain`*`/`*`selector`*`/enable`

Note that you *must* include the *selector*. Activating a DKIM key pair automatically disables all other key pairs (if any) for the domain. You also *must* include a POST variable **`value`** set to 1.

2.13.5 Deactivating a DKIM Key Pair

To deactivate a DKIM key pair, use:

`POST /api/2.0/realm/`*`some_realm`*`/dkim_key/`*`domain`*`/`*`selector`*`/disable`

Note that you *must* include the *selector*. You also *must* include a POST variable **`value`** set to 1.

2.13.6 Deleting a DKIM Key Pair

To delete a DKIM key pair for a domain, use:

`DELETE /api/2.0/realm/`*`some_realm`*`/dkim_key/`*`domain`*`/`*`selector`*

2.14 Incidents

2.14.1 Retrieving an Incident

To retrieve a single incident:

`GET /api/2.0/incident/`*`incident_id`*

This will return a serialized hash reference containing all of the incident's metadata, as follows:

incident_id Incident ID number

unixtime UNIX timestamp for incident creation time

auto_action_date Date for any automatic actions

incident_score Total score for this message

incident_bayes Bayes probability, between 0 and 1

msghash Hash for locating held message

hold_reason Reason for incident creation

status Current status of the incident

resolution Current resolution of the incident.

frozen 1 if incident frozen, 0 if not.

subject Subject of message creating this incident.

header_from The email address in the From: header of the message.

resolved_by User ID of the user resolving this incident (if resolved)

stream Stream ID.

recipients An array of recipients.

hosts An array of sending hosts. Each host is a serialized hash reference containing the following keys:

> **num_transmissions** The number of times the host retried the incident. Note that recent versions of CanIt-Domain-PRO stop incrementing this counter once it reaches 11, so a value of 11 really means "11 or more transmissions."
>
> **ip_address** The IP address of the host.
>
> **host_name** The host name.
>
> **sender** The envelope sender address.
>
> **country_code** The country in which the host is located.
>
> **region** The region (state, province, etc.) in which the host is located.
>
> **city** The city in which the host is located.
>
> **latitude** The approximate latitude of the host's location.
>
> **longitude** The approximate longitude of the host's location.
>
> Note that the location entries are approximate. They may also be null if CanIt-Domain-PRO could not determine the location of the host.

realm Realm name.

incident_report The full incident report for this incident.

cookie Unique random value used for unauthenticated incident resolution.

urls An array of all URLs found in the incident. ***Note:*** This information is available only on installations of CanIt-Domain-PRO where Storage Manager is enabled.

In JSON format, it should be similar to:

```
{"incident_id": "12345ba76d",
 "unixtime": 1200518218,
 "auto_action_date": null,
 "incident_score": 6.4,
 "incident_bayes": 0.75,
 "msghash": "5abbc74d2ed2d6dcf8e5763d6d29a517e3edb9f0",
 "hold_reason": "SpamScore",
 "status": "spam",
 "resolution": "reject",
 "frozen": 1,
 "subject": "Re: user@example.com - don't miss this",
 "header_from": "someone@example.org",
 "resolved_by": "dmo",
 "stream": "base:default",
 "incident_report": "lots-of-text-here",
 "cookie": 23412334,
 "recipients": ["user1@example.org", "user2@example.org"],
 "hosts": [{"num_transmissions": 1,
           "ip_address": "192.168.10.1",
           "sender": "someone@example.org",
           "host_name": "relay.example.org",
           "country_code": "CA",
           "region": "Quebec",
           "city": "Montreal",
           "latitude": 45.5,
           "longitude": -73.5833}]}
```

2.14.2 Retrieving the Message Associated with an Incident

To retrieve the message associated with an incident, use one of the following calls:

GET /api/2.0/incident/*incident_id*/message
GET /api/2.0/incident/*incident_id*/message/raw

The first version returns a hash containing a single element named **message**; the value of this element is the entire raw MIME message as a JSON or YAML string.

If you use the **raw** modifier, then on success the API returns the message as **message/rfc822** data and *not* YAML or JSON. The API client libraries supplied by AppRiver handle this special case and convert the returned data into a hash with the single element **message** containing the entire MIME message.

2.14.3 Updating an Incident

To update an incident's status:

`POST /api/2.0/incident/`*`incident_id`*

The POST body may contain:

frozen 1 or 0, to freeze or unfreeze the incident

status The new status for the incident. Valid values are **spam, not-spam** and **pending**. If status is provided, resolution must also be provided.

resolution The new resolution for the incident. Valid values are **allow, allow-domain, allow-host, allow-sender, blacklist-domain, blacklist-host, blacklist-sender, auto-reject, discard, pending** and **reject**. If resolution is provided, status must also be provided.

2.14.4 Listing Incidents in a Stream

To list all incidents in a stream:

`GET /api/2.0/realm/`*`some_realm`*`/stream/`*`stream_name`*`/incidents/`*`status`*

The returned incidents may be limited by status (any, spam, not-spam, or pending) by specifying the status at the end of the URI. (If you don't specify the status, it defaults to "any".)

This will produce a list of serialized hashes. Each hash will contain the same information returned by a GET of a single incident.

You can limit the number of incidents returned by specifying a count after *status*. (If you do not specify such a limit, then the count defaults to 100.)

Finally, you can specify an offset for the beginning of results by specifying an offset after the count. (If you do not specify an offset, then it defaults to zero.)

For example, the following call will return the most recent 20 spam incidents:

`GET /api/2.0/realm/`*`some_realm`*`/stream/`*`stream_name`*`/incidents/spam/20`

and this will return the next most recent 10 spam incidents (21 through 30):

`GET /api/2.0/realm/`*`some_realm`*`/stream/`*`stream_name`*`/incidents/spam/10/20`

Note that the offset is zero-based. The most general form of the URL is:

`GET /api/2.0/realm/`*`some_realm`*`/stream/`*`stream_name`*`/incidents/`*`status`*`/`*`count`*`/`*`offset`*

Filtering the List of Incidents

The **`GET /api/2.0/realm/`*`some_realm`*`/stream/`*`stream_name`*`/incidents`** permits optional additional GET parameters to filter the returned list of incidents. The optional parameters are described below. Note that all matches except for the numerical and date parameters are case-insensitive sub-string matches.

- **from**: Limit the returned list to those incidents with a From: header address containing the specified value.

- **minbayes**: This parameter should range from 0 to 100, and limits the list to those incidents scoring at least *minbayes* percent in Bayes analysis.
- **maxbayes**: This parameter should range from 0 to 100, and limits the list to those incidents scoring at most *maxbayes* percent in Bayes analysis.
- **minscore**: This parameter should be numeric and limits the list to those incidents scoring at least *minscore* points.
- **maxscore**: This parameter should be numeric and limits the list to those incidents scoring at most *maxscore* points.
- **mindate**: This parameter should be a valid date in the format YYYY-MM-DD. It limits the list to those incidents created on or after *mindate*.
- **maxdate**: This parameter should be a valid date in the format YYYY-MM-DD. It limits the list to those incidents created on or before *maxdate*.
- **reason**: Limit the returned list to those incidents whose Hold Reason contains *reason*.
- **relay_ip**: Limit the returned list to those incidents whose relay IP contains *relay_ip*.
- **report**: Limit the returned list to those incidents whose Spam Analysis Report contains *report*.
- **subject**: Limit the returned list to those incidents whose subject contains *subject*.
- **to**: Limit the returned list to those incidents sent to a recipient whose email address matches *to*.

The following example gets the first 50 pending incidents whose subject matches "viagra" (case-insensitively) that scored at most 8 points:

```
GET /api/2.0/realm/some_realm/stream/stream_name/incidents/pending/50?subject=viagra&maxscore=8
```

2.15 Voting

To vote something as spam or non-spam:

```
POST /api/2.0/vote
```

with the following POST variables:

- `i` – the ID to vote on. This is the same as the "i" parameter in the Bayes training link.
- `m` – the Magic value. This is the same as the "m" parameter in the Bayes training link.
- `c` – the actual vote, which must be one of "s" for spam, "n" for non-spam or "f" to forget the training. This is the same as the "c" parameter in the Bayes training link.
- `t` – the Storage Manager token. This parameter is required only if you are using Storage Manager. It is the same as the "t" parameter in the Bayes training link.

Note: Even if you have enabled unauthenticated voting, the API does *not* permit unauthenticated voting. To vote using the API, you must first log in. Additionally, the user you log in as must have permission to vote in the relevant stream.

2.16 Phishing URLs

The Phishing URL API calls allow you to manipulate the list of known phishing URLs and the phishing URL votes. Note that all API calls in this section are available *only* to **base** realm administrative accounts. That is, the API user must be in the **base** realm and have "root" privileges.

2.16.1 Normalized URLs

CanIt-Domain-PRO normalizes URLs before processing them as phishing URLs. The normalization involves removing the leading `https:` or `http:` schema and making other small adjustments to the URL to canonicalize it.

2.16.2 Listing known Phishing URLs

To obtain a list of known phishing URLs, use the following API call:

`GET /api/2.0/phishing_urls`

This API call returns an array of elements. Each element of the array is a hash containing the following elements:

url The URL itself, in normalized form.

source The source of the URL. URLs added locally by the CanIt-Domain-PRO administrator have a source of `local` while URLs supplied by AppRiver via RPTN have a source that starts with `RPTN:`.

expiry The expiry date (if any) of the URL. This is a standard UNIX timestamp (which is an integer specifying the number of seconds since 1 January 1970 UTC.) URLs with no expiry date have `null` as the value of this element.

is_phish Set to 1 if the URL is malicious or 0 if it is benign.

You may supply the following optional query parameters to the GET call:

url_filter A string; only normalized URLs containing this string will be returned.

source_filter A string; only entries whose source field contains this string will be returned. For example, specifying `local` returns only locally-entered URLs where specifying `RPTN:` returns only those URLs distributed by AppRiver.

count Limit the number of returned URLs to this integer value. If this parameter is not supplied, then all the URLs are returned.

offset Start returning URLs from this offset. For example, the first 100 URLs could be obtained with a count of 100 and an offset of 0. The next 100 would have a count of 100 and an offset of 100.

2.16.3 Manipulating Individual Phishing URL Entries

Note: In all of the following API calls, the URL is normalized prior to being used.

Retrieving a Phishing URL

To retrieve a specific phishing URL entry, use:

`GET /api/2.0/phishing_url/`*`b64_encoded_url`*

In this api call, **`b64_encoded_url`** is the normalized URL encoded using Base-64 encoding. For example, if the URL you want is **`www.example.com/subdir?x=1`**, then the API call is:

`GET /api/2.0/phishing_url/d3d3LmV4YW1wbGUuY29tL3N1YmRpcj94PTE=`

If the call succeeds, it returns a hash containing the elements described in Section 2.16.2.

Deleting a Phishing URL

To delete a specific phishing URL entry, use:

`DELETE /api/2.0/phishing_url/`*`b64_encoded_url`*

where **`b64_encoded_url`** is once again the normalized URL encoded using Base-64 encoding.

Creating or Updating a Phishing URL

To create or update a specific entry, use:

`POST /api/2.0/phishing_url/`*`b64_encoded_url`*

with the following POST parameters (all optional):

is_phish 1 for a malicious URL; 0 for a benign one.

expiry The expiry time of the entry as a UNIX time stamp. If the expiry time is not supplied, it defaults to 120 days from the current time.

source The source of the entry, which defaults to **`local`**. You cannot supply a value for source that starts with **`RPTN:`** in any combination of upper- or lower-case.

If the phishing URL already exists, the existing entry will be updated. If it does not exist, a new entry will be created.

2.16.4 Listing Phishing URL Votes

To obtain a list of Phishing URL votes, use:

`GET /api/2.0/phishing_url_votes`

This API call returns an array of elements. Each element is a hash containing the following entries:

url The normalized URL.

source Always set to **local** for votes submitted via the CanIt-Domain-PRO web interface.

last_vote The UNIX timestamp of the last vote.

num_votes The number of times this URL has been voted malicious.

decided Set to **`true`** if the URL is in the Known Phishing URL list; false otherwise.

is_phish Set to 0 if the URL is in the Known Phishing URL list and marked as benign. Set to 1 if the URL is marked malicious. And set to **`null`** if the URL is not in the Known Phishing URL list at all.

expiry If the URL is in the Known Phishing URL list, this is the UNIX timestamp of that entry's expiry (if any).

