

The Cascabel Framework

Tracking issues
Version 1.7

Tommy M. McGuire

This manual is for Cascabel (version 1.7, 22 April 2003), which is a framework for creating issue tracking systems.

Copyright © 2003, The University of Texas at Austin.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with no Invariant Sections, with no Front-Cover Texts and with no Back-Cover Texts.

The Cascabel Framework is distributed under the terms of the GNU General Public License: This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

Table of Contents

1	Introduction to Cascabel	1
1.1	Basic operation: managing tickets, sending mail	1
1.2	Requirements: Python, MySQL, etc.	1
1.3	Setting up Cascabel	2
1.4	Security issues	2
1.5	The history of Cascabel	3
2	Database Structure	4
2.1	The queues table	4
2.2	The allowedstatus table	5
2.3	The mailfilters table	5
2.4	The log table	5
2.5	The synonyms table	6
2.6	The staff and queuestaff tables	6
2.7	The users table	7
2.8	The ticket tables	7
3	Queue Operations	9
3.1	Database operations	9
3.2	Basic queue operations	10
3.3	Mail support operations	10
3.4	Ticket operations	11
3.5	Staff operations: mkstaff, rmstaff, etc.	12
3.6	User operations: mkuser, etc.	12
3.7	Log operations: getlog, log	13
4	Mail Handling	14
4.1	Sending mail: send and resend.	14
4.2	Mail header manipulation	14
4.3	Message body	15
4.4	The MailHandler	16
5	Utility Functions	18
6	CGI support	19
6.1	Machine.php: Session-based state machine	19
6.1.1	Machine: The state machine engine	20
6.1.2	Transition: Persistent transitions	20
6.1.3	TransitionKey: Identifying transitions	21
6.2	TemplateCache.php: Reusable templates	21

7	Scripts: cascabel-admin, etc.	22
7.1	cascabel-admin	22
7.2	cascabel-config	22
7.3	MySQL makecascabel script	22
7.4	MySQL initcascabel script	22
7.5	Example scripts	22
8	Cascabelwrapper and the Password File	23
	Concept, Function, and Table Indices	24
	Concept Index	24
	Function Index	25
	Table Index	27

1 Introduction to Cascabel

Issues, be they bugs, requests, purchase orders, donations, or whatever, need tracking. Unfortunately, no one seems to make little radio collars for them. As a result, we need to create programs that maintain information about issues. That is what Cascabel is for.

1.1 Basic operation: managing tickets, sending mail

Cascabel is a ticket tracking system. It handles tickets, which maintain information about specific issues. A ticket may be something like a purchase order, a bug report, or a request for support. The issue is assigned an identifier, or ticket id (an integer), and all communications about the issue (that pass through the Cascabel system, obviously) are logged and potentially forwarded to everyone associated with the ticket. Also, status information about the ticket is kept, such as its state (helpfully called the “status” in Cascabel) and the times it was created and last updated.

These tickets are organized in queues. The tickets in a queue should be related, like purchase requests, bug reports, and so on. The people who handle the issues, called staff members, are associated with the particular queues. In most Cascabel queues (depending on the interfaces built with the Cascabel framework, of course), the staff members for a queue can view and edit the information about every ticket in the queue.

The life-cycle of a ticket is something like:

1. The ticket is created, either by:
 - Sending a random piece of mail to a queue’s incoming address.
 - A queue-specific interface, such as a CGI, command-line, or GUI program.

The individual creating the ticket, assuming they are not staff members, is a user. Other users can be associated with the ticket; there can be any number of users associated with any ticket.

2. The users and the staff members communicate about the issue, hopefully using email through Cascabel’s incoming mail alias for the queue. This mail is logged to the ticket.
3. Staff members can also change the status information about the ticket.
4. Eventually, the issue is resolved, and the ticket is marked “resolved”. Most interfaces do not display resolved tickets unless specifically asked, but the information is kept for future reference.

1.2 Requirements: Python, MySQL, etc.

Cascabel requires other packages to function:

- **MySQL**.
- Python 2.0 or later, with the following modules:
 - mxDateTime, from [eGenix.com](http://egenix.com). Tested with egenix-mx-base 2.0.3.
 - MySQLdb, from <http://sourceforge.net/projects/mysql-python>. Tested with 0.9.2.

- Sendmail, or another mail transport agent which provides a `sendmail` program for sending mail.
- PHP, for CGI interfaces. Tested with 4.2.2.
- A web server, for the CGI interfaces.
- Other programs, as needed by particular queue interfaces.

1.3 Setting up Cascabel

Cascabel is configured using the `configure` script included with the distribution. Check the ‘README’ file in the distribution for up-to-date information. In general, in addition to the usual `configure` arguments, you will need to supply:

- A user name, which will own the secure information needed to contact the database, and the `setuid` program that controls access to that file.
- A default email domain, used to complete email addresses when only a user name is supplied.

