PreciseMail Anti-Spam Gateway Programming Guide

July 2010

This manual provides documentation for public programming interfaces provided as part of PreciseMail Anti-Spam Gateway.

Software Version: PreciseMail Anti-Spam Gateway V3.2

Process Software

20 July 2010

Process Software, LLC
959 Concord Street
Framingham, MA 01701-4682 USA
Voice: +1 508 879 6994; FAX: +1 508 879 0042
info@process.com

Contents

INDEX

Preface

This manual provides documentation for public programming interfaces provided as part of PreciseMail Anti-Spam Gateway.

1 User Database API

The PreciseMail user database stores user-specific options for message filtering and the web user interface. The contents of the user database are roughly the same as the options displayed in the Preferences section of the web user interface. Every user who modifies their preferences from the system defaults has an entry automatically created for them in the user database.

Prior to version 2.4 of PreciseMail Anti-Spam Gateway, the user database could only be accessed through the web interface and the pmasadmin tool. This document describes an application programming interface (API) that allows PreciseMail sites to develop their own custom software that reads and modifies entries in the user database.

Machine-readable indexed files are used to store the user database. Each entry in the database consists of a unique email address and the values of several user options. This API allows those options to be read and/or modified on a user-by-user basis. Note that only users who have changed their settings from the system defaults will have entries in the user database.

This document assumes that you already have a basic understanding of the options that can be set on a user-by-user basis in PreciseMail. Knowledge of how the PreciseMail filtering engine scores messages and treats messages identified as spam or possible spam is also assumed. See the PreciseMail Manager's Guide for more information about these topics.

1.1 Using The API

The PreciseMail user database API consists of two files: a header include file (*userdb_api.h*), and a shareable image (*PMAS_USERDB_API.EXE*) for VMS, or a shareable object (*libpmas_userdb.so*) for UNIX. Include *userdb_api.h* in any source file that makes use of the API. Below are the first few lines from a sample C program showing the included user database header file:

```
#include <stdlib.h>
#include <stdio.h>
#include "userdb_api.h"
```

The exact syntax required to link against the *libpmas_userdb* shareable will depend upon your compiler and linker, but in general the following syntax should work:

Linux

```
$ export LD_RUN_PATH=/pmas/bin
$ gcc -fPIC -c -o program.o program.c
$ gcc -L/pmas/bin/ -o program program.o -lpmas_userdb
```

Solaris

```
$ gcc -fPIC -c -o program.o program.c
$ gcc -L/pmas/bin/ -o program program.o -lpmas_userdb
```

Tru64

```
$ cc -Wl,-rpath,/pmas/bin/ -L/pmas/bin/ -o program program.c -lpmas_userdb
```

OpenVMS

```
$ LINK program,PMAS_COM:pmas_userdb_api.opt/OPT
```

Programs that access the PreciseMail user database need to be run by a user who has sufficient privileges to access the database files. On VMS, the SYSTEM user should always have sufficient privileges. On UNIX, the root user should always have sufficient privileges. In addition, UNIX sites using PreciseMail integrated with PMDF can use the pmdf user; PreciseMail integrated with Sun Messaging Server can use the mailsrv user; and PreciseMail integrated with Sendmail can use the daemon user.

1.2 Example Programs

Six example programs that use the user database API are included in the PreciseMail distribution. Fully commented source code and build files are available in the */pmas/api/userdb/* directory on UNIX and the *PMAS_ROOT:[API.USERDB]* directory on OpenVMS.