2.17 Realms

2.17.1 Creating a realm

To create a realm, PUT to that realm's name. Only the site administrator may use this API call:

`PUT /api/2.0/realm/`*`some_realm`*

The body of the PUT should contain (serialized appropriately) a key of ”description” with a text value describing the new realm. In JSON format, this would look like:

```
{"description": "Example.com Realm"}
```

In addition to the ”description” key, you can optionally include ”parent”, which is the name of the parent realm; ”expiry”, which must be a valid date in the form YYYY-MM-DD; and ”userfield1” through ”userfield4”.

2.17.2 Updating an Existing Realm

To modify an existing realm:

`POST /api/2.0/realm/`*`some_realm`*

Only the site administrator may use this API call.

The POST body may contain:

description A new description for that realm

parent The name of the realm's parent

expiry An expiry date in the form YYYY-MM-DD

userfield1 Arbitrary data

userfield2 Arbitrary data

userfield3 Arbitrary data

userfield4 Arbitrary data

2.17.3 Renaming a Realm

Note: Renaming a realm can take a long time. It adjusts many database tables and can fail with a constraint violation if something else also adjusts a table while you are renaming a realm. If the rename API call fails, we recommend trying it two or three more times just in case it fails because of a race condition with the filter. As much as possible, you should avoid renaming realms; this API call is intended only for occasional corrections to your realm-naming policy.

Renaming a realm adjusts only the database tables. It does not adjust on-disk files, meaning that if you rename a realm, *all Bayes training and archived mail for that realm will be lost.*

To rename a realm:

`POST /api/2.0/realm/`*`some_realm`*`/rename`

The POST body must contain:

new_name The new name of the realm.

The POST body may contain:

force_all_tables Either 1 to force all database tables containing the realm to be renamed, or 0 if only the main tables should be renamed. The default is 0. *Only the CanIt-Domain-PRO site administrator can set this parameter to 1.* If **force_all_tables** is not set to 1, then renaming a realm does *not* update the log-index tables (for those installations with the log-indexing component installed) or the statistics tables. The reason is that updating those tables can be very time-consuming and can lock the database for a long time.

If you rename a realm without having **force_all_tables** set to 1, then you lose the statistics for that realm. Additionally, the realm administrator will be unable to access the realm's mail logs created prior to the renaming and the site administrator will need to search the logs using the old realm name to find logs prior to the renaming.

There are several restrictions to this API call:

- Only a realm administrator may rename a realm.

- The new realm name must not exist.
- The old realm must exist and must be owned by the calling user.
- You cannot rename a realm to or from **base**.
- The new realm name must be a legal realm name; if any illegal characters are found in the name, the API call fails.

2.17.4 Listing realms

`GET /api/2.0/realms`

This will produce a list of hashes (serialized appropriately) containing each realm, description, expiry and user-fields. An additional field **path** contains an array of realm names giving the complete path from the given realm following parent realms up to **base**. An example in JSON format:

```
[{"description": The Base Realm,
"parent": null,
"path": ["base"],
"expiry": null,
"userfield1": null,
"userfield2": null,
"userfield3": null,
"userfield4": null,
"name": "base"},
{"description": "Example.com Realm",
"parent": base,
"path": ["example", "base"],
"expiry": "2029-12-31",
"userfield1": "Secret internal data",
"userfield2": null,
"userfield3": null,
"userfield4": null,
"name": "example"}]
```

2.17.5 Retrieving a Subtree

To retrieve a list of realms rooted at a given realm:

`GET /api/2.0/realm/`*`realmname`*`/subtree`

This call returns a list of hashes just like **`GET /api/2.0/realms`** except that only the subtree of realms rooted at *realmname* is returned.

2.17.6 Retrieving a Single Realm

To retrieve a single realm:

`GET /api/2.0/realm/`*`realmname`*

This call returns a hash for the realm *realmname* that contains the same fields as each hash returned by the **`GET /api/2.0/realms`** API call.

2.17.7 Deleting a realm

To remove a realm, and all things associated with it:

`DELETE /api/2.0/realm/`*`some_realm`*

This will remove **EVERYTHING** associated with that realm – streams, users, settings, etc – so please be sure that any code using this API call does so only after suitable confirmation from the end user.

Deleting a realm is *not* recursive. If a realm has any sub-realms, then those sub-realms are reparented to be under the deleted realm's parent.

2.17.8 Adding a realm mapping

A domain may be mapped to a realm by:

`PUT /api/2.0/realm_mapping/`*`example.com`*

Only the site administrator may use this API call.

The body of the PUT should contain (serialized appropriately) a key of "realm" and a value of the appropriate realm name. In JSON format, this would look like:

```
{"realm": "example"}
```

2.17.9 Modifying a realm mapping

A domain may be mapped from one realm to a different realm by:

`POST /api/2.0/realm_mapping/`*`example.com`*

The site administrator can remap a domain from one realm to any other realm. Realm administrators can only map a domain from an old realm to a new realm if they own both the old and the new realm.

The body of the POST should contain a key of "realm" and a value of the appropriate realm name.

2.17.10 Listing realm mappings

All realm mapping entries can be retrieved with:

`GET /api/2.0/realm_mappings`

This will produce a list of hashes (serialized appropriately) containing domains and the realm they are mapped to. An example in JSON format:

```
[{"domain": "example.com",
"realm": "example}",
{"domain": "example.net",
"realm": "example}",
{"domain": "example.ca",
"realm": "example-canada"}]
```

You can also list only those realm mappings that map to a particular realm:

`GET /api/2.0/realm/`*`some_realm`*`/realm_mappings`

2.17.11 Removing a realm mapping

Realm mappings can be removed with:

`DELETE /api/2.0/realm_mapping/`*`example.com`*

This will remove the entry mapping the domain example.com to a particular realm.

2.17.12 Listing Domains Seen for a Realm

To retrieve a list of all domains seen for all streams within a realm:

`GET /api/2.0/realm/`*`some_realm`*`/domains_seen`

This call returns a list of domain names that have received email within the last 61 days and are mapped to the given realm. Each element of the list contains the following keys:

domain The domain name.

last_seen The date the domain was last seen to receive email.

realm The realm name.

<u>**Note:**</u> CanIt-Domain-PRO will *not* collect a list of domains seen if it has no way to validate them. If none of the domains seen for a realm can be verified (for example, there is no user-lookup or verification server) then CanIt-Domain-PRO will return an empty list.

2.18 Sender/Domain/Network Action Rules

2.18.1 Retrieving a single rule

To retrieve a single rule:

`GET /api/2.0/realm/`*`some_realm`*`/stream/`*`stream_name`*`/rule/`*`type`*`/`*`data`*

In the URL, *type* is one of **`sender`**, **`domain`** or **`network`**, and *data* is data appropriate for the rule, such as **`bob@example.com`**, **`example.org`** or **`192.168.2.15`**.

The API will return a serialized hash reference containing:

type Type of action rule (**network, sender** or **domain**).

value The content to match on – sender address for sender, domain name for domain, and IP network for network

action What to do on a match. (**allow-always, hold-always, hold-if-spam, reject** or **no-rbl**).

expiry The expiry date (if any) for the rule.

who The user ID who created the rule.

message The comment associated with the rule.

stream The stream name.

realm The realm name.

In JSON format, it should be similar to:

```
{"type": "sender",
 "value": "spammer@example.net",
 "action": "reject",
 "expiry": null,
 "who": "admin",
 "message": "A bad spammer",
 "stream": "foo"}
```

2.18.2 Listing Rules

To list all sender, domain, and/or host action rules in a stream:

`GET /api/2.0/realm/`*`some_realm`*`/stream/`*`stream_name`*`/rules`

This will produce a list of serialized hashes. Each hash will contain the same information as an individual GET (see above), with the following additions:

- **uri** — the URI for the rule.
- **hit_count** — the number of hits CanIt-Domain-PRO has seen for the rule. Note that CanIt-Domain-PRO obtains the hit counts from the mail logs; if the log-indexing add-on is not installed, **hit_count** will be zero. Also, the hit count is not updated in real-time; the value may be up to 30 minutes behind real-time.
- **last_hit** — the date the rule was last hit, or the empty string if it has never been hit or the log-indexing add-on is not installed.

2.18.3 Creating a Rule

To create a new action rule in a stream:

`POST /api/2.0/realm/`*`some_realm`*`/stream/`*`stream_name`*`/rule/`*`type`*`/`*`data`*

The body of the POST request must contain:

action What to do on a match. (**allow-always, hold-always, hold-if-spam, reject** or **no-rbl**).

expiry The expiry date of the rule in the format YYYY-MM-DD. This parameter is optional; if omitted, the rule does not expire.

message (Optional.) A comment to attach to the rule.

2.18.4 Deleting a Rule

To delete an action rule:

`DELETE /api/2.0/realm/`*`some_realm`*`/stream/`*`stream_name`*`/rule/`*`type`*`/`*`data`*

2.18.5 Bulk-Creation of Rules

To bulk-create rules:

`POST /api/2.0/realm/`*`some_realm`*`/stream.`*`stream_name`*`/rules`

The body of the POST request must contain:

action One of **allow-always**, **hold-always**, **hold-if-spam** or **reject**.

expiry (Optional) The expiry date of the rule in the form YYYY-MM-DD. If omitted, the rule does not expire.

message (Optional) An optional comment to apply to the rules created.

rules A *string* that is a comma-separated list of email addresses, domain names, IP addresses or CIDR networks.

This API call creates a number of sender, domain and/or network rules. It is substantially faster than calling POST for each individual rule.

On success, an array of hashes is returned. The array order is always the same as the original order of elements in the comma-separated list.

Each hash contains the following elements:

rule The rule whose creation was attempted.

success 1 if the rule was successfully created; 0 otherwise.

If the rule was successfully created, an additional **type** element contains one of **sender**, **domain** or **network** depending on the type of rule created. CanIt-Domain-PRO deduces the type of the rule by inspecting the element:

- Elements that resemble an email address (**`local@domain.tld`**) cause a **sender** rule to be created.

- Elements that resemble an IPv4 or IPv6 address or network cause a **network** rule to be created.
- Elements that resemble a valid domain name, or `@` followed by a valid domain name cause a **domain** rule to be created.
- Unrecognized elements result in an error.

If the rule was not successfully created, an additional **error** element contains an error message explaining the failure.

Note: If invalid parameters are given to the API call, the entire call fails and an error is returned. If valid parameters are given, some attempts to create rules may fail and others may succeed; you must inspect the returned array of hashes to find out which succeeded and which (if any) did not.

2.19 Recipients

2.19.1 Listing Valid Recipients

To list all entries in the Valid Recipients table for a stream:

`GET /api/2.0/realm/`*`some_realm`*`/stream/`*`stream_name`*`/valid_recipients`

This will produce a list of serialized hashes. Each hash will contain:

stream Stream name.

recipient Recipient address.

realm The realm name.

2.19.2 Creating a Valid Recipient

To create an entry in the Valid Recipients table for a stream:

`PUT /api/2.0/realm/`*`some_realm`*`/stream/`*`stream_name`*`/valid_recipient/`*`recipient`*

The body of the PUT request should be a serialized hash containing:

recipient The recipient email address.

2.19.3 Deleting a Valid Recipient

To delete an entry from the Valid Recipients table for a stream:

`DELETE /api/2.0/realm/`*`some_realm`*`/stream/`*`stream_name`*`/valid_recipient/`*`recipient`*

recipient The recipient email address.

2.19.4 Listing Banned Recipients

To list all entries in the Blocked Recipients table for a stream:

`GET /api/2.0/realm/`*`some_realm`*`/stream/`*`stream_name`*`/banned_recipients`

This will produce a list of serialized hashes. Each hash will contain:

stream Stream name.

recipient Recipient address.

realm The realm name.

2.19.5 Creating a Banned Recipient

To create an entry in the Blocked Recipients table for a stream:

`PUT /api/2.0/realm/`*`some_realm`*`/stream/`*`stream_name`*`/banned_recipient/`*`recipient`*

The body of the PUT request should be a serialized hash containing:

recipient The recipient email address.

2.19.6 Deleting a Banned Recipient

To delete an entry from the Blocked Recipients table for a stream:

`DELETE /api/2.0/realm/`*`some_realm`*`/stream/`*`stream_name`*`/banned_recipient/`*`recipient`*

recipient The recipient email address.