Once its requirements are met and Cascabel is configured, built, and installed (in the normal, `configure-make-make install` way), you need to:

1. Create (See [Section 7.3 \[makecascabel\], page 22.](#)) and set up (See [Section 7.4 \[initcascabel\], page 22.](#)) the Cascabel database in the MySQL DBMS.
2. Create and set up the queue-specific database table and table entries. See [Section 7.5 \[Examples\], page 22.](#)
3. Create queue interface programs, either CGI, command-line, or GUI. See [Section 7.5 \[Examples\], page 22.](#)
4. Set up the queue’s mail aliases. See [Section 7.5 \[Examples\], page 22.](#)

1.4 Security issues

The information about tickets may be more or less confidential. Cascabel keeps all of the information in a MySQL database, and keeps this database secured by a single SQL password. The password, along with the other information needed to contact the database, is kept in a file that is protected by Unix file permissions—only the special user identified when Cascabel is installed is allowed to read the file. In this document, the user account is called the Cascabel user.

Cascabel programs which need to contact the database must be run `setuid` to the Cascabel user. Since many Unixes such as Unix do not allow Python scripts to be `setuid`, a wrapper program, written in C, is used to call the actual script. This program selects the script to run based on its `argv[0]` argument—a symbolic link, named after the script in your `PATH`, to the wrapper program makes the connection. The wrapper program looks in a specific directory (created during Cascabel’s installation) for the script to actually execute.

Thus, Cascabel tries to protect the information about tickets, but can only do so with the help of the interfaces. Those programs make the choices about what information to expose to whom.

Note that the scripts connect to the MySQL database over a network—this connection may be insecure.

1.5 The history of Cascabel

In about 1996, the UT CS Department needed a system to track purchase requests. We looked around and came up with req 1.2.7, by Remy Evard. Req was “an email-based request tracking system.” It satisfied our needs (mostly) for a number of years.

Later, we decided we needed a system to track shop (hardware support) requests. Req did not handle multiple queues, but a possible replacement system, RUST, did. RUST, however, was poorly written. (For example, its original file locking system meant a command paused for at least a second every time it ran. Also, RUST had about three subsystems for sending mail, only one of which called sendmail—the others would not handle MX DNS records nor queueing mail for later delivery.) It did not replace req, but we used it for shop requests for quite a while.

After trying to clean up RUST, I decided it would be simpler to just write my own damn tracking system, which became Cascabel. Rather than using flat files, like req and RUST, I chose to base the design around a SQL database, picking MySQL. The combination of Python as an implementation language and MySQL, plus some serious napkin-designing made Cascabel relatively small and simple, as well as flexible.

When we wanted to add CGI interfaces for the queues, I decided that Python was probably not the right choice—start-up latencies, don’t you know. I chose PHP and simply re-created the Cascabel queue operations in PHP, which was fast and easy.

Nowadays, there are other tracking systems, such as RT, which provide much better functionality and support than the options that I looked at before starting Cascabel. If I were starting now, I would probably use RT. On the other hand, Cascabel has been very satisfactory, and is a bit more flexible than RT.

2 Database Structure

Cascabel stores all of the data for all of the queues (including as much metadata as is convenient) in a SQL database. The currently supported database manager is MySQL; although it would be some work to add support for other DBMS, Cascabel is designed to allow for that and it should be reasonable trivial.

Cascabel is intended to be very flexible, particularly regarding the information stored about tickets by differing queues. Creating each queue requires adding at least one table to the database. This new table contains the individual ticket status information for the tickets in the queue.

This section describes each of the tables used by Cascabel. (It does not show the indices used in the tables. For that, check the `initcascabel` script. See [Section 7.4 \[initcascabel\]](#), page 22.

2.1 The queues table

The queues table contains much of the metadata about each queue managed by Cascabel. The fields are:

<code>'queue'</code>	The queue name.
<code>'incoming'</code>	The email address which delivers mail to Cascabel for the queue.
<code>'outgoing'</code>	The email address to which Cascabel sends new tickets and ticket updates.
<code>'subjpattern'</code>	A regular expression identifying a queue and a ticket, from the subject of a mail message.
<code>'subjformat'</code>	The Python <code>printf</code> -like string used to create outgoing subject headers.
<code>'receipt'</code>	The template mail message sent back to someone creating a new ticket.
<code>'remoteaction'</code>	A flag indicating whether mail messages should be examined for remote X-Cascabel commands.

The SQL schema for the table is:

```
CREATE TABLE queues (
    queue CHAR(64) NOT NULL PRIMARY KEY,
    incoming VARCHAR(64),
    outgoing VARCHAR(64),
    subjpattern CHAR(64),
    subjformat CHAR(64),
    receipt TEXT,
    remoteaction ENUM('True', 'False')
);
```

2.2 The allowedstatus table

The allowedstatus provides more metadata about each queue: The allowable values of the status field.

The SQL schema for the table is:

```
CREATE TABLE allowedstatus (
    queue CHAR(64) NOT NULL,
    status CHAR(64) NOT NULL
);
```

‘queue’ The queue name

‘status’ An allowed status value.

2.3 The mailfilters table

The mail filters used to variously ignore incoming mail messages.