To build the example programs on UNIX, run the *make* command specifying the example makefile:

```
$ make -f userdb_api_makefile
gcc -c -o userdb_api_example1.o userdb_api_example1.c
gcc -o userdb_api_example1 -R/pmas/bin -L/pmas/bin userdb_api_example1.o -lpmas_use

[...]

gcc -c -o userdb_api_example6.o userdb_api_example6.c
gcc -o userdb_api_example6 -R/pmas/bin -L/pmas/bin userdb_api_example6.o -lpmas_use
$
```

To build the example programs on OpenVMS, run MMS or MMK:

```
$ MMS/DESCRIP=USERDB_API_DESCRIP.MMS
```

The six example programs are:

userdb_api_example1

Checks that a user specified on the command line exists in the user database, and then opts the user into message filtering. Sample run:

```
$ ./example1 user@example.com
Successfully opted-in user@example.com
$
```

userdb_api_example2

Reads a list of users from a file specified on the command line, and deletes them the user database. Schools might want to run a similar program at the end of a semester to remove graduating students. Sample run:

```
$ ./example2 graduates.txt\BOLD
user1@example.com deleted
user2@example.com deleted
bogus@example.com does not exist in user database
user3@example.com deleted
$
```

userdb_api_example3

Print the specified user's threshold settings if the password supplied on the command line is correct. Sample run:

```
$ ./example3 user@example.com secret
Thresholds for user@example.com
Tagging: system
Quarantine: enabled, 5.000
Discard: disabled, 20.000
$
```

userdb_api_example4

Prints a list of every user who has enabled discarding and their discard threshold. Sample run:

```
$ ./example4
user1@example.com, 50.000
user2@example.com, system
user3@example.com, system
user4@example.com, 20.500
$
```

userdb_api_example5

Renames every user whose address belongs to the first domain specified on the command line to the second domain specified on the command line. Sites that are changing their domain name might want to run a program like this. (Note that this only renames the user database entry - other programs will need to handle user rule files and quarantined messages.) Sample run:

```
$ ./example5 example.org example.com
Renamed user1@example.org to user1@example.com
Renamed user2@example.org to user2@example.com
Renamed user3@example.org to user3@example.com
Renamed user4@example.org to user4@example.com
$
```

userdb_api_example6

Gets a list of every user in the user database, and checks each one whose address ends in .de to see if Subject line tagging is enabled. If tagging is enabled, the tag text is changed to ABFALL (German for "trash".) Sample run:

```
$ ./example6
Changed tag for hans@example.de
Changed tag for jacob@example.de
Changed tag for gunther@example.de
$
```

Note: Like any program that accesses the user database, these examples must be run by a user with sufficient privileges to access the user database files.

1.3 User Database API Functions

This section contains a complete list of the user database API functions, arranged in alphabetical order. Entries in the user database are indexed by the user's email address, so the terms "user" and "email" are used interchangeably. All strings are standard NULL-terminated ASCII strings.

userCheckDBPassword

C

status = **userCheckDBPassword**
(email, password)

argument information

```
int userCheckDBPassword(char *email, char *password)
```

ARGUMENTS

email

The user whose password is being checked.

password

The supplied password that will be checked against the user's stored password.

DESCRIPTION

Checks *password* to see if it matches the user's password in the PreciseMail password database. Note that this does not authenticate users against other authentication sources, such as an LDAP directory server. The use of this function is demonstrated in *userdb_api_example3*.

RETURN VALUES

-1	An error occurred and the password couldn't be checked
0	Supplied password is incorrect
1	Supplied password is correct

userCreateNew

C

status = **userCreateNew**
(email, password)

argument information

```
int userCreateNew(char *email, char *password);
```

ARGUMENTS

email
The email address of the user being created.

password
If not NULL, the user's password inside the PreciseMail user database.

DESCRIPTION

Creates a new user in the PreciseMail user database. If *password* is a non-NULL value, it will be stored as the user's password in the database. (This password will be checked when the user authenticates if the PMAS authentication method is specified in the *auth_methods* configuration variable. See Chapter 2 of the PreciseMail Administrator's Guide for more information.)

RETURN VALUES

0	Failure
1	Success

userDelete

C

status = **userDelete**
(email)

argument information

```
int userDelete(char *email);
```

ARGUMENTS

email
The user to remove from the user database.