2.20 Streams

2.20.1 Retrieving a Single Stream

To retrieve a single stream:

`GET /api/2.0/realm/`*`some_realm`*`/stream/`*`stream_name`*

This will return a serialized hash reference containing:

stream Stream name.

final Whether or not this stream is marked final.

opted_in Whether or not this stream is opted in.

special Whether or not this stream is marked special.

active Set to 1 if the stream appears in the Active Streams list; 0 otherwise.

parent_stream The name of the parent stream, or NULL if the stream has no parent.

realm The realm name.

parent_realm The realm name of the parent stream.

2.20.2 Listing Streams

To list all streams in a particular realm:

`GET /api/2.0/realm/`*`some_realm`*`/streams`

This will produce a list of serialized hashes. Each hash will contain the following:

stream The stream name.

opted_in Whether or not this stream is opted in.

admin_ok Whether or not to explicitly allow this stream to opt in to spam filtering.

special Whether or not this stream is marked special.

realm The realm name.

parent_stream The name of the parent stream.

parent_realm The realm name of the parent stream.

uri URI for retrieving further information on this stream.

2.20.3 Listing Active Streams

To list all currently-active streams in a particular realm:

`GET /api/2.0/realm/`*`some_realm`*`/streams/active`

This will produce a list of serialized hashes. Each hash will contain the same information as a stream list (see above).

2.20.4 Listing Special Streams

To list all special streams in a particular realm:

`GET /api/2.0/realm/`*`some_realm`*`/streams/special`

This will produce a list of serialized hashes containing the usual stream information (see above).

2.20.5 Listing Streams with Pending Incidents

To list all streams that have new pending incidents since the last Pending Notification was sent:

`GET /api/2.0/realm/`*`some_realm`*`/streams/with_pending`

This produces a list of serialized hashes. Each hash will contain a **stream** element giving the stream name and a **realm** element giving the realm name.

2.20.6 Updating a Stream

To update an existing stream:

`POST /api/2.0/realm/`*`some_realm`*`/stream/`*`stream_name`*

The POST body may contain:

admin_ok Whether or not to explicitly allow this stream to opt in to spam filtering. Valid values are 0 and 1.

opted_in Whether or not to opt this stream in for spam filtering. Valid values are 0, 1 and (representing undef)

special Whether or not to mark this stream as special. Valid values are 0 and 1.

final Whether or not to mark this stream as final. Valid values are 0 and 1.

parent_stream The name of the stream to inherit from. It should be a special stream.

parent_realm The realm in which **parent_stream** exists. You can only inherit across realms if **parent_realm** is an ancestor of the current realm and if **parent_stream** is `default`.

2.20.7 Creating a Stream

To create a new stream:

`PUT /api/2.0/realm/`*`some_realm`*`/stream/`*`stream_name`*

The body of the PUT request should be a serialized hash containing zero or more of the items for updating a stream explained above.

2.20.8 Deleting a Stream

To delete a stream:

`DELETE /api/2.0/realm/`*`some_realm`*`/stream/`*`stream_name`*

2.20.9 Listing Stream Settings

To list all settings for a given stream:

`GET /api/2.0/realm/`*`some_realm`*`/stream/`*`stream_name`*`/settings`

This will produce a list of serialized hashes. Each hash will contain:

variable The name of the setting.

value The value for this setting.

uri The URI for directly accessing this setting.

stream The stream name.

realm The realm name.

2.20.10 Retrieving a Single Stream Setting

To retrieve a single stream setting:

`GET /api/2.0/realm/`*`some_realm`*`/stream/`*`stream_name`*`/setting/`*`settingname`*

This returns a serialized hash, with the contents of the setting stored under the “value” key. In addition, if a stream is inheriting its setting from another stream, that stream’s name is in the “origin_stream” key and its realm name is in the “origin_realm” key.

2.20.11 Setting a Single Stream Setting

To set a single stream setting:

`PUT /api/2.0/realm/`*`some_realm`*`/stream/`*`stream_name`*`/setting/`*`settingname`*

The body of the PUT should contain a serialized hash, with the new contents of the setting stored under the ”value” key.

2.20.12 Deleting a Single Stream Setting

To delete a single stream setting:

`DELETE /api/2.0/realm/`*`some_realm`*`/stream/`*`stream_name`*`/setting/`*`settingname`*

2.20.13 Listing Users for a Stream

To list all allowed users for a stream:

`GET /api/2.0/realm/`*`some_realm`*`/stream/`*`stream_name`*`/users`

This will produce a list of serialized hashes. Each hash contains one key/value pair:

userid The userid of the users allowed for this stream

2.20.14 Getting the Pending Flag for a Stream

Each stream has a "pending" flag that is set when a new incident is created and cleared when a pending notification is generated. To get the pending flag for a stream:

`GET /api/2.0/realm/`*`some_realm`*`/stream/`*`stream_name`*`/pending_flag`

This produces a serialized hash containing the following:

pending_flag 1 if the stream has new pending incidents; 0 if it does not.

stream Stream name.

realm Realm name.

2.20.15 Setting the Pending Flag for a Stream

You can set or reset the pending flag for a stream:

`PUT /api/2.0/realm/`*`some_realm`*`/stream/`*`stream_name`*`/pending_flag`

The body of the PUT should contains a serialized hash with a single key **value**. The value should be 0 to reset the pending flag or 1 to set it.

2.20.16 Listing Addresses for a Stream

To list addresses explicitly mapped to a stream:

`GET /api/2.0/realm/`*`some_realm`*`/stream/`*`stream_name`*`/addresses`

This will produce a list of serialized hashes containing:

address The address itself.

stream The name of the stream for this address.

realm The name of the realm for this address.

cached 1 or 0, indicating whether or not this address mapping is a cached entry or a manually-entered one.

cache_date Creation time of cached entry as number of seconds from the UNIX epoch.

Note that only addresses explicitly mapped to the stream using the product's internal Address Mappings table will be listed here. Addresses mapped using a different Domain Mapping method will not be returned.

2.20.17 Requesting a Pending Notification

To request an immediate Pending Notification for a stream:

`POST /api/2.0/realm/`*`some_realm`*`/stream/`*`stream_name`*`/send_notification`

The body of the POST should contain a serialized empty hash. This API call requests a Pending Notification message to be sent. Note that it may take several minutes before the notification message is actually sent, because a background task generates the notifications.

2.20.18 Listing Addresses Seen

To retrieve a list of all email addresses seen by CanIt-Domain-PRO for a particular stream in the last 61 days:

`GET /api/2.0/realm/`*`some_realm`*`/stream/`*`stream_name`*`/addresses_seen`

This returns a list of addresses seen. Each entry of the list has the following keys:

address The email address.

last_seen The date the address was last seen.

stream The stream name.

realm The realm name.

Note: CanIt-Domain-PRO will *not* collect a list of addresses seen if it has no way to validate them. If none of the addresses seen for a stream can be verified (for example, there is no user-lookup or verification server, or the stream is used for outbound email) then CanIt-Domain-PRO will return an empty list.

2.21 Aliases

Note: CanIt-Domain-PRO only permits administrators and realm administrators to manipulate aliases via the API.

2.21.1 Listing Aliases in a Realm

To list all aliases in realm ***`some_realm`***:

`GET /api/2.0/realm/`*`some_realm`*`/aliases`

The returned data will be a list of hashes. Each hash will contain the following keys:

original_addr The address that is to be rewritten (the alias).

rewrite_addr The address to which it should be rewritten (the primary email address).

userid The user who created the alias.

realm The realm containing the user.

2.21.2 Retrieving an Alias

To retrieve a single alias:

`GET /api/2.0/realm/`*`some_realm`*`/alias/`*`original_addr`*

The returned data will be a single hash containing the same information as each element of the previous section.

2.21.3 Creating an Alias

To create an alias:

`PUT /api/2.0/realm/`*`some_realm`*`/alias/`*`alias@example.com`*

with a JSON body of:

```
{"rewrite_addr": "primary@example.com"}
```

This creates an alias that rewrites *alias@example.com* to *primary@example.com*.

2.21.4 Deleting an Alias

To delete an alias:

`DELETE /api/2.0/realm/`*`some_realm`*`/alias/`*`alias@example.com`*

2.22 User Administration

2.22.1 Listing Users in a Realm

To list all users in realm ***`some_realm`***:

`GET /api/2.0/realm/`*`some_realm`*`/users`

The returned data will be a list containing user records. Each record will be a hash containing the following keys:

email The user's email address (if known).

userid The user's User-ID.

can_edit 1 if the user has write-permission; 0 otherwise.

is_root 1 if the user has administrator privileges; 0 otherwise.

uri The URI for the user.

realm The realm containing the user.

2.22.2 Retrieving a User

To retrieve a single user:

`GET /api/2.0/realm/`*`some_realm`*`/user/`*`userid`*

The returned data will contain the same information as in the previous section.

2.22.3 Creating a User

To create a user in realm ***`some_realm`***:

`PUT /api/2.0/realm/`*`some_realm`*`/user/`*`userid`*

with a JSON body of:

```
{"email": "address@example.com",
 "password": "unencrypted_password_goes_here",
 "can_edit": 1,
 "is_root": 0}
```

If ***`userid`*** is an email address, the domain of that address must map to the realm specified in the URI, otherwise an error will be returned.

“email” and “password” are required arguments.

Currently, if the user should be prevented from logging in against the CanIt database (either because the user shouldn’t be permitted to log in at all, or because the user is authenticating using an external method), use a password of “*”.

`can_edit` controls whether or not the user has edit permissions, and should almost always be set to 1. If not provided, it defaults to 1.

`is_root` defaults to 0 if not provided.

2.22.4 Updating a User

To update a user’s information:

`POST /api/2.0/realm/`*`some_realm`*`/user/`*`userid`*

The POST body may contain:

can_edit controls whether or not the user has edit permissions in their allowed streams.

is_root if 1, sets the user to a root user in the user’s realm.

email sets the user’s email address.

password unencrypted password. The user’s password will be set to this value. Callers should ensure that they’ve done two-entry verification before POSTing a password change.

2.22.5 Deleting a User

To delete a user:

`DELETE /api/2.0/realm/`*`some_realm`*`/user/`*`userid`*

2.22.6 Listing a User's Allowed Streams

To list all allowed streams for a user:

`GET /api/2.0/realm/`*`some_realm`*`/user/`*`userid`*`/allowed_streams`

This will produce a list of serialized hashes. Each hash contains the same information returned by the stream-listing API calls.

2.22.7 Adding an Allowed Stream to User

To add an allowed stream to a user:

`POST /api/2.0/realm/`*`some_realm`*`/user/`*`userid`*`/allowed_stream`

The POST body must contain:

stream Name of stream to be added to this user.

2.22.8 Removing an Allowed Stream from User

To remove an allowed stream from a user:

`DELETE /api/2.0/realm/`*`some_realm`*`/user/`*`userid`*`/allowed_stream/`*`stream_name`*

2.22.9 Listing User Preferences

To list a user's preferences:

`GET /api/2.0/realm/`*`some_realm`*`/user/`*`userid`*`/preferences`

This will produce a list of serialized hashes. Each hash will contain:

variable The name of the preference.

value The value for this preference.

uri The URI for directly accessing this preference.

userid The user-ID.

realm The realm name.

2.22.10 Retrieving a User Preference

To retrieve a single user preference:

GET /api/2.0/realm/*some_realm*/user/*userid*/preference/*prefname*

This returns a serialized hash, with the contents of the setting stored under the ”value” key.

2.22.11 Setting a User Preference

To set a single user preference:

PUT /api/2.0/realm/*some_realm*/user/*userid*/preference/*prefname*

The body of the PUT should contain a serialized hash, with the new contents of the setting stored under the ”value” key.

2.22.12 Deleting a User Preference

To delete a single user preference:

DELETE /api/2.0/realm/*some_realm*/user/*userid*/preference/*prefname*

2.23 Group Administration

2.23.1 Listing Groups in a Realm

To list all groups in realm ***some_realm***:

GET /api/2.0/realm/*some_realm*/groups

The returned data will be a list containing multiple group records in the JSON format:

```
[{"groupid": "foo",
  "description": "The Foo Group",
  "realm": "foorealm"},
 {"groupid": "admins",
  "description": "All administrators",
  "realm": "base"}]
```

2.23.2 Retrieving a Group

To retrieve a single group:

GET /api/2.0/realm/*some_realm*/group/*groupid*

The returned data will contain the following:

```
{"groupid": "foo",
 "description": "The Foo Group",
 "realm": "foorealm"}
```

2.23.3 Creating a Group

To create a group in realm ***some_realm***:

PUT /api/2.0/realm/*some_realm*/group/*groupid*

with a JSON body of:

```
{"description": "Yet another group"}
```

In the URL and the body, the keys have the following meanings:

groupid The name of the new group

description A description of the group

Both ‘groupid’ and ‘description’ are required values.