The SQL schema for the table is:

```
CREATE TABLE mailfilters (
    queue CHAR(64),
    pattern VARCHAR(255),
    type ENUM('Ignore', 'Nobody', 'Reject') DEFAULT 'Ignore'
);
```

‘queue’ The queue name. A null queue name indicates the pattern applies to all queues.

‘pattern’ The pattern matched against the incoming mail message’s source address.

‘type’ One of “Ignore”, “Nobody”, or “Reject”. See the sections on mail handling for the details about each.

By default, the ‘mailer-daemon’ address is Ignored.

```
INSERT INTO mailfilters SET pattern='mailer-daemon';
```

2.4 The log table

All actions and mail messages regarding any ticket in the queue are permanently recorded in the log table, under the queue and ticket id.

The SQL schema for the table is:

```
CREATE TABLE log (
    queue CHAR(64) NOT NULL,
    id INTEGER NOT NULL,
    timestamp TIMESTAMP,
    entry TEXT NOT NULL
);
```

‘queue’ The queue name

‘id’ The ticket id
‘timestamp’
 The time the entry was logged
‘entry’ The text of the mail message or other log entry

2.5 The synonyms table

It is possible for a ticket to be either merged with another ticket in the same queue or (although this is currently unimplemented) moved to another queue. The synonyms table handles the mapping.

The SQL schema for the table is:

```
CREATE TABLE synonyms (
    squeue CHAR(64) NOT NULL,
    sid INTEGER NOT NULL,
    dqueue CHAR(64) NOT NULL,
    did INTEGER NOT NULL
);
```

‘squeue’
‘sid’ The source queue and ticket id.
‘dqueue’
‘did’ The destination queue and ticket id.

2.6 The staff and queuestaff tables

The staff are specially singled-out users: They are assumed to get mail about tickets either automatically from forwarded messages or through queue-specific processing. As a result, Cascabel tries very hard to make sure that staff addresses for a queue are not inserted into the list of users for any ticket in the queue. They also have a password assigned, for authenticated CGI applications. For Python command-line programs, the userid of the person running the program is added to the default mail domain and the result is used to look for an entry in the staff table.

A person can be a staff member of many queues. The staff record is always the same; the queuestaff table is used to indicate which queues an address is a staff member for.

The SQL schema for the staff table is:

```
CREATE TABLE staff (
    address VARCHAR(255) NOT NULL,
    password VARCHAR(255),
    staffid INTEGER NOT NULL AUTO_INCREMENT
);
```

‘address’ The staff member’s email address
‘password’
 The staff member’s Cascabel password

‘staffid’ An identifier used to join the two tables.

The SQL schema for the queuestaff table is:

```
CREATE TABLE queuestaff (
    queue CHAR(64) NOT NULL,
    staffid INTEGER
);
```

‘queue’ The queue name

‘staffid’ The staff member’s identifier.

2.7 The users table

The users table is used to record the user email addresses associated with each ticket in every queue.

The SQL schema for the table is:

```
CREATE TABLE users (
    queue CHAR(64) NOT NULL,
    id INTEGER NOT NULL,
    address VARCHAR(255) NOT NULL
);
```

‘queue’ The queue name

‘id’ The ticket id

‘address’ The email address

2.8 The ticket tables

Every ticket needs a home. Or, at least, some place to hang its status information. In Cascabel, every queue has its own table of ticket status information, although each of these tables must have a minimal set of standard fields. This approach is the key to Cascabel’s flexibility—some queues will need very specialized information about each ticket and rather than trying to anticipate all of the possible fields, Cascabel allows them to be extended from a base set as needed.¹

Each table should be called *queueticket*; i.e. the queue name followed by the word “ticket”.

The SQL schema below describes the table for the “cascabel” queue, which is an example and testing queue included in the basic Cascabel configuration. It illustrates the minimum set of columns.

The SQL schema for the table is:

¹ A DBMS capable of table-ish inheritance would make this more natural, probably. It might also allow a closer relationship between queues. On the other hand, it doesn’t seem needed; Cascabel’s database manipulation is pretty simplistic.

```
CREATE TABLE cascabelticket (  
    id INTEGER NOT NULL AUTO_INCREMENT PRIMARY KEY,  
    status CHAR(65) NOT NULL DEFAULT 'open',  
    subject VARCHAR(255),  
    created DATETIME,  
    updated DATETIME  
);
```

'id' The ticket id

'status' The ticket status. The default status will be used for new tickets.

'subject' The subject line for the ticket; a short summary.

'created' The timestamp when the ticket was created.

'updated' The last time the ticket was modified.

Additional fields are added as needed and processed by the queue-specific code which uses the framework. There should be some examples around, near wherever you found Cascabel.

It would be fairly trivial to add additional tables associated with queues, assuming you avoid naming conflicts. Thus, for example, a queue could have tickets of different types, where each type of ticket had its own set of extra fields. The complexity of the application, however, might make extra tables unworkable.

3 Queue Operations

The major operations of the framework revolve around the connection to the database. These operations include requesting or updating information about queues, tickets, users, or staff members.