DESCRIPTION

Deletes the specified user and all of their settings from the user database. Database entries cannot be recovered if accidentally deleted, so use this function carefully. The use of this function is demonstrated in *userdb_api_example2*.

RETURN VALUES

0	Failure
1	Success

userExists

C

status = **userExists**
(email)

argument information

```
int userExists(char *email);
```

ARGUMENTS

email
The user whose existence is being checked.

DESCRIPTION

Checks for the existence of a user named *email* in the PreciseMail password database. The use of this function is demonstrated in *userdb_api_example1* and *userdb_api_example2*.

RETURN VALUES

0	User does not exist in database
1	User exists in database

userGetDiscardOptions

C

status = **userGetDiscardOptions**
(email, enabled, threshold, system)

argument information

```
int userGetDiscardOptions(char *email, int *enabled, double *threshold,
                          int *system);
```

ARGUMENTS

email
The user to retrieve discard settings for.

enabled
Set to 1 if discarding is enabled for the user.

threshold
The message score threshold above which messages for this user are discarded.

system
Set to 1 if the user is using the system defaults for discarding.

DESCRIPTION

Retrieves a user's discard options from the user database. Note that PreciseMail ignores the values of *threshold* and *enabled* if *system* is set to 1. The use of this function is demonstrated in *userdb_api_example3* and *userdb_api_example4*.

RETURN VALUES

0	Failure
1	Success

userGetGUINoPopups

C

status = **userGetGUINoPopups**
(email, on)

argument information

```
int userGetGUINoPopups(char *email, int *on);
```

ARGUMENTS

email
The user to retrieve GUI settings for.

on
Set to 1 if web interface popups are enabled for the user.

DESCRIPTION

Retrieves a user's preferences about the use of popups in the web user interface. If *on* is 1, popup windows will be used in place of interstitial pages for some operations, such as releasing a message from quarantine.

RETURN VALUES

0	Failure
1	Success

userGetOptIn

C

status = **userGetOptIn**
(email, opt_in)

argument information

```
int userGetOptIn(char *email, int *opt_in);
```

ARGUMENTS

email
User to get opt-in status for.

opt_in
Set to 1 if user is opted-in.

DESCRIPTION

Determines if the specified user is opted-in to message filtering.

RETURN VALUES

0	Failure
1	Success

userGetQuarantineOptions

C

status = **userGetQuarantineOptions**
(email, enabled, threshold, system)

argument information

```
int userGetQuarantineOptions(char *email, int *enabled, double *threshold,
                             int *system);
```

ARGUMENTS

email
The user to get quarantine settings for.

enabled
Set to 1 if quarantining is enabled for the user's account.

threshold
The message score threshold above which messages for this user are quarantined.

system
Set to 1 if the user is using the system defaults for quarantining messages.

DESCRIPTION

Retrieves a user's quarantine options from the user database. Note that PreciseMail ignores the values of *threshold* and *enabled* if *system* is set to 1. The use of this function is demonstrated in *userdb_api_example3*.

RETURN VALUES

0	Failure
1	Success

userGetQuarantineSortOrder

C

status = **userGetQuarantineSortOrder**
(email, sort)

argument information

```
int userGetQuarantineSortOrder(char *email, int *sort);
```

ARGUMENTS

email

The user to get the default quarantine sort order for.

sort

The method used to sort quarantined messages in the user's quarantine listing. The possible values are:

Value	Meaning
0	Normal (ascending by time received)
1	Ascending by score
2	Descending by score
3	Ascending by Subject line
4	Descending by Subject line
5	Descending by time received

DESCRIPTION

Gets the default sort order for messages on the user's quarantine listing page in the web user interface. (Once the quarantine listing is loaded, users can sort the page any way they want by clicking on a column header.)