2.23.4 Updating a Group

To update a group’s information:

POST /api/2.0/realm/*some_realm*/group/*groupid*

The POST body may only contain a value for ‘description’, as described above in ”Create a Group”.

2.23.5 Deleting a Group

To delete a group:

DELETE /api/2.0/realm/*some_realm*/group/*groupid*

2.23.6 Listing Members of a Group

To list all members of a group:

GET /api/2.0/realm/*some_realm*/group/*groupid*/members

This will produce a list containing multiple records in the format:

```
[{"userid": "foouser",
  "uri": "http://server/api/2.0/realm/realname/user/foouser"},
 {"userid": "baruser",
  "uri": "http://server/api/2.0/realm/realname/user/baruser"}]
```

2.23.7 Adding a Member to a Group

To add a new member to a group:

POST /api/2.0/realm/*some_realm*/group/*groupid*/member

The POST body must contain:

userid userid of user to be added to this group.

The user ID must be in the same realm as the group.

2.23.8 Removing a Member from a Group

To remove a member from a group:

`DELETE /api/2.0/realm/`*`some_realm`*`/group/`*`groupid`*`/member/`*`userid`*

2.24 User Lookups

2.24.1 Retrieving a User Lookup

To retrieve a single user lookup method:

`GET /api/2.0/realm/`*`some_realm`*`/user_lookup/`*`key`*

This will return a serialized hash reference containing:

key The name of the lookup method

settings The configuration values applicable to this user lookup method, as a serialized hash. Values vary from one method to another, but these will always exist:

- **Method** Back-end module used for this method (ie: LDAP, LDAP_AD, etc).
- **Comment** Description for this method.
- **UseForStreaming** Whether or not this method can be used for stream lookups.
- **UseForAuth** Whether or not this method can be used for authentication lookups.
- **Inheritable** If set to 1, this User Lookup may be inherited by child realms. Otherwise, it may only be used in the realm in which it exists.

2.24.2 Listing User Lookups

To list all user lookup methods:

`GET /api/2.0/realm/`*`some_realm`*`/user_lookups`

This will produce a list of serialized hashes. Each hash will contain the same information as an individual GET (see above).

2.24.3 Creating a User Lookup

To create a new user lookup method:

`PUT /api/2.0/realm/`*`some_realm`*`/user_lookup/`*`key`*

The body of the PUT request should be a serialized hash containing the same information as is retrievable via the GET method for an individual lookup method. See above for details.

2.24.4 Deleting a User Lookup

To delete a user lookup method:

DELETE /api/2.0/realm/*some_realm*/user_lookup/*key*

2.25 Verification Servers

2.25.1 Retrieving a Verification Server Entry

To retrieve one verification server entry:

GET /api/2.0/realm/*some_realm*/verification_server/*example.com*

This will return a serialized hash reference containing:

domain The domain for which this server will be used

server_list Serialized array reference containing one or more server names or IP addresses

queue_if_unavail Whether or not we should queue messages locally if this server is unreachable.

queue_only_seen_addresses Normally, CanIt-Domain-PRO queues mail for an address only if the address has been validated within the last 60 days. If this parameter is 0, then CanIt-Domain-PRO will queue mail for *all* addresses if the verification server is unreachable.

2.25.2 Listing Verification Servers

To list all verification server entries:

GET /api/2.0/realm/*some_realm*/verification_servers

This will produce a list of serialized hashes. Each hash will contain the same information as an individual GET (see above).

2.25.3 Creating a Verification Server Entry

To create a new verification server entry:

PUT /api/2.0/realm/*some_realm*/verification_server/*example.com*

The body of the PUT request should be a serialized hash containing the same information as is retrievable via the GET method for an individual lookup method, but without the 'domain' field, as this is referenced in the URI.

You can also use a POST request rather than PUT; the difference is that PUT will fail if an entry already exists for the domain where POST will create the entry if it does not exist or update the entry if it already exists.

Note that because POST requests do not allow structured data, you can specify the server list as a comma-separate string of servers rather than an array.

2.25.4 Deleting a Verification Server Entry

To delete a verification server entry:

`DELETE /api/2.0/realm/`*`some_realm`*`/verification_server/`*`example.com`*

2.26 Searching Logs

Note: The following API calls are available *only* if you have the **`canit-log-correlator`** package installed. This feature is available only on CanIt-Domain-PRO appliances; see the Administration Guide for details.

2.26.1 Performing a Log Search

To perform a search:

`GET /api/2.0/log/search/`*`offset`*`/`*`count`*`?`*`query`*

This call returns an array of log search results.

The URL parameters are:

- *offset*: A decimal number indicating the offset at which to begin returning results. An offset of 0 means to start from the beginning of the result set. An offset of 10 means to start after the first 10 items.
- *count*: The maximum number of results to return.

The query itself is coded as *url-encoded* key/value parameters. There are two ways to perform a search: A *simple* search and a *complex* search.

Simple Log Search

The possible search parameters for a simple search are:

- **`start_date`**: the date at which the log search should start (in the format YYYY-MM-DD.)
- **`start_time`**: the 24-hour time at which the log search should start (in the format HH-MM). If omitted, defaults to 00:00.
- **`end_date`**: the date at which the log search should end. If **end_date** is omitted, it defaults to today. If **start_date** is omitted, it defaults to the day before **end_date**.
- **`end_time`**: the 24-hour time at which the log search should end (in the format HH-MM). If omitted, defaults to 23:59.
- **`queue_id`**: match a Sendmail queue-ID.

- **`stream`**: match a particular stream.
- **`realm`**: match a particular realm. If a realm administrator performs a log search, the search is restricted to that realm and its descendants even if no **realm** parameter is supplied.
- **`incident_id`**: match an incident ID.
- **`message_id`**: match a Message-ID: header.
- **`sender`**: match an envelope sender.
- **`recipients`**: match one of the envelope recipients.
- **`subject`**: match a message subject.
- **`reporting_host`**: match the host reporting the log messages.
- **`src_relay_ip`**: match the IP address of the sending relay.
- **`dst_relay_ip`**: match the IP address of the relay to which the message was forwarded by CanIt-Domain-PRO.
- **`reason`**: match the "reason=xxx" field in the "what=yyy" log line.
- **`detail`**: match the "detail=xxx" field in the "what=yyy" log line.
- **`what`**: match the CanIt-Domain-PRO classification. Possible values for **what** are: "accepted", "rejected", "tagged", "discarded", "greylisted" or "pending".

In addition to the search fields, some allow you to choose a "contains" rather than "is" relation. To specify that, include a query parameter of **`rel_`*`field`*`=contains`**. The allowed fields for this purpose are:

- **`sender`**
- **`recipients`**
- **`subject`**
- **`message_id`**
- **`reporting_host`**

For **`score`**, you may specify a relation of

Here are some example log searches. The GET requests have been split into two lines for readability, but each request is really a single line:

```
GET /api/2.0/log/search/0/40?subject=Viagra
  &sender=gmail.com&rel_subject=contains
```

```
GET /api/2.0/log/search/0/40?sender=bob@example.org
  &recipients=june&rel_recipients=contains
```

Complex Log Search

To perform a *complex* log search, supply only **`start_date`**, **`end_date`** and **`query`** parameters. The **`query`** parameter is a *serialized query*. You can obtain the correct query by generating a query in the Web interface and then clicking "(Show query for /log/search API)". Here is an example that looks for log lines where the sender contains "gmail.com" and the recipient contains "dfs" or "billw". (The query is shown split across multiple lines for readability.)

```
GET /api/2.0/log/search/0/40?query=(,(,sender,contains,gmail.com,),)
  ,AND,(,(,recipients,contains,dfs,),OR,(,recipients,contains,billw,),)
```

Results of a Log Search

The return value from a log search is an array of log search results. Each result is a hash with some or all of the following members:

- **`what`**: The message classification.
- **`recipients`**: An array of message recipients.
- **`ts`**: The UNIX timestamp of the first log line.
- **`subject`**: The message subject.
- **`stream`**: The stream.
- **`realm`**: The realm.
- **`incident_id`**: The incident ID.
- **`reporting_host`**: The host that reported the logs.
- **`queue_id`**: The Sendmail queue-ID.
- **`sender`**: The envelope sender.

2.26.2 Obtaining Actual Log Lines

A set of log lines is uniquely identified by the combination of **`reporting_host`** and **`queue_id`**. Both of those items are part of each result hash returned from a search. To obtain the actual log lines themselves, call:

`GET /api/2.0/log/`*`queue_id`*`/`*`reporting_host`*

This call returns an array of hashes (typically, there's only one array element but in some cases there could be more than one.) Each element of the array is a hash containing some or all of the following members:

- **`loglines`**: An array of actual log lines.

- **`ts`**: The UNIX timestamp of the first log line.
- **`reporting_host`**: The host that reported the logs.
- **`queue_id`**: The Sendmail queue-ID.
- **`what`**: The message classification.
- **`recipients`**: An array of message recipients.
- **`subject`**: The message subject.
- **`stream`**: The stream.
- **`realm`**: The realm.
- **`incident_id`**: The incident ID.
- **`message_id`**: The message ID.
- **`score`**: The incident score, if any.
- **`reason`**: The "reason=xxx" entry, if any.
- **`detail`**: The "detail=xxx" entry, if any.
- **`reporting_host`**: The host that reported the logs.
- **`sender`**: The envelope sender.
- **`src_relay_ip`**: The source relay IP address.
- **`dst_relay_ip`**: The destination relay IP address.

2.27 UserDB

The UserDB is a general-purpose key/value store. It is accessible only via the API; there is no access to it via the Web interface. The UserDB allows each user to store and retrieve key/value pairs in CanIt-Domain-PRO's database.

A *value* is simply a text string. The database uses two keys to locate a value: The *family* and the *variable*. In addition, each user who logs in via the API has his or her own completely separate UserDB.

2.27.1 Setting a Value in the UserDB

To set a particular value in the UserDB:

`POST /api/2.0/realm/`*`some_realm`*`/user/`*`userid`*`/db/`*`family`*`/`*`variable`*

(The realm and userid fields will typically be `@@` unless you want to manipulate another user's UserDB.)

The POST body must contain:

value The value to which the UserDB entry should be set.

2.27.2 Retrieving Values from the UserDB

There are three API calls for retrieving values from the UserDB:

GET /api/2.0/realm/*some_realm*/user/*userid*/db/*family*/*variable*

GET /api/2.0/realm/*some_realm*/user/*userid*/db/*family*

GET /api/2.0/realm/*some_realm*/user/*userid*/db

The first form retrieves a single entry. The return value is a hash containing the following elements:

family The entry's family.

variable The entry's variable name.

value The entry's value.

If the variable does not exist, a "404 Not found" error is raised.

The second form retrieves all entries in the given family. It returns a list of hashes where each hash has the elements described above. Note that if there are no entries, an empty list is returned rather than an error being raised.

The third form retrieves all entries in the UserDB. The return value is the list of hashes described above.

2.27.3 Deleting Values from the UserDB

There are three API calls for deleting values from the UserDB:

DELETE /api/2.0/realm/*some_realm*/user/*userid*/db/*family*/*variable*

DELETE /api/2.0/realm/*some_realm*/user/*userid*/db/*family*

DELETE /api/2.0/realm/*some_realm*/user/*userid*/db

The first form deletes a single variable. The second deletes an entire family and the third deletes all entries in the UserDB. (As mentioned, each user has his or her own UserDB, so deleting values affects only one user's UserDB.)

2.28 Audit Trail

CanIt-Domain-PRO provides access to the Audit Trail feature. (This is the feature that tracks changes to settings, rules, etc.)

2.28.1 Audit Paths

Every auditable feature is associated with a *path*. The list of accessible paths is given by the following API call:

`GET /api/2.0/realm/`*`some_realm`*`/stream/`*`stream_name`*`/audit/paths`

The return value is an array; each element is a string that specifies a path. Examples of paths are **`rules/senders`** for auditing changes to sender rules, **`prefs/prefs`** for auditing changes to preferences, etc.