Not all of the operations are supplied by all of the interfaces. For example, since the PHP interface handles unauthenticated web interactions, it does provide the login and changepassword operations. The Python interface currently does not since it is used for already-authenticated command-line interactions. On the other hand, the PHP interface does not provide the mail support operations or some of the other operations that are not needed by web interfaces. And, of course, neither interface is really complete—some operations have simply not been implemented.

These operations are currently provided by the ‘`CascabelMySQL.py`’, ‘`CascabelMySQLExpressions.py`’, and ‘`CascabelMySQL.php`’, with the interface modules for accessing them being ‘`Cascabel.py`’ and ‘`Cascabel.php`’.

3.1 Database operations

There are two ways of getting ticket information from the database. First, if you have the ticket id, all the information can be recovered directly and quickly. Second, if you wish to find one or more tickets matching some criteria, you will need to create a predicate describing the criteria. These predicates can be given to the `lstickets` and `getlog` methods, which return a list of ticket records and log entries, respectively. The predicates are built of a thin layer of functions over the underlying SQL. The database SQL operators that Cascabel provides are:

And *predicate* [*predicate*]... [Function]

Or *predicate* [*predicate*]... [Function]

And and **Or** are boolean operators and take either multiple arguments (Python) or an array of arguments (PHP) and combine them.

Not *predicate* [Function]

Not is the unary boolean operator.

IsNull *field* [Function]

Is true when **field** is null (i.e. empty, not having a value).

True [Function]

False [Function]

True and **False** are constants.

Equal <i>field value</i>	[Function]
NotEqual <i>field value</i>	[Function]
AtMost <i>field value</i>	[Function]
AtLeast <i>field value</i>	[Function]
LessThan <i>field value</i>	[Function]
MoreThan <i>field value</i>	[Function]
Rlike <i>field pattern</i>	[Function]

Compare *field* with *value*. The *field* should be a string giving the column name and the *value* is an appropriate value (i.e. an integer, or a string). The *pattern* for **Rlike** is a regular expression.

3.2 Basic queue operations

The basic queue operations provide meta-data about a queue. The normal meta-data consist of the information from the queues table and the list of allowed status for each queue from the allowedstatus table.

Connection [<i>host user password</i>] [<i>debug</i>]	[Function]
--	------------

Open a connection to the database (on *host*, using *user* and *password* to make the connection). The arguments *host*, *user*, and *password* are optional; if they are not provided, the Cascabel ‘password’ file is consulted for the information. See [Chapter 8 \[Cascabelwrapper\]](#), page 23. If *debug* is greater than 1, the SQL statements before value substitution are printed on stderr; if greater than 2, the statements after value substitution are printed. The *debug* argument defaults to 0.

getqueue <i>queue</i>	[Method on Connection]
------------------------------	------------------------

Return the information from the queues table associated with *queue*.

getqueues	[Method on Connection]
------------------	------------------------

Return all of the queues table information about every queue in the database.

lsstatus <i>queue</i>	[Method on Connection]
------------------------------	------------------------

Return the information from the allowedstatus table, with the allowed status for *queue*.

3.3 Mail support operations

Filters are regular expressions matched against email addresses. The filter types are “Nobody”, “Ignore”, and “Reject”.

- Mail from “Ignore” filtered addresses is completely ignored.
- “Nobody” filtered addresses are not added to the users for a ticket, and do not get ticket receipts.
- Mail from “Reject” addresses is passed on to the queue distribution address, but not otherwise processed by Cascabel.

lsfilters *queue* [*type*] [Method on Connection]
 Return information from the mailfilters table matching either the *queue* or with a null queue. If the *type* is present, the filters are limited to that *type*.

3.4 Ticket operations

The ticket operations include creating a ticket, retrieving the information about a ticket or about all the tickets in a queue, updating a ticket, and merging tickets.

mkticket *queue info* [Method on Connection]
 Create a new ticket in *queue* with the fields specified in *info*. Returns the id number of the new ticket.

getticket *queue id* [Method on Connection]
 Return the status information about ticket *id* in *queue*. Generally, the status information will be a dictionary where the keys are columns from the database table. The *getticket* method checks the synonyms table to find the actual ticket information.

lstickets *queue predicate* [*order*] [*user*] [Method on Connection]
 Return a list of ticket status information. The tickets will be from *queue*, match *predicate*, be in order according to the *order* database column. If *user* is true, the ticket table will be joined with the users table, to find tickets associated with a given user. The user's email address should be part of the *predicate*. The *lstickets* method does not check the synonyms table; this method is used to generate a list of *existing* tickets.

setticket *queue id info* [Method on Connection]
 Update the status information for ticket *id* in *queue*. The *info* should be a dictionary mapping from database columns to new values. This method does check that an updated status is allowed by the queue.

update *queue id* [*time*] [Method on Connection]
 Set the last-updated time of ticket *id* in *queue* to *time*, if present, or the current time, otherwise.

mvticket *source-queue source-id dest-queue dest-id* [Method on Connection]
 Merge ticket *source-id* from *source-queue* into *dest-id* in *dest-queue*. *dest-id* should already exist.