RETURN VALUES

0	Failure
1	Success

userGetQuarDisplayAll

C

status = **userGetQuarDisplayAll**
(email, on)

argument information

```
int userGetQuarDisplayAll(char *email, int *on);
```

ARGUMENTS

email
The user to retrieve quarantine display settings for.

on
Set to 1 if all of the user's quarantined messages are displayed by default.

DESCRIPTION

Retrieves the specified user's preferences for the amount of messages displayed by default in the web user interface. If *on* is 0, only messages quarantined for the user during the current calendar day are displayed. If *on* is 1, every message quarantined for the user is displayed regardless of when it was quarantined. Note that generating a listing of all quarantined messages for a user can require a substantial amount of system resources if there are a very large number of messages quarantined for the user.

RETURN VALUES

0	Failure
1	Success

userGetQuarNoticeEnabled

C

status = **userGetQuarNoticeEnabled**
(email, enabled)

argument information

```
int userGetQuarNoticeEnabled(char *email, int *enabled);
```

ARGUMENTS

email
The user to retrieve quarantine display settings for.

enabled
Set to 1 if quarantine notification emails are sent to the user.

DESCRIPTION

Retrieves the specified user's preferences for receiving quarantine notification emails when new mail has been quarantined for them since the last notification was sent. By default, the notification messages are sent twice a day (the frequency and time of notifications are configurable by the system administrator). If enabled is set to 0, the user will not receive quarantine notification emails.

RETURN VALUES

0	Failure
1	Success

userGetTagAppend

C

status = **userGetTagAppend**
(email, append)

argument information

```
int userGetTagAppend(char *email, int *append);
```

ARGUMENTS

email
The user to retrieve Subject line tag settings for.

append()

If set to 1, spam messages are tagged by having a text token appended to the end of the Subject line. If set to 0 (the default), the tag is prepended to the front of the Subject line.

DESCRIPTION

Retrieves the specified user's preferences for where a text token is placed in the Subject line of messages tagged as spam. The current text token can be obtained by calling *userGetTagText*, and it can be set with *userSetTagText*.

RETURN VALUES

0	Failure
1	Success

userGetTagOptions

C

status = **userGetTagOptions**
(email, enabled, threshold, system)

argument information

```
int userGetTagOptions(char *email, int *enabled, double *threshold,
                      int *system);
```

ARGUMENTS

email
The user to retrieve tagging settings for.

enabled
Set to 1 if tagging is enabled for the user.

threshold
The message score threshold above which messages for this user are tagged.

system
Set to 1 if the user is using the system defaults for tagging messages.

DESCRIPTION

Retrieves a user's Subject line tagging options from the user database. Note that PreciseMail ignores the values of *threshold* and *enabled* if *system* is set to 1. The use of this function is demonstrated in *userdb_api_example3* and *userdb_api_example6*.

RETURN VALUES

0	Failure
1	Success

userGetTagText

C

status = **userGetTagText**
(email, text)

argument information

```
int userGetTagText(char *email, char *text);
```

ARGUMENTS

email
The user to retrieve message tagging settings for.

text
The text placed in the Subject line of messages that cross the tagging threshold.

DESCRIPTION

Retrieves the text placed in the Subject line of tagged messages for the specified user. By default, the text is [SPAM].

RETURN VALUES

0	Failure
1	Success

userList

C

users = **userList**
(num_users)

argument information

```
char **userList(int *num_users);
```

ARGUMENTS

num_users
Will be set to the number of users in the user database when the function returns.

DESCRIPTION

Generates an array of strings, each corresponding to a user in the user database. This function is useful if you want to perform an action on every user or a subset of users in the database. The use of this function is demonstrated in *userdb_api_example4*, *userdb_api_example5*, and *userdb_api_example6*.

RETURN VALUES

NULL	Failure
	An array of character pointers, each of which points to the name of a user who has a record in the user database.

userOptIn

C

status = **userOptIn**
(email)

argument information

```
int userOptIn(char *email);
```

ARGUMENTS

email
The user to be opted-in to message filtering.