2.28.2 Audit Trails

To retrieve an audit trail, use the following API call:

`GET /api/2.0/realm/`*`some_realm`*`/stream/`*`stream_name`*`/audit/`*`path`*

where *path* is one of the elements returned by the **`audit/paths`** API call. For example, to retrieve a list of changes to sender rules, use:

`GET /api/2.0/realm/`*`some_realm`*`/stream/`*`stream_name`*`/audit/rules/senders`

The return value for an audit trail is an array of hashes that detail changes to settings. Each hash has the following elements:

path The path given in the API request.

type One of **`INSERT`**, **`DELETE`** or **`UPDATE`**.

userid The User-ID of the person who made the change.

realm The realm containing the User-ID of the person who made the change.

stream_affected The stream (if any) in which the change was made.

realm_affected The realm (if any) in which the change was made.

date The date the change was made.

key_columns An array of the key columns identifying the row changed. Each element of the array is a two-element array; the first element is the column name and the second element is the column value.

value_columns An array of the value columns identifying the changes made to the row. For an **`INSERT`** change, each element is a two-element array giving the column name and the newly-inserted value. For a **`DELETE`** change, each element is a two-element array giving the column name and the old value prior to deletion. Finally, for an **`UPDATE`** change, each element is a three-element array giving the column name, the old value before the update, and the new value after the update.

You can limit the amount of data returned by an audit-trail request by supplying one or more of the following URL parameters in the **`GET`** request:

- **`startdate`** The starting date in the form YYYY-MM-DD. Any changes earlier than this date will not be returned.

- **`enddate`** The ending date in the form YYYY-MM-DD. Any changes later than this date will not be returned.
- **`filter`** An arbitrary string. Only changes whose key-column or value-column values contain the string will be returned.

2.29 Archived Mail

CanIt-Domain-PRO provides API access to the Mail Archiving feature.

Note: Mail Archiving is an add-on component for CanIt-Domain-PRO. If you have not purchased the archiving component, the API calls in this section will not be available. If you wish to purchase mail archiving, please contact your sales representative.

2.29.1 Searching the Archive

To search archived mail:

`GET /api/2.0/archive/search/`*`offset`*`/`*`count`*`?query=`*`query`*

This call returns an array of archive search results.

The URL parameters are:

- *offset*: A decimal number indicating the offset at which to begin returning results. An offset of 0 means to start from the beginning of the result set. An offset of 10 means to start after the first 10 items.
- *count*: The maximum number of results to return.

The query itself is coded as a *serialized query* given in the **query** GET parameter. Additional GET parameters are:

- **`startdate`**: The earliest date to search in the form *YYYY-MM-DD*. If omitted, it is set to 30 days before **`enddate`**.
- **`enddate`**: The latest date to search in the form *YYYY-MM-DD*. If omitted, it is set to the current date.

Serialized Queries

A serialized query has the following form. Note that the spaces are for readability only and should not appear in the serialized query. The commas and parentheses are literal and should appear exactly as-is.

`serialized_query:` *`expr`* **`[`** `,` *`LOGOP`* `,` *`expr`* ... **`]`**

An individual expression *expr* is defined as:

expr: (, *term* **[,** *LOGOP* **,** *term* **...] ,)**

And finally, each *term* is:

term: (, *field* **,** *relation* **,** *value* **,)**

A *LOGOP* is one of **AND**, **OR**, **ANDNOT** or **ORNOT**. The **NOT** variants serve to negate the next term or expression.

The *field* is one of the following:

- **subject**
- **body**
- **envelope_sender**
- **header_from**
- **envelope_recipients**
- **attachment_filenames**
- **realm**
- **stream**
- **id**
- **helo**
- **relay_address**
- **relay_hostname**
- **real_relay_address**
- **real_relay_hostname**
- **message_id**
- **refs**
- **queue_id**
- **archive_host**
- **size**

The *relation* is one of the following:

- **is** (meaning case-insensitive match)
- **contains** (meaning case-insensitive substring match)

- **`gt`** (meaning > case-insensitively)
- **`lt`** (meaning < case-insensitively)
- **`ge`** (meaning ≥ case-insensitively)
- **`le`** (meaning ≤ case-insensitively)
- **`matches`** (meaning a case-insensitive full-text match)

The *relation* has the following restrictions:

- **`subject`** and **`body`** permit *only* the **`matches`** relation.
- **`envelope_recipients`**, **`attachment_filenames`** and **`refs`** permit only **`is`** and **`contains`**.
- All other fields permit all possible relations except for **`matches`**.

The *value* is the value to match. A comma within the value *must* be encoded as **`%2C`** and a percent sign *must* be encoded as **`%25`**. This encoding happens before any URL-encoding required for GET parameters.

The easiest way to generate a serialized query is to create a query interactively via the CanIt-Domain-PRO web interface. Then click on **(Show query for API)** to reveal the serialized query.

Serialized Query Examples

Query:

- Subject matches Testing

Serialized Query:

```
(,(,subject,matches,Testing,),)
```

Query:

- Header From is bob@example.com AND Envelope Recipient is jane@example.net ***OR***
- Header From is jane@example.net AND Envelope Recipient is bob@example.com

Serialized Query shown split over several lines (it should really be one long string.)

```
(,(,header_from,is,bob@example.com,),AND,\
(,envelope_recipients,is,jane@example.net,),),OR,\
(,(,header_from,is,jane@example.net,),AND,\
(,envelope_recipients,is,bob@example.com,),)
```

Query:

- Subject matches two,three%

Serialized Query:

```
(,(,subject,matches,two%2Cthree%25,),)
```

Search Results

A search returns an array of hashes (one hash for each archived message that matched the search.) Each hash contains the following elements:

- **`id`**: An identifier that can be used later on to retrieve the actual archived message.
- **`path`**: An identifier that must be used in conjunction with **`id`** to retrieve a message.
- **`realm`**: The realm containing the message.
- **`stream`**: The stream containing the message.
- **`helo`**: The HELO string emitted by the SMTP client.
- **`relay_address`**: The IP address of the SMTP client that directly delivered the message to CanIt-Domain-PRO.
- **`relay_hostname`**: The host name of the SMTP client that directly delivered the message to CanIt-Domain-PRO.
- **`real_relay_address`**: The IP address of the originating relay.
- **`real_relay_hostname`**: The host name of the originating relay.
- **`message_id`**: The Message-Id: header contents.
- **`refs`**: An array of entries in the References: header.
- **`queue_id`**: The Sendmail queue-ID of the processed mail.
- **`subject`**: The message subject.
- **`envelope_sender`**: The envelope sender.
- **`envelope_recipients`**: An array of envelope recipients.
- **`archive_timestamp`**: When the message was archived. This is a string in the form “YYYY-MM-DD hh:mm:ss-offset” where “offset” is the number of hours offset from UTC.
- **`archive_host`**: The name of the host that archived the message.
- **`header_from`**: The address in the message’s From: header.

- **`header_sender`**: The address in the message's Sender: header (if any).
- **`size`**: The size of the message in bytes.
- **`attachment_filenames`**: An array of attachment filenames.

2.29.2 Retrieving an Archived Message

To retrieve an archived message, use:

`GET /api/2.0/archive/message/`*`path`*`/`*`id`*

Here, *path* is the **`path`** member of the hash returned by a search and *id* is the **`id`** member. Note that a path may include slashes. *Do not* url-encode the slashes. That is, if a message has a path that is `base/20110406/ay` and an ID that is `01Eskvpay`, then the API call to retrieve it is:

`GET /api/2.0/archive/message/base/20110406/ay/01Eskvpay`

Note: This API call *always* returns data of type **`message/rfc822`** and *not* YAML or JSON! The API client libraries supplied by AppRiver handle this special case and convert the returned data into a hash with the single element **`message`** containing the entire MIME message.

2.29.3 Retrieving Archived Metadata

To retrieve metadata about an archived message, use:

`GET /api/2.0/archive/meta/`*`path`*`/`*`id`*

This API call returns a hash that contains metadata about the message. This hash has exactly the same elements as the hashes returned in the array from the **`/api/2.0/archive/search`** API call.

2.29.4 Searching for Related Messages

Given an existing message **id**, you can search for related messages as follows:

`GET /api/2.0/archive/related/`*`offset`*`/`*`count`*`?`*`query`*

This call returns an array of archive search results.

The URL parameters are:

- *offset*: A decimal number indicating the offset at which to begin returning results. An offset of 0 means to start from the beginning of the result set. An offset of 10 means to start after the first 10 items.
- *count*: The maximum number of results to return.

The query itself is coded as *url-encoded* key/value parameters. The search parameters are:

- **`startdate`**: The earliest date to search.
- **`enddate`**: The latest date to search.

- `id`: The ID of an existing message.

The return value is an array of hashes exactly as returned by the **`/api/2.0/archive/search`** call. The messages returned are all related to the original message **id** in one of the following ways:

- A returned message's Message-ID appears in **id**'s References: header.
- A returned message's References: header contains **id**'s Message-ID.

Generally, the returned array contains a complete conversational thread.

2.29.5 Configuring Archiving

Obtaining Current Configuration

To obtain the current archiving configuration for a stream, use this API call:

GET /api/2.0/realm/*some_realm*/stream/*stream_name*/archive_config

The result is an array of hashes. Each hash is in exactly the same format as the Stream Setting hash described in Section 2.20.10.

Updating Archiver Configuration

To make a change to the archiving configuration for a stream, use:

POST /api/2.0/realm/*some_realm*/stream/*stream_name*/archive_config

The body of the POST should be a series of key/value pairs:

- **ArchiveMail** — the value of this key should be `0` to disable archiving for the stream or `1` to enable it.
- **ArchiveTaggedMail** — the value of this key should be `0` to disable archiving of tagged mail (if the stream is in tag-only mode), or `1` to enable archiving of tagged mail. Note that if **ArchiveMail** is set to 0, then no mail is archived even if **ArchiveTaggedMail** is set to 1.
- **ArchiveMaxAttachmentSizeKB** — the value of this key should be set to an integer specifying the maximum size of attachment (in kB) to archive. Larger attachments are stripped and not put in the archive. Setting this value to zero means all attachments are archived regardless of size.

The POST body can additionally contain the following key/value pairs, but *only* if the stream is **default** (which implies that these settings can be adjusted only by realm administrators:)

- **ArchiveExpiryMonths** — an integer specifying how long to retain archived mail in months. A value of -1 specifies indefinite retention.
- **ArchiveMegabytesPerZip** — an integer specifying the maximum size of Zip file to create, in megabytes. Zero means no limit.

- **ArchiveMessagesPerZip** — an integer specifying the maximum number of messages to put in a Zip file. Zero means no limit.
- **ArchiveMonthlyZipNotification** — if this is set to a non-blank value, then it should be set to an email address. In this case, the system will automatically zip up each month's worth of about-to-expire messages and send a notification to the specified email address. To disable the monthly zip file via the API, use a value of `0` for this parameter.

The POST body can further contain the following key/value pairs, but **only** if the stream is **default** and *only* if the API call is made by the CanIt-Domain-PRO site administrator:

- **ArchiveAllowExpiryZip** — if this is set to zero, then other users cannot request a monthly zip file of about-to-expire messages. If set to one, then users can request the zip file.
- **ArchiveMaxMegabytesPerZip** — specifies an upper limit on the allowable value of **ArchiveMegabytesPerZip**.
- **ArchiveMaxMessagesPerZip** — specifies an upper limit on the allowable value of **ArchiveMessagesPerZip**.
- **ArchiveMaxPossibleAttachmentKB** — specifies an upper limit on the allowable value of **ArchiveMaxAttachmentSizeKB**.
- **MaxArchiveExpiryMonths** — specifies an upper limit on the allowable value of **ArchiveExpiryMonths**.

2.30 Secure Messaging

CanIt-Domain-PRO provides API access to configure Secure Messaging.

Note: Secure Messaging is an add-on component for CanIt-Domain-PRO. If you have not purchased the Secure Messaging component, the API calls in this section will not be available. If you wish to purchase Secure Messaging, please contact your sales representative.