3.5 Staff operations: `mkstaff`, `rmstaff`, etc.

A staff member is an email address that is allowed to edit the status of a ticket. As a result, they need to be tracked more closely than the normal user addresses.

mkstaff *queue address [password]* [Method on Connection]

1. Adds an entry to the staff table for *address*, if one does not already exist. The entry uses *password* if one is provided; otherwise the user name portion of the *address* is used.
2. If *queue* is non-null, adds an entry to the queuestaff table for *address*.

lsstaff [*queue*] [*address*] [Method on Connection]

Return a list of dictionaries with keys “address”, “password”, and “queue” for staff entries. If the *queue* is specified, the list is limited to that queue; if *address*, that staff address. Giving both provides an easy test for whether the *address* is a staff member for *queue*. If neither is given, `lsstaff` generates a list of all staff addresses.

rmstaff *address [queue]* [Method on Connection]

If *queue* is given, remove the *queue* entries for *address* from the queuestaff table. Otherwise, remove *address* from the staff table and remove all corresponding queuestaff entries.

login *queue address password* [Method on Connection]

Authenticate staff *address* in *queue* with *password*. Returns true or false.

changepassword *address password* [Method on Connection]

Change the password entry for the staff *address* to *password*.

3.6 User operations: `mkuser`, etc.

Users are addresses which are not staff members, but which are involved with particular tickets. These do need some managing, particularly when a non-existent address gets inserted into the database.

Cascabel tries very hard to prevent staff addresses for a queue from getting listed as users on tickets in the queue. This is part of the “don’t send extra copies of mail” effort.

mkuser *queue id address* [Method on Connection]

Add *address* to the user table for ticket *id* in *queue*.

lsusers *queue id* [Method on Connection]

Return the user entries associated with ticket *id* in *queue*.

rmusers *ids* [*queue*] [Method on Connection]
Remove addresses *ids* from the users table, limited to entries for *queue* if it is present.

luseraddresses [*queue*] [Method on Connection]
Return a list of user addresses, either all unique addresses in the table or only those for *queue*, if it is present. *Deprecated*

3.7 Log operations: getlog, log

Log entries record information from mail messages and status changes about a ticket. Every event should be mentioned here.

For mail message entries, all of the headers are preserved, to handle MIME messages, for example.

getlog [*queue*] [*id*] [*predicate*] [Method on Connection]
Return the log entries for *queue* and *id* or matching *predicate*. The legal combinations are:

- *queue* and *id*
- *queue* and *predicate*
- Just *predicate*

For the first option, the synonyms table is checked.

log *queue id entry* [*time*] [Method on Connection]
Log *entry* for ticket *id* in *queue* at *time* (or the current time).

4 Mail Handling

One of the primary purposes of Cascabel is to manage communication about issues, and the primary tool for communication in this system is electronic mail. Cascabel fundamentally provides an archived mailing list, where the information regarding a particular ticket is collected in the logs.

There are two ways mail about existing tickets can be handled:

- By passing all mail about every ticket to every staff member, using the outgoing mail address. This is the basic approach, and the only one currently supported by Cascabel.
- By sending mail about a ticket only to the users associated with the ticket and to a particular staff member, who “owns” the ticket. This approach reduces the noise for most staff members, at the cost of potentially not keeping everyone up to date. Some hooks to handle this approach are present, but not at all tested.

The mail handling functions and classes are to be found in the ‘`CascabelMail.py`’ Python module.

4.1 Sending mail: `send` and `resend`.

Cascabel, unlike some other systems, makes no pretensions about being a mail transport agent. It uses the `sendmail` program for all mail messages sent from the Python module. (There is no similar PHP module—use the standard PHP `mail()` function.)

send *message addresses* [Function]
Send the formatted *message* to the list of *addresses*, using `sendmail`.

resend *message sender addresses* [Function]
Resend *message* to the list of *addresses* without altering it, except by adding the “Resent-From: *sender*” and “Resent-To:” headers.

4.2 Mail header manipulation

The basic mail handling module provides a number of functions for dealing with email headers. While these are primarily used in the ‘`CsacabelMail.py`’ module, most of the queue-specific mail handling scripts use a subclass of `MailHandler`, which may need to use these functions.

filterheader *field headers* [Function]
Return *headers* with any *field* headers removed.

replaceheader *field value headers* [Function]
Replace the instances of *field* headers with a “*field*: *value*” header in *headers*, returning the result.

filterfromline *headers* [Function]
Return headers with the “From ” line removed. The `from_` line is added by the mail transport agent when the message is being stored to contain the envelope information.

headervalues *field headers* [Function]
Return a list containing the values of the header *field* from *headers*.

headervalue *field headers* [Function]
Return the first value given by **headervalues**.

Subject *headers* [Function]
The value of the Subject header.

XCascabel [Function]
The values of any application-specific X-Cascabel headers.