DESCRIPTION

Opts the specified user into message filtering. All of the user's incoming mail will be scanned by PreciseMail. If the user is already opted-in, this function will not produce an error. The use of this function is demonstrated in *userdb_api_example1*.

RETURN VALUES

0	Failure
1	Success

userOptOut

C

status = **userOptOut**
(email)

argument information

```
int userOptOut(char *email);
```

ARGUMENTS

email
The user to be opted-out of message filtering.

DESCRIPTION

Opts the specified user out of message filtering. None of the user's incoming mail will be filtered. If the user is already opted-out of filtering, this function will not produce an error.

RETURN VALUES

0	Failure
1	Success

userRename

C

status = **userRename**
(old_email, new_email)

argument information

```
int userRename(char *old_email, char *new_email);
```

ARGUMENTS

old_email

The user's current email address as stored in the user database.

new_email

The new email address that the user's information should be associated with in the user database.

DESCRIPTION

Renames a user, preserving all of their personal preferences. The old user record is not removed until the new user record is successfully created, preventing any data loss in the event of a software or hardware failure. The use of this function is demonstrated in *userdb_api_example5*.

RETURN VALUES

0	Failure
1	Success

userSetDiscardOptions

C

status = **userSetDiscardOptions**
(email, enabled, threshold, system)

argument information

```
int userSetDiscardOptions(char *email, int enabled, double threshold,
                          int system);
```

ARGUMENTS

email
The user to enable or disable discarding for.

enabled
Set to 1 if you want to enable discarding for the user, 0 if you want to disable discarding.

threshold
The score threshold above which the user's messages will be discarded.

system
Set to 1 if you want the user to use the system default discard settings, 0 if not.

DESCRIPTION

Sets a user's discard options in the user database. Note that PreciseMail ignores the values of *threshold* and *enabled* if *system* is set to 1. The values of *threshold* and *enabled* will still be updated in the database, but it will have no effect on filtering operations.

RETURN VALUES

0	Failure
1	Success

userSetGUINoPopups

C

status = **userSetGUINoPopups**
(email, on)

argument information

```
int userSetGUINoPopups(char *email, int on);
```

ARGUMENTS

email
The user to retrieve GUI settings for.

on
Set to 1 to enable web interface popups for the user, 0 to disable them.

DESCRIPTION

Sets a user's preferences about the use of popups in the web user interface. If *on* is set to 1, popup windows will be used in place of interstitial pages for some operations, such as releasing a message from quarantine.

RETURN VALUES

0	Failure
1	Success

userSetPassword

C

status = **userSetPassword**
(email, password)

argument information

```
int userSetPassword(char *email, char *password);
```

ARGUMENTS

email
The user whose password is to be changed.

password
The user's new password, in plain text.

DESCRIPTION

Sets the user's password in the user database, replacing any existing password in the database.

RETURN VALUES

0	Failure
1	Success

userSetQuarantineOptions

C

status = **userSetQuarantineOptions**
(email, enabled, threshold, system)

argument information

```
int userSetQuarantineOptions(char *email, int enabled, double threshold,
                             int system);
```

ARGUMENTS

email
The user to enable or disable quarantining for.

enabled
Set to 1 if you want to enable quarantining for the user, 0 if you want to disable quarantining.

threshold
The message score threshold above which messages for this user are quarantined.

system
Set to 1 if you want the user to use the system default quarantine settings, 0 if not.

DESCRIPTION

Sets a user's quarantine options in the user database. Note that PreciseMail ignores the values of *threshold* and *enabled* if *system* is set to 1. The values of *threshold* and *enabled* will still be updated in the database, but it will have no effect on filtering operations.