2.30.1 Configuring Secure Messaging

Obtaining Current Configuration

To obtain the current Secure Messaging configuration for a stream, use this API call:

`GET /api/2.0/realm/`*`some_realm`*`/stream/`*`stream_name`*`/secure_messaging_config`

The result is an array of hashes. Each hash is in exactly the same format as the Stream Setting hash described in Section 2.20.10. The possible variables are:

- **EncryptMail** — this variable is set to 1 if secure messaging is enabled for the stream or 0 if it is not.

- **EncryptedEmailExpiryDays** — this variable specifies how long secure messages are retained before being expired. It can range from 7 to 180.

Updating Secure Messaging Configuration

To make a change to the Secure Messaging configuration for a stream, use:

`POST /api/2.0/realm/`*`some_realm`*`/stream/`*`stream_name`*`/secure_messaging_config`

The body of the POST should be a series of key/value pairs. The possible keys and ranges of values are described above in the list of possible return values for the GET call.

2.31 Provisioning a Domain

The provisioning API call is a convenience feature that lets you quickly provision a domain. It implements most of the "Domain Setup Wizard" via an API call.

Normally, only the site administrator can provision domains. However, you can grant realm administrators permission to provision domains with the **Provision Domains via API** permission. *Granting permission to provision domains confers much power to a realm administrator and carries security risks. Do not allow realm administrators to provision domains unless they are trusted.*

To provision a domain, make the following API call:

`POST /api/2.0/provision`

The following POST variable must be supplied:

- **domain** — the name of the domain to provision.

The following POST variables are optional; at most one should be supplied:

- **in_realm** — the name of an *existing* realm in which to map the domain. The caller of the API must own the realm.
- **new_realm** — the name of a *new* realm which should be created and in which the domain will be placed. The realm *must not* exist.

If neither **in_realm** nor **new_realm** is supplied, CanIt-Domain-PRO constructs a realm name based on the domain name. If you require a new realm, we recommend *not* supplying **new_realm**, but instead allowing CanIt-Domain-PRO to construct the name of the new realm.

If you do not supply **in_realm**, the following parameter may be supplied:

- **parent_realm** — the name of the parent realm for the newly-created realm. The parent realm must exist and must be owned by the API caller. If this parameter is not supplied, then the parent realm is decided as follows:

1. If the domain to be provisioned is a subdomain of an existing provisioned domain, then the existing provisioned domain's realm becomes the parent of the new realm. If the API caller does not own this realm, then the API call fails with an error.
2. Otherwise, the API calling user's realm becomes the parent realm of the new realm.

If a new realm is being created, the following variables may be supplied.

- **description** — Text to put in the new realm's "description" field. This is a *required* parameter if a new realm is being created.
- **expiry** — The expiry date (in the form YYYY-MM-DD) of the new realm. This parameter is optional and is ignored if the API caller does not have permission to set a realm's expiry date.
- **userfield1** through **userfield4** — Data to store in the user-defined fields for a new realm. These parameters are optional and are ignored if the API caller does not have permission to set the fields.

The following additional parameters may be supplied:

- **domain_mapping** — The domain mapping to use for the newly-provisioned domain. May be one of **AsIs**, **Database**, **ChopUser** or any other valid mapping method.
- **route_to** — *(ONLY for CanIt-Domain-PRO appliances or Hosted CanIt; API call fails if supplied on non-appliances.)* A comma-separated list of servers or IP addresses to which to route mail. Note that if you create a domain route, you *must* activate the routes in a separate API call (Section 2.12.5). If you supply the **route_to** parameter, the following additional parameters may be supplied:
  - **route_treat_as_mx** — Either 0 or 1 (default 0). 1 specifies that the servers in the **route_to** parameter should be treated as MX records rather than A records. *In most cases, you should omit this parameter.*
  - **port** — A number from 1 to 65535 specifying the TCP port to which to route mail (default 25). Note that only the site administrator can specify a port other than 25 or 587 that is lower than 1024.
  - **queue_threshold** — a positive number specifying how many queued messages should accumulate before a warning notification is sent.
  - **msg_age_threshold** — a positive number specifying how many hours a message must be queued before a warning notification is sent.
  - **notify_address** — the email address to which queue warning notifications should be sent. This email address *cannot* be in the domain being provisioned.
- **verification_server** — If this parameter is supplied, it should be a comma-separated list of verification servers for the domain. Also, the following optional parameters may be supplied:
  - **verification_treat_as_mx** — Either 0 or 1 (default 0). 1 specifies that the servers in the **verification_server** parameter should be treated as MX records rather than A records. *In most cases, you should omit this parameter.*

  - **queue_if_unavail** — Either 0 or 1 (default 1). 1 specifies that mail should be queued if the verification server is unreachable whereas 0 specifies that mail should be tempfailed in that situation.
  - **queue_only_seen_addresses** — Either 0 or 1 (default 1). 1 specifies that if the verification server is unreachable, then only queue for recipients that have been validated in the last 60 days. Other recipients will be tempfailed.

- **create_admin_user** — Either 0 or 1 (default 0). If a new realm is being created, specifying this parameter as 1 causes CanIt-Domain-PRO to create a new administrative user (with a random password) in the new realm. If this parameter is supplied as 1 for an existing realm, the API call fails with an error. If a new user is being created, the following parameter is required:
  - **admin_email** — The email address of the new administrative user.

If the API call succeeds, a hash will be returned. The hash will contain some of the following members. The **domain**, **realm** and **parent_realm** members are always present. The other members are present only if the appropriate parameter was passed into the provisioning request.

- **domain** — the name of the newly-provisioned domain.
- **realm** — the name of the realm in which the domain was provisioned.
- **parent_realm** — the parent realm of **realm**.
- **route_to** — an array of servers to which to route mail.
- **route_treat_as_mx** — whether or not route_to names are treated as MX records.
- **port** — the port for the route_to server.
- **verification_server** — the comma-separated list of verification servers.
- **verification_treat_as_mx** — whether or not verification_server names are treated as MX records.
- **queue_if_unavail** — whether or not to queue if the verification server is unreachable.
- **queue_only_seen_addresses** — whether or not to queue only recently-validated addresses (1) or all addresses (0).
- **domain_mapping** — the name of the domain mapping that was created.
- **admin_userid** — if an administrative user was created, the user-ID of the user. It is always **`admin@`*`domain`***.
- **admin_email** — the email address of the administrative user.
- **admin_password** — the password for the administrative user. This is a randomly-generated 8-character password.

2.32 Obtaining Provisioning Counts

The following API call returns data showing various usage counts for a given realm. This API call is available only to realm administrators.

`GET /api/2.0/realm/`*`some_realm`*`/provisioning`

The return value is an array of hashes. One hash is returned for the supplied realm and for each of its descendants, all the way down the inheritance tree.

Each hash corresponds to one realm and contains the following members:

name The name of the realm.

parent The name of the realm's parent.

level The level in the descendant tree. The original supplied realm has a level of 0. Its direct children have a level of 1. Their children have a level of 2 and so on.

addresses The number of addresses in this realm *only* that have received email in the last 30 days.

addresses_all The number of addresses in this realm *and all of its descendants* that have received email in the last 30 days.

streams The number of streams in this realm *only* that have received email in the last 30 days.

streams_all The number of streams in this realm *and all of its descendants* that have received email in the last 30 days.

uses_outbound 1 if any domain mapped to this realm is also listed as an associated domain for a known network; 0 otherwise.

outbound_addresses 0 if **uses_outbound** is false; otherwise the same as **addresses**.

outbound_addresses_all The sum of all the **outbound_addresses** in this realm *and all of its descendants*.

outbound_streams 0 if **uses_outbound** is false; otherwise the same as **streams**.

outbound_streams_all The sum of all the **outbound_streams** in this realm *and all of its descendants*.

expiry The expiry date (in the form YYYY-MM-DD) of the realm. This member is absent if the user making the API call lacks permission to read the realm expiry date.

user_fields A four-element array corresponding to the user-fields associated with the realm. Elements for which the user lacks read permission are returned as **null**. If the user lacks permission to see any user field, then the entire element is absent.

domains An array of domains associated with the realm. Each element of the array is a hash containing the following elements:

- **name** The name of the domain.

- **mx_points_here** True if the domain's MX records point at CanIt-Domain-PRO as of the last nightly check; false otherwise.
- **rejects_bad** True if the domain correctly validates recipients as of the last nightly check; false otherwise.
- **outbound_networks** If the domain is associated with any Known Networks entries, this element consists of an array of associated Known Networks. If the domain is not associated with any Known Networks, then this element is an empty array.

If the Archiver add-on is installed, the following additional members are present:

archive_months The archive retention time in months for this realm.

archiving_streams The number of streams with archived mail in the last 90 days in this realm *only*.

archiving_streams_all The number of streams with archived mail in the last 90 days in this realm *and all of its descendants*. This element is a hash; the key of each element is a number specifying the number of months archived mail is retained and the value is the count of streams that retain mail for that long, in this realm and all of its descendants.

If the Secure Messaging add-on is installed, the following additional members are present:

encryption_streams The number of streams that have enabled Secure Messaging in this realm **only**.

encryption_streams_all The number of streams that have enabled Secure Messaging in this realm *and all of its descendants*.

2.33 Obtaining Provisioning History

The following API call returns data showing historical usage counts for a given realm. This API call is available only to realm administrators.

`GET /api/2.0/realm/`*`some_realm`*`/provision_history`

The following optional GET parameters may be supplied:

- **start_date**: The start date from which historical data is required, in the form YYYY-MM-DD.
- **end_date**: The end date up to which historical data is required, in the form YYYY-MM-DD.
- **ndays**: The number of days' worth of data required.

If both **start_date** and **end_date** are supplied, then **ndays** is ignored. If only one date is supplied, then data is returned for **ndays** starting at **start_date** or for **ndays** ending at **end_date**, depending on which date is required.

If one date is supplied but **ndays** is not, then **ndays** defaults to 365. If neither date is supplied, then **end_date** defaults to the current date.

The return value is an array of hashes. Each hash contains the following members:

- **realm** — the name of the realm.
- **date** — the date of the provisioning snapshot, in the form YYYY-MM-DD.
- **addresses** — the number of inbound email addresses seen.
- **streams** — the number of inbound streams seen.
- **addresses_all** — the number of inbound email addresses seen, *including* subrealms.
- **streams_all** — the number of inbound streams seen, *including* subrealms.
- **outbound_addresses** — the number of outbound email addresses seen.
- **outbound_streams** — the number of outbound streams seen.
- **outbound_addresses_all** — the number of outbound email addresses seen, *including* subrealms.
- **outbound_streams_all** — the number of outbound streams seen, *including* subrealms.
- **archiving_streams** — the number of streams seen that have email archiving enabled.
- **archiving_streams_all** — the number streams seen that have email archiving enabled, *including* subrealms. NOTE: This return value is not a simple number but rather a hash; the keys of the hash are a number representing the number of months of archive retention and the corresponding values are the number of streams that archive for that length of time.
- **encryption_streams** — the number of streams seen that have Secure Messaging enabled.
- **encryption_streams_all** — the number streams seen that have Secure Messaging enabled, *including* subrealms.
- **archive_months_net** — this is **archiving_streams** multiplied by the retention time of archived mail in months. For example, if 12 streams are archiving mail in a given realm and the retention time is 24 months, then **archive_months_net** will be $12 \times 24 = 288$.
- **archive_months_net_all** — the sum of **archive_months_net** in this realm and all subrealms.

Note: If Archiving or Secure Messaging is not installed, then the corresponding hash elements are not present in the results.

2.34 Single Sign On

CanIt-Domain-PRO has a mechanism that can be used to implement single sign on from within another Web application. The general flow is as follows:

1. A user logs on to a Web application that you control, such as a Webmail platform.
2. You create a script in PHP, Perl, Python or any other programming language of your choice that can be accessed within your Web application. You provide a link to this script.

3. When the user clicks the link, your script runs. It obtains the user's credentials from your Web application and then makes an API call to CanIt-Domain-PRO. CanIt-Domain-PRO generates a temporary URL that automatically logs in the user; your script then redirects to that URL with a `Location:` header.

The API call to generate the temporary URL is as follows:

`POST /api/2.0/xauth`

The body of the POST should contain:

ttl A time-to-live for the temporary URL, in seconds. This parameter is *required* and can range from 5 to 60 seconds. The URLs are deliberately very short-lived to limit the damage if they somehow leak out.

userid The userid that will be automatically logged in. This parameter is required.

logout_redirect (Optional) If this parameter is supplied, it must be a valid URL. If the user logs out of CanIt-Domain-PRO, then his or her browser is redirected to this URL. You can use this mechanism to also log the user out of a single sign-on system, either unconditionally or after confirmation according to your preferred policy.