Comment [Function]
The values of any Comment headers.

addressesfromheaders *field headers* [Function]
Return the list of addresses given in header *field*.

From [Function]
Addresses from the From header.

To [Function]
Addresses from the To header.

CC [Function]
Addresses from the CC header.

Sender [Function]
Addresses from the Sender header.

4.3 Message body

quotebody *body* [Function]
Quote each line of the *body* of an email message with “>” strings.

4.4 The MailHandler

The `MailHandler` class provides the basic functionality for dealing with incoming mail. The functions it provides are broken up into several methods, to allow subclasses for specific queues to take queue-defined actions under queue-defined circumstances.

MailHandler *queue* [Function]
Return a `MailHandler` instance for *queue*.

SilentMailHandler *queue* [Function]
Return an instance of a subclass of `MailHandler` which does not send or forward mail. This class is useful for processing an existing mail log and for testing.

close [Method on `MailHandler`]
Close the database connection used by the `MailHandler`.

incoming *message* [Method on `MailHandler`]
Process an incoming mail *message*:

1. Add a comment to the message headers indicating that Cascabel has received the message. If one of these is already present in the message, the message is ignored.
2. Drop the message if `filter` returns true.
3. Test if the message has a ticket id:
 - If so, find the associated ticket (if there is none, call `returnunknown`). If the sender is a staff member, check for `remoteactions`.
 - If not, `filter` for Reject addresses, make the new ticket, update the subject line with the new id, check for `remoteactions`, and send a `receipt`.
4. Log the message.
5. Add any unknown From, To, and CC addresses to the users for the ticket. (Known addresses are staff members, queue addresses, “Nobody” filter address, and known users.)
6. `forward` the message.

hasid *subject* [Method on `MailHandler`]
Check to see if the *subject* matches the subject/ticket id pattern in the queues table.

returnunknown *pseudo-id* [Method on `MailHandler`]
Return the message to the sender, indicating that no ticket *pseudo-id* exists.

receipt *id* [Method on `MailHandler`]
Send a message to the originator indicating that ticket *id* has been created. The format of the message is given by the receipt column of the queues table.

forward *id* [Method on MailHandler]

Pass the message on to the staff and users.

filter *headers* [*type*] [Method on MailHandler]

Check the mail *headers* against the filters in the database. *type* defaults to Ignore. If the message is filtered, it is resent to the queue's outgoing address, to avoid losing mail without warning.

remoteactions [Method on MailHandler]

Handle action-at-a-distance. The format of the header is "X-Cascabel: <cmd>". The two possible remote actions are:

set Assigns a value to a field in the ticket. The syntax is "field=values".

user Adds the users to the ticket. The syntax is "user[,user]...".

5 Utility Functions

Cascabel provides a number of utility functions that are needed by many queues and interfaces. These functions are found in the ‘`Cascabel.py`’ and ‘`Cascabel.php`’ modules.

age *date* [*now*] [Function]

Return a string describing the approximate difference between *age* and *now*, where *now* defaults to the current time. The string will be something like “4 mins”.

canonical *address* [Function]

Ensures *address* is a valid (more or less) email address, by checking for an ‘@’ character. If one is not found, the default domain (as given when configuring Cascabel) is appended to *address*, separated by ‘@’.

thisuser [Function]

Return the canonical address of the user running the program. Obviously, this is only valid in the Python implementation. It can be used to test if the user running the program is a staff member, or to use the user running the program as a default wherever a user is needed.

6 CGI support

Unrelated to Cascabel, but useful in many of the CGI interfaces written using PHP, `Machine.php` and `TemplateCache.php` are PHP modules providing a session-based state machine and a simple template mechanism.

6.1 `Machine.php`: Session-based state machine

The session-based state machine supports writing a CGI application as a collection of state machine transitions. In this approach, the states are represented by displayed web pages. The transitions are instances of class `Transition`. (This mechanism is related to the idea of using continuations for web programming, but is performed manually and is less powerfully.) The instances of `Transition` are created before displaying a page and saved in the session, with a transition identifier inserted into the page (via an instance of `TransitionKey`). When a user submits a form, a hidden field in the form contains a transition identifier. The transition is recovered from the session and its `execute` method is called with the input fields from the form.

The transitions in the session store are kept for the lifetime of the session, (hopefully) allowing the user to navigate the CGI application using any mixture of browser back buttons, form buttons and links, and multiple windows. The programmer using `Machine` will obviously need to take into consideration any transactional semantics needed in the application.

The following example should illustrate the basic approach:

```
class Search extends Transition {
    function Search ($id) { $this->id = $id; }
    function execute ($mc, $rq) {
        echo search_form($mc, $this->id, new Query);
        return 'search';
    }
}
```

In the example, a `Search` instance is created with an *id*, which records the user's identity associated with the session. This transition leads from a basic menu to a state showing a search form. When the transition is executed, the search form is printed, based on the state machine, the user identity, and a new, bare `Query` (which will be filled in from the form). The return value identifies the new state. (The state names other than "start" are for documentation purposes.)