Note that if the user does not log out of CanIt-Domain-PRO, but simply lets the session time out, the browser will never be redirected to the **logout_redirect** URL.

redirect (Optional) The URL to which the user should be redirected. By default, this is set to `index.php` which takes the user to the CanIt-Domain-PRO home page, but you can supply any valid CanIt-Domain-PRO URL for this parameter subject to the following restrictions:

1. The URL cannot include a scheme such as `http:` or a host name.
2. The URL must be relative. That is, it cannot start with a slash.
3. The URL can include query parameters.

Examples of valid settings of the **redirect** parameter:

- `rules.php`
- `showincident.php?id=01LcgN1iS&submit=Go`

Examples of invalid settings of the **redirect** parameter:

- `http://example.com/canit/rules.php` (Cannot include a scheme or a host)
- `/canit/showincident.php?id=01LcgN1iS` (URL cannot be absolute)

stream (Optional) The "home stream" that the user will be placed in after logging in. If this parameter is omitted, then CanIt-Domain-PRO determines the home stream as follows:

- If the userid looks like an email address (`local@example.com`), then CanIt-Domain-PRO runs it through the address-to-stream process and uses the resulting stream.
- Otherwise, the home stream is the same as the userid.

realm (Optional) The realm containing the userid. If this parameter is omitted, then CanIt-Domain-PRO determines the realm as follows:

- If the userid looks like an email address (**`local@example.com`**), then CanIt-Domain-PRO runs it through the address-to-stream process and uses the resulting realm.
- Otherwise, the realm is set to **`base`**.

Note that if you supply either **realm** or **stream**, then you must supply both of them.

Upon success, the **`xauth`** call returns a hash with a single element:

- **`url`** An *auto-login URL* to which the user's browser should be redirected. When this URL is loaded, the user is automatically logged in. After an auto-login URL has been used once, it may not be reused. Since a browser is typically redirected to the auto-login URL very quickly after it is created, the likelihood of an auto-login URL leaking into the wrong hands is very small.

Note that the **`xauth`** call will *refuse* to create a login link for a user with administrative privileges. Such users must always log in via the CanIt-Domain-PRO login page. The **`xauth`** API call is available only to realm administrators. It can only be used to authenticate users in the administrator's realm or any subrealm thereof.

2.35 Deprecated URLs

The following URLs exist for backward-compatibility with the 1.0 API. However, you should use the new versions because the old ones will be removed in a future release.

Old: **`/api/2.0/users`**
New: **`/api/2.0/realm/*/users`**

Old: **`/api/2.0/address_mapping`**
New: **`/api/2.0/realm/*/address_mapping`**
Old: **`/api/2.0/address_mappings`**
New: **`/api/2.0/realm/*/address_mappings`**

Old: **`/api/2.0/auth_mapping`**
New: **`/api/2.0/realm/*/auth_mapping`**
Old: **`/api/2.0/auth_mappings`**
New: **`/api/2.0/realm/*/auth_mappings`**

Old: **`/api/2.0/domain_mapping`**
New: **`/api/2.0/realm/*/domain_mapping`**
Old: **`/api/2.0/domain_mappings`**
New: **`/api/2.0/realm/*/domain_mappings`**

Old: **`/api/2.0/verification_server`**
New: **`/api/2.0/realm/*/verification_server`**
Old: **`/api/2.0/verification_servers`**
New: **`/api/2.0/realm/*/verification_servers`**

Chapter 3

API Client Libraries

CanIt-Domain-PRO ships with three client libraries: One in Perl, one in PHP and one in Python. We recommend that you use our client libraries if possible; they make interacting with the API server much easier.

CanIt-Domain-PRO also ships with a command-line tool that offers API access from shell scripts. While this can be used for simple tasks, we do not recommend using it for important production tasks because of the quoting problems inherent in UNIX shell scripting.

3.1 Perl Library: CanIt::API::Client

The Perl library is called `CanIt::API::Client`. You can install the `CanIt/API/Client.pm` on any machine; it does not depend on any other CanIt modules. `CanIt::API::Client` works with Perl 5.8 or newer and has the following non-core dependencies:

- `LWP` (sometimes called `libwww-perl`)
- `URI`
- `JSON::Any`

Most Linux distributions provide prepackaged versions of the dependencies. If your system does not provide them, they are easily obtainable on CPAN.

`CanIt::API::Client` provides an object-oriented programming interface. In general, you create a `CanIt::API::Client` object and then call methods on it. You can run `man CanIt::API::Client` for the man page.

3.1.1 Constructor

To create a `CanIt::API::Client` object, call the `new` method with a single hashref argument. The hashref should contain two members: `base` is the base URL of the API server and `version` is the API server version (currently 2.0). For example:

```
use CanIt::API::Client;

my $api = CanIt::API::Client->new({
    base     => 'http://canit.example.com/api',
    version  => '2.0',
});
```

3.1.2 Logging In and Out

After creating the `CanIt::API::Client` object, you need to log in to the API server. When you have finished, you should log out. For example:

```
$api->login('username', 'password');
# Do things with $api
$api->logout();
```

The `login` method returns true if it was successful or false if not.

3.1.3 Making Requests

To make requests, call the appropriate function from the following list:

- `do_get(`*`$uri_fragment`*`, `*`$content_ref`*`)`
- `do_delete(`*`$uri_fragment`*`)`
- `do_post(`*`$uri_fragment`*`, `*`$content_ref`*`)`
- `do_put(`*`$uri_fragment`*`, `*`$content_ref`*`)`

These functions perform a GET, DELETE, POST and PUT request respectively. The argument *$uri_fragment* is the portion of the URI immediately after the `/api/2.0` part, including a leading slash. For the POST and PUT requests, *$content_ref* is a reference to a hash containing key/value pairs to use for the POST or PUT request. For a GET request, *$content_ref* is optional; if present, it is converted into a query string and appended to the URL. Note that *$content_ref* is a native Perl hash.

For a GET request, a *$content_ref* of:

```
{  foo => 'b/ar', quux => 2345 }
```

would be converted to a query string like:

```
?foo=b%2far&quux=2345
```

The return values of the functions are:

- `do_get` returns a Perl data structure representing the returned information. If the request fails, returns `undef`.
- `do_delete` returns true if the request was successful or false if it was not.
- `do_post` returns true if the request was successful or false if it was not.
- `do_put` returns true if the request was successful or false if it was not.

3.1.4 Error Return

The method `$api->get_last_error()` returns a textual error message related to the last API call that failed. If one of the `do_` methods fails, you can call `get_last_error` to find out why.

`CanIt::API::Client` has a few other seldom-used methods; see the man page for details.

3.1.5 Value Return

The method `$api->get_last_value()` returns the Perl data structure from the most recent `do_get`, `do_put` or `do_post` method. Since `do_put` and `do_post` only return success/failure indications, you must use this method to access the actual returned data from the PUT or POST.

If the previous `do_get`, `do_put` or `do_post` method did not succeed, this method returns `undef`.

3.1.6 Example

The following example marks Incident 01D88b3rr as spam:

```
use CanIt::API::Client;

my $api = CanIt::API::Client->new({
    base     => 'http://canit.example.com/api',
    version  => '2.0',
});

$api->login('admin', 's3cr3tpw6');

my $incident = $api->do_get('/incident/01D88b3rr');
if ($incident) {
    $api->do_post('/incident/01D88b3rr',
            { status => spam, resolution => discard });
    } else {
        print STDERR "Failed: " . $api->get_last_error() . "\n";
    }
    $api->logout();
```

3.2 PHP Library: CanItAPIClient

The PHP library is called `CanItAPIClient`. You can install the `/usr/share/canit/canit-api-client.php` file on any machine with PHP5 installed. The file requires the "cURL" PHP extension. Most systems provide packaged versions of the cURL extension. If yours does not, you can obtain the extension from the PHP web site.

`CanItAPIClient` provides an object-oriented programming interface. In general, you create a `CanItAPIClient` object and then call methods on it.

3.2.1 Constructor

To create a `CanItAPIClient` object, call the `new` method with a single URI argument. This argument is the base URL of the API server, including the version number (currently 2.0). For example:

```
    require_once("canit-api-client.php");
    $api = new CanItAPIClient('http://canit.example.com/canit/api/2.0');
```

3.2.2 Logging In and Out

After creating the `CanItAPIClient` object, you need to log in to the API server. When you have finished, you should log out. For example:

```
    $api->login('username', 'password');
    # Do things with $api
    $api->logout();
```

The `login` method returns true if it was successful or false if not.

3.2.3 Making Requests

To make requests, call the appropriate function from the following list:

- `do_get(`*`$uri_fragment`*`, `*`$content`*`)`
- `do_delete(`*`$uri_fragment`*`)`
- `do_post(`*`$uri_fragment`*`, `*`$content`*`)`

- `do_put(`*`$uri_fragment`*`, `*`$content`*`)`

These functions perform a GET, DELETE, POST and PUT request respectively. The argument *$uri_fragment* is the portion of the URI immediately after the `/api/2.0` part, including a leading slash. For the GET, POST and PUT requests, *$content* is an array containing key/value pairs to use for the request. (*$Content* is optional for a GET request. If it is supplied, it is converted to a query string and appended to the URL.)

The return values of the functions are:

- `do_get` returns a PHP array representing the returned information. If the request fails, returns `NULL`.
- `do_post` returns an array of data returned by the server upon success or `NULL` on failure.
- `do_put` returns an array of data returned by the server upon success or `NULL` on failure.
- `do_delete` *always* returns `NULL`. You need to call the `succeeded` method to see if the DELETE was successful.

3.2.4 Error Return

The method `$api->succeeded()` returns true if the last API call was successful. It returns false if it was not.

The method `$api->get_last_error()` returns a textual error message related to the last API call that failed. If one of the `do_` methods fails, you can call `get_last_error` to find out why.

3.2.5 Example

The following example marks Incident 01D88b3rr as spam:

```
require_once("canit-api-client.php");
$api = new CanItAPIClient('http://canit.example.com/canit/api/2.0');

$api->login('admin', 's3cr3tpw6');

$incident = $api->do_get('/incident/01D88b3rr');
if ($api->succeeded()) {
    $api->do_post('/incident/01D88b3rr',
        array('status' => 'spam',
              'resolution' => 'discard'));
} else {
    print "Failed: " . $api->get_last_error() . "\n";
}
$api->logout();
```

3.3 Python Library: CanItAPIClient

The Python library is called `CanItAPIClient`. You can install the `/usr/share/canit/CanItAPIClient.py` file on any machine with Python installed. You will also need the following Python modules:

- `pycurl`
- `json`
- `urllib`
- `StringIO` (This is in core Python.)

`CanItAPIClient` provides an object-oriented programming interface. In general, you create a `CanItAPIClient` object and then call methods on it.

3.3.1 Constructor

To create a `CanItAPIClient` object, call the `CanItAPIClient` method with a single URI argument. This argument is the base URL of the API server, including the version number (currently 2.0). For example:

```
import CanItAPIClient
api = CanItAPIClient.CanItAPIClient('http://canit.example.com/canit/api/2.0')
```

3.3.2 Logging In and Out

After creating the `CanItAPIClient` object, you need to log in to the API server. When you have finished, you should log out. For example:

```
api.login('username', 'password')
# Do things with api
api.logout()
```

The `login` method returns true if it was successful or false if not.

3.3.3 Making Requests

To make requests, call the appropriate function from the following list:

- `do_get(`*`uri_fragment`*`, `*`content`*`)`
- `do_delete(`*`uri_fragment`*`)`
- `do_post(`*`uri_fragment`*`, `*`content`*`)`

- `do_put(`*`uri_fragment, content`*`)`

These functions perform a GET, DELETE, POST and PUT request respectively. The argument *uri_fragment* is the portion of the URI immediately after the `/api/2.0` part, including a leading slash. For the GET, POST and PUT requests, *content* is a dictionary containing key/value pairs to use for the request. (*Content* is optional for a GET request. If it is supplied, it is converted to a query string and appended to the URL.)

The return values of the functions are:

- `do_get` returns a dictionary or array representing the returned information. If the request fails, returns `None`.
- `do_post` returns a dictionary or array of data returned by the server upon success or `None` on failure.
- `do_put` returns a dictionary or array of data returned by the server upon success or `None` on failure.
- `do_delete` *always* returns `None`. You need to call the `succeeded` method to see if the DELETE was successful.

3.3.4 Error Return

The method `api.succeeded()` returns true if the last API call was successful. It returns false if it was not.

The method `api.get_last_error()` returns a textual error message related to the last API call that failed. If one of the `do_` methods fails, you can call `get_last_error` to find out why.

3.3.5 Example

The following example marks Incident 01D88b3rr as spam:

```
import CanItAPIClient
api = CanItAPIClient.CanItAPIClient('http://canit.example.com/canit/api/2.

api.login('admin', 's3cr3tpw6')

incident = api.do_get('/incident/01D88b3rr')
if (api.succeeded()):
    api.do_post('/incident/01D88b3rr',
                {'status' : 'spam', 'resolution' : 'discard'})
else:
    print "Failed: " + api.get_last_error()

api.logout()
```

3.4 Command-line tool canit-api-wrapper

The command-line tool `canit-api-wrapper` allows you to access the CanIt API from UNIX shell scripts. Using the wrapper involves the following steps:

1. Use `canit-api-wrapper` to log in to the API. This returns a *cookie* that is used in subsequent invocations.
2. Invoke `canit-api-wrapper` (using the cookie from Step 1) to perform a series of desired API calls.
3. Invoke `canit-api-wrapper` with the special `logout` argument to log out of the API.

3.4.1 Logging In and Obtaining a Cookie

To log in to the API, invoke `canit-api-wrapper` as follows:

```
C=`canit-api-wrapper --url=URL login user pass`
```

In the command-line above:

- ***URL*** is the base URL of the API server, *including* the **`/api/2.0`** component.
- ***user*** and ***pass*** are the username and password with which to log in. If you omit ***pass***, you will be prompted for the password. If you omit both ***user*** and ***pass***, you will be prompted first for the username and then for the password. Note: The interactive prompting only works if both standard input and standard error are associated with a terminal. If not, then omitting either the username or the password causes the command to fail.

For example, you might invoke `canit-api-wrapper` as follows. (The entire command should be entered on a single line.)

```
C=`canit-api-wrapper --url=https://canit.example.org/canit/api/2.0 login
admin s3cr3t`
```

3.4.2 Making API calls

To make API calls, invoke `canit-api-wrapper` with the cookie returned by the login. Use “get”, “put”, “post” or “delete” to perform a GET, PUT, POST or DELETE call, respectively. For those calls that take arguments, add them in the form VAR=VAL after the API call. Here are some examples:

```
# List all users
canit-api-wrapper --cookie=$C get /realm/@@/users

# Add a user
canit-api-wrapper --cookie=$C put /realm/@@/user/foo \
    email=devnull@example.com password=wookie

# Look at that user: Output is JSON
canit-api-wrapper --cookie=$C get /realm/@@/user/foo

# Look at that user: Output is human-readable
canit-api-wrapper --pretty --cookie=$C get /realm/@@/user/foo

# Change the user's email address
canit-api-wrapper --cookie=$C post /realm/@@/user/foo \
    email=foo@example.com

# Delete the user
canit-api-wrapper --cookie=$C delete /realm/@@/user/foo

# Log out
canit-api-wrapper --cookie=$C logout
```

3.4.3 Command-Line Options

`canit-api-wrapper` takes the following command-line options:

- **--url=URL** or **-u URL** — specify the URL of the CanIt API server. Used only by the `login` argument.
- **--cookie=C** or **-c C** — specify the cookie for all API calls once logged in.
- **--pretty** or **-p** — for GET calls, print the results in human-readable format rather than JSON.

3.4.4 Return Code

If the API call succeeded, `canit-api-wrapper` exits with exit code 0. If the API call failed, `canit-api-wrapper` exits with exit code 1. It also prints an error message to standard error.

Appendix A

The CanIt-Domain-PRO License

READ THIS LICENSE CAREFULLY. IT SPECIFIES THE TERMS AND CONDITIONS UNDER WHICH YOU CAN USE CANIT-DOMAIN-PRO

This license may be revised from time to time; any given release of CanIt-Domain-PRO is licensed under the license version which accompanied that release.

CanIt-Domain-PRO is distributed in source code form, but it is not Free Software or Open-Source Software. Some CanIt-Domain-PRO components are Free Software or Open-Source, and we detail them below:

The following files may be redistributed according to the licenses listed here. An asterisk (*) in a file name signifies a version number; the actual file will have a number in place of the asterisk.

File	License
`src/Archive-Tar-*.tar`	Perl License
`src/Config-Tiny-*.tar`	Perl License
`src/DBD-Pg-*.tar`	Perl License
`src/DBI-*.tar`	Perl License
`src/Data-ResultSet-*.tar`	Perl License
`src/Data-UUID-*.tar`	Perl License
`src/Digest-MD5-*.tar`	Perl License
`src/Digest-SHA1-*.tar`	Perl License
`src/File-Spec-*.tar`	Perl License
`src/File-Temp-*.tar`	Perl License
`src/HTML-Parser-*.tar`	Perl License
`src/HTML-Tagset-*.tar`	Perl License
`src/IO-Zlib-*.tar`	Perl License
`src/IO-stringy-*.tar`	Perl License
`src/Log-Syslog-Abstract-*.tar`	Perl License
`src/MIME-Base64-*.tar`	Perl License
`src/MIME-tools-*.tar`	Perl License
`src/Mail-SPF-Query-*.tar`	Perl License
`src/Mail-SpamAssassin-*.tar`	Apache License, Version 2.0
`src/MailTools-*.tar`	Perl License

File	**License**
`src/Module-Pluggable-Tiny-*.tar`	Perl License
`src/Net-CIDR-Lite-*.tar`	Perl License
`src/Net-DNS-*.tar`	Perl License
`src/Net-IP-*.tar`	Perl License
`src/Time-HiRes-*.tar`	Perl License
`src/TimeDate-*.tar`	Perl License
`src/URI-*.tar`	Perl License
`src/YAML-Syck-*.tar`	Perl License
`src/clamav-*.tar`	GPLv2
`src/p0f-*.tar`	GPLv2
`src/libwww-perl-*.tar`	Perl License
`src/mimedefang-*.tar`	GPLv2

ALL REMAINING FILES IN THIS ARCHIVE (referred to as ”CanIt-Domain-PRO”) ARE DISTRIBUTED UNDER THE TERMS OF THE CANIT LICENSE, WHICH FOLLOWS:

THE CANIT LICENSE

1. CanIt-Domain-PRO is the property of AppRiver LLC (”AppRiver”) This license gives you the right to use CanIt-Domain-PRO, but does not transfer ownership of the intellectual property to you.

2. CanIt-Domain-PRO is licensed with a limit on the number of allowable protected domains or mailboxes. This limit is called ”the Usage Limit”.

   CanIt-Domain-PRO usage may be purchased on a yearly basis, or you may purchase a perpetual license.

3. You may use CanIt-Domain-PRO up to the Usage Limit you have purchased. If you have purchased yearly usage, you may continue to use CanIt-Domain-PRO until your purchased usage time expires, unless you purchase additional time. If you have purchased a perpetual license, you may continue to use CanIt-Domain-PRO indefinitely, providing you do not violate this license.

   If you have purchased yearly usage, you may exceed your purchased limit by up to 10% until the yearly renewal date, at which time you must purchase a sufficient limit for the increased number of domains or mailboxes.

   If you have purchased a perpetual license, or wish to increase your usage more than 10% above your paid-up limit, you must purchase the additional usage within 60 days of the increase.

4. You may examine the CanIt-Domain-PRO source code for education purposes and to conduct security audits. You may hire third-parties to audit the code providing you first obtain permission from AppRiver. Such permission will generally be granted providing the third-party signs a non-disclosure agreement with AppRiver.

5. You may modify the CanIt-Domain-PRO source code for your own internal use, subject to the restrictions in Paragraph 9 below. However, if you do so, you agree that AppRiver is released

from any obligation to provide technical support for the modified software. If you wish your modifications to be incorporated into the mainstream CanIt-Domain-PRO release, you agree to transfer ownership of your changes to AppRiver.

6. You may make backups of CanIt-Domain-PRO as required for the prudent operation of your enterprise.

7. You may not redistribute CanIt-Domain-PRO in source or object form, nor may you redistribute modified copies of CanIt-Domain-PRO or products derived from CanIt-Domain-PRO.

8. If you violate this license, your right to use CanIt-Domain-PRO terminates immediately, and you agree to remove CanIt-Domain-PRO from all of your servers.

9. Restrictions on modification:

   (a) Notwithstanding Paragraph 5, you may not make changes to CanIt-Domain-PRO or your software environment which would allow CanIt-Domain-PRO to run without a valid License Key as issued by AppRiver. You also agree not to set back the time on your server to artificially extend the validity of a License Key, or do anything else which would artificially extend the validity of a License Key.

   (b) You may modify the Web-based interface only providing you adhere to the following restrictions:

   (c) At the bottom of every CanIt-Domain-PRO web page, the following text shall appear, in a size, color and font which are clearly legible:

   Powered by CanIt-Domain-PRO (Version x.y.z) from AppRiver LLC

   where x.y.z is the product version. In addition, "CanIt-Domain-PRO" shall be a clearly-marked hypertext link to `https://www.roaringpenguin.com/powered-by-canit.php`

   (d) You may not include elements on the CanIt-Domain-PRO Web interface that require plug-ins (such as, but not limited to, Macromedia Flash, RealPlayer, etc.) to function.

   (e) You may not include Java applets on the CanIt-Domain-PRO Web interface.

   (f) If you include JavaScript on the Web interface, you shall ensure that the interface functions substantially unimpaired in a browser with JavaScript disabled.

   (g) You shall not include browser-specific elements on the Web interface. You shall ensure that the Web interface functions substantially unimpaired on the latest versions of the following browsers:

      - Internet Explorer for Windows
      - Mozilla for Windows
      - Mozilla for Linux
      - Konqueror for Linux

   (h) You may not include banner ads on the CanIt-Domain-PRO Web interface.

10. Disclaimer of Warranty (Virus-Scanning)

NOTE: ALTHOUGH CANIT-DOMAIN-PRO IS DISTRIBUTED WITH CLAM ANTIVIRUS, WE DO NOT MAKE ANY REPRESENTATIONS AS TO ITS EFFECTIVENESS AT STOPPING VIRUSES. APPRIVER HEREBY DISCLAIMS ALL WARRANTY ON THE ANTIVIRUS CODE INCLUDED WITH CANIT-DOMAIN-PRO, OR WHICH INTERFACES TO CANIT-DOMAIN-PRO. WE ARE NOT RESPONSIBLE FOR ANY VIRUSES THAT MIGHT EVADE A VIRUS-SCANNER INTEGRATED WITH CANIT-DOMAIN-PRO.

11. Disclaimer of Warranty (Time-Critical Mass Mailings)

    CANIT-DOMAIN-PRO IS NOT DESIGNED FOR TIME-CRITICAL EMERGENCY MASS MAILINGS. AN EMERGENCY MASS-MAILING MAY OVERLOAD CANIT-DOMAIN-PRO AND CAUSE DELAYS. APPRIVER HEREBY DISCLAIMS ALL WARRANTY ON THE ABILITY OF CANIT-DOMAIN-PRO TO DELIVER MASS MAILINGS IN A TIMELY FASHION. IF YOU REQUIRE EMERGENCY MASS-MAILINGS YOU MUST CONFIGURE THEM TO BYPASS THE CANIT-DOMAIN-PRO FILTER.

A.1 THE CANIT DATA LICENSE

AppRiver makes available certain data that are used by CanIt. This license covers the RPTN Bayes data and the AppRiver RBLs. The data are owned by AppRiver and their use is licensed under the following terms:

1. You may update the RPTN data once per day per AppRiver download username. AppRiver reserves the right to cut off downloads if more than one download per day per username is attempted.
2. You may use the RPTN data only in conjunction with your properly-licensed CanIt installation.
3. You may not redistribute the RPTN data.
4. If your support term expires, you lose the right to use RPTN data for any purpose whatsoever.
5. You may make use of the AppRiver RBLs from within CanIt. You may not query them with any other software.
6. You may use the AppRiver RBLs only in conjunction with your properly-licensed CanIt installation.
7. You may not redistribute the AppRiver RBL data.
8. If your support term expires, you lose the right to use the AppRiver RBLs.