To continue the example, the function `search_form` gets a template, replaces the message and action substitutions, inserts the query information, and inserts a transition key associated with a `Transition` that lists the tickets matching the query.

```
function search_form ($mc, $id, $q, $msg="") {
    $t = $mc->get_template('search-form');
    $t->replace('message', $msg);
    $t->replace('action', $_SERVER['PHP_SELF']);
    $t->replace_all($q->values());
    $k = $mc->set_transition(new ListTickets ($id, $q));
}
```

```

    $t->replace('k', $k->as_form_field());
    return $t->show();
}

```

6.1.1 Machine: The state machine engine

Machine [Function]

The state machine tracks the state and the set of transitions that are available. Creating a state machine object restores the information from PHP's session and executes the next transition, which may be either a default transition (the initial transition in a new session), or a transition identified from the CGI request.

initial_transitions [Method on Machine]

Create an initial transition table. The transition table is an array mapping transition identifiers to instances of subclasses of Transition; the initial table should map the "start" transition identifier to a default transition.

The following example is the body of a basic `initial_transitions` function.

```
return array("start" => new ToAuth);
```

In this example, `ToAuth` is a subclass of `Transition`, whose constructor does not take any arguments.

set_transition *transition* [Method on Machine]

Records the *transition* in the session, returning an instance of `TransitionKey` associated with the *transition*.

shutdown [Method on Machine]

Clear the transition table.

6.1.2 Transition: Persistent transitions

Transition [Function]

Creates a Transition. The constructor for a subclass should take as arguments any information that the transition should preserve until it is invoked.

execute *machine request* [Method on Transition]

The `execute` method is invoked by the state machine when the transition is taken. Its arguments are the state *machine* object (an instance of `Machine`) invoking the method, and the CGI *request* (PHP's `REQUEST` variable) from the user.

6.1.3 TransitionKey: Identifying transitions

When the `set_transition` method of `Machine` is invoked, the result is a `TransitionKey`, identifying the `Transition` that has been stored in the session. The `TransitionKey` can be used to create either a hidden form field (for use in a form) or a CGI GET query string (for use as a simple link).

as_form_field [Method on `TransitionKey`]

Returns a string like

```
<input type="hidden" name="k" value="id">
```

The *id* value can be used to restore the transition from the session store.

as_query [Method on `TransitionKey`]

Returns a string like

```
k=id
```

suitable for use (after a “?”) in a URL. The *id* can be used to restore the transition from the session.

6.2 TemplateCache.php: Reusable templates

Templates are used to separate the presentation of the CGI application from the code implementing it. The templates are kept in separate files which are read by the application as needed. The templates have positions in them for substitutions to be performed by the application.

In order for templates (for, say, table rows) to be reused many times in the same program execution, the templates are cached in the `TemplateCache` instance.

get_template *name* [Method on `TemplateCache`]

Returns a `Template` object containing the template identified by *name*. The file the template is taken from is `./templates/name.tpl`.

The `Templates` returned have two functions which give replacement texts, `replace` and `replace_all`. The templates should have substitution points given by `‘%name%’`; i.e. the name of the substitution point surrounded by percent signs.

replace *name text* [Method on `Template`]

Replaces the all occurrences of the substitution *name* with *text*.

replace_all *dict* [Method on `Template`]

The *dict* should be an array mapping *names* to *texts*; `replace_all` performs the same operation as multiple calls to `replace`.

show [Method on `Template`]

Return the contents of the template with all of the substitutions performed.

7 Scripts: `cascabel-admin`, etc.

Cascabel comes with a number of scripts. Each has a man page which you should examine for specific details.

7.1 `cascabel-admin`

The `cascabel-admin` script provides a command-line interface for managing the Cascabel queues. At the time of this writing, it is limited to:

- listing and removing user email addresses
- listing, removing, and adding staff accounts

See the '`cascabel-admin(1)`' man page for more information.

7.2 `cascabel-config`

The `cascabel-config` script is used to provide access to the configuration information given when installing Cascabel. This script is primarily used while configuring individual queue packages.

7.3 MySQL `makecascabel` script

The '`MySQL/makecascabel`' script sets up the Cascabel database and creates the Cascabel user in the DBMS.

7.4 MySQL `initcascabel` script

The '`MySQL/initcascabel`' script creates the basic set of Cascabel tables in the database.

7.5 Example scripts

The `cascabel-query`, `cascabel-show`, and `cascabel-mail` scripts are examples, used with the sample cascabel queue. The sample queue is set up in the database by the '`makecqueue`' MySQL script.

Other examples, including a near-direct translation of Req and the programs and what-not for the other queues from UTCS, should be available from the Cascabel home page.

8 Cascabelwrapper and the Password File

Cascabel keeps the SQL database contact information including the password in a file that is readable only by the user identified when Cascabel is configured. The format of the file is:

```
hostname:database:password
```

Being able to read this file involves using the `cascabelwrapper` program, which is installed setuid to the Cascabel user. This program uses the name it is invoked under to switch between the Python scripts—a symbolic link to `cascabelwrapper` named “foo” will try to execute the Python “foo” script in the Cascabel library’s bin directory.

See the `cascabelwrapper(1)` man page for more information.

Concept, Function, and Table Indices

Concept Index

A

addresses, staff	12
addresses, user	12
And (SQL)	9
AtLeast (SQL)	9
AtMost (SQL)	9

C

cannical email addresses	18
casabel-admin	22
casabel-config	22
casabel-mail	22
casabel-query	22
casabel-show	22
casabelwrapper	23
connection, database	10, 23

D

database connection	10, 23
database password	2
database password file	10, 23
default domain	6, 18
domain, default	18

E

e-mail filters	10
email addresses, canonical	18
email handling	16
email headers	14
email remote actions	17
email, sending	14
entries, log	13
Equal (SQL)	9

F

False (SQL)	9
filters, e-mail	10

G

general queue operations	10
--------------------------------	----

H

headers, email	14
----------------------	----

I

id, ticket	1, 11
Ignore	5, 10, 17
initcasabel	22
initial transition (state machine)	20
IsNull (SQL)	9

L

LessThan (SQL)	9
life-cycle, ticket	1
log entries	13
logging	1

M

Machine	19
makecasabel	22
makecqueue	22
merge tickets	11
modules, Python	1
MoreThan (SQL)	9
mxDateTime (Python module)	1
MySQL	1, 4
MySQLdb (Python module)	1

N

Nobody	5, 10, 16
Not (SQL)	9
NotEqual (SQL)	9

O

operations, ticket	11
operators, SQL	9
Or (SQL)	9

P

password file, database	10, 23
password, database	2
password, staff	12
PHP	1
Python	1

Q

queue 1
 queue operations, general 10
 queue-specific tables 7

R

Reject 5, 10, 16
 remote actions, email 17
 resolved ticket 1
 Rlike (SQL) 9

S

search, for ticket 11
 searching log entries 13
 sending email 14
 session, PHP 19
 setuid scripts 2
 SQL database password 2, 23
 SQL operators 9
 staff addresses 12
 staff members 1
 staff passwords 12
 state machine 19
 state machine transitions 20
 status information 1

Function Index

A

addressesfromheaders 15
 age 18
 And 9
 as_form_fieldon TransitionKey 21
 as_queryon TransitionKey 21
 AtLeast 10
 AtMost 10

C

canonical 18
 CC 15
 changepasswordon Connection 12
 closeon MailHandler 16
 Comment 15
 Connection 10

E

Equal 10
 executeon Transition 20

T

tables, queue-specific 7
 template 21
 TemplateCache 21
 ticket 1
 ticket id 1, 11
 ticket information 11
 ticket life-cycle 1
 ticket operations 11
 ticket search 11
 ticket tables 7
 ticket, resolved 1
 transition, initial (state machine) 20
 transition, state machine 20
 True (SQL) 9

U

update tickets 11
 update time 11
 user addresses 12

X

X-Cascabel header 17

F

False 9
 filteron MailHandler 17
 filterfromline 15
 filterheader 14
 forwardon MailHandler 17
 From 15

G

get_templateon TemplateCache 21
 getlogon Connection 13
 getqueueon Connection 10
 getqueueson Connection 10
 getticketon Connection 11

H

hasidon MailHandler 16
 headervalue 15
 headervalues 15

I

incomingon MailHandler..... 16
 initial_transitionon Machine 20
 IsNull..... 9

L

LessThan..... 10
 logon Connection 13
 loginon Connection..... 12
 lsfilterson Connection..... 11
 lsstaffon Connection..... 12
 lsstatuson Connection..... 10
 lsticketson Connection..... 11
 lsuseraddresseson Connection..... 13
 lsuserson Connection..... 12

M

Machine..... 20
 MailHandler 16
 mkstaffon Connection..... 12
 mkticketon Connection..... 11
 mkuserson Connection..... 12
 MoreThan..... 10
 mvticketon Connection..... 11

N

Not..... 9
 NotEqual..... 10

O

Or..... 9

Q

quotebody 15

R

receipton MailHandler..... 16
 remoteactionson MailHandler 17
 replaceon Template..... 21
 replace_allon Template..... 21
 replaceheader..... 14
 resend..... 14
 returnunknownon MailHandler 16
 Rlike..... 10
 rmstaffon Connection..... 12
 rmuserson Connection..... 13

S

send..... 14
 Sender..... 15
 set_transitionon Machine..... 20
 setticketon Connection..... 11
 shownon Template..... 21
 shutdownon Machine..... 20
 SilentMailHandler..... 16
 Subject..... 15

T

thisuser..... 18
 To..... 15
 Transition..... 20
 True..... 9

U

updateon Connection..... 11

X

XCascabel..... 15

Table Index

A

allowedstatus 5, 10, 11

L

log..... 5, 13

M

mailfilters..... 5, 10, 17

Q

queues..... 4, 10, 16

queestaff 6, 12

S

staff 6, 12

synonyms..... 6, 11, 13

T

ticket 7, 11

U

users..... 7, 11, 12, 13