RETURN VALUES

0	Failure
1	Success

userSetQuarantineSortOrder

C

status = **userSetQuarantineSortOrder** *(email, sort)*

argument information

```
int userSetQuarantineSortOrder(char *email, int sort);
```

ARGUMENTS

email

The user to set default quarantine listing sort order for.

sort

A value between 0 and 5 inclusive that specifies the way quarantined messages are sorted in the user's quarantine listing. The possible values are:

Value	Meaning
0	Normal (ascending by time received)
1	Ascending by score
2	Descending by score
3	Ascending by Subject line
4	Descending by Subject line
5	Descending by time received

DESCRIPTION

Changes the way messages are sorted by default on the user's quarantine listing page in the web user interface.

RETURN VALUES

0	Failure
1	Success

userSetQuarDisplayAll

C

status = **userSetQuarDisplayAll**
(email, on)

argument information

```
int userSetQuarDisplayAll(char *email, int on);
```

ARGUMENTS

email
The user to set quarantine display settings for.

on
Set to 1 to display all of the user's quarantined messages by default, 0 to show only today's quarantined messages.

DESCRIPTION

Sets the specified user's preferences for the amount of messages displayed by default in the web user interface. If you set on to 0, only messages quarantined for the user during the current calendar day are displayed. If on is 1, every message quarantined for the user is displayed regardless of when it was quarantined.

RETURN VALUES

0	Failure
1	Success

userSetQuarNoticeEnabled

C

status = **userSetQuarNoticeEnabled**
(email, enabled)

argument information

```
int userSetQuarNoticeEnabled(char *email, int enabled);
```

ARGUMENTS

email
The user to set quarantine options for.

enabled
Set to 1 to have quarantine notification emails sent to the specified user, 0 to turn off the notifications.

DESCRIPTION

Sets the specified user's preferences for receiving quarantine notification emails when new mail has been quarantined for them since the last notification was sent. By default, the notification messages are sent twice a day (the frequency and time of notifications are configurable by the system administrator).

RETURN VALUES

0	Failure
1	Success

userSetTagAppend

C

status = **userSetTagAppend**
(email, append)

argument information

```
int userSetTagAppend(char *email, int append);
```

ARGUMENTS

email
The user to set Subject line tag options for.

append
Set to 1 to append the tag to the end of a spam message's Subject line. Set to 0 to place the tag at the beginning of the Subject line.

DESCRIPTION

Sets the specified user's preferences for where a text token is placed in the Subject line of messages tagged as spam.

RETURN VALUES

0	Failure
1	Success

userSetTagOptions

C

status = **userSetTagOptions**
(email, enabled, threshold, system)

argument information

```
int userSetTagOptions(char *email, int enabled, double threshold,
                      int system);
```

ARGUMENTS

email
The user's email address.

enabled
Set to 1 if you want to enable tagging for the user, 0 if you want to disable tagging.

threshold
Message score threshold above which the user's messages will be tagged

system
Set to 1 if you want the user to use the system default tag settings, 0 if not.

DESCRIPTION

Sets a user's Subject line tagging options in the user database. Note that PreciseMail ignores the values of *threshold* and *enabled* if *system* is set to 1. The values of *threshold* and *enabled* will still be updated in the database, but it will have no effect on filtering operations. The use of this function is demonstrated in *userdb_api_example6*.

RETURN VALUES

0	Failure
1	Success

userSetTagText

C

status = **userSetTagText**
(email, text)

argument information

```
int userSetTagText(char *email, char *text);
```

ARGUMENTS

email
The user to set message tagging settings for.

text
The text placed in the Subject line of messages that cross the tagging threshold.

DESCRIPTION

Sets the text token placed in the Subject line of tagged messages for the specified user. By default, the text is [SPAM].

RETURN VALUES

0	Failure
1	Success

1.4 Concurrency Issues

The user database provides automatic granular locking, so multiple writes will not collide. The data in this particular database is "write rarely, read often", so it's unlikely that there will be a data concurrency issue. Still, it's important to keep in mind that data in the database can change between operations performed by your program if users make changes via the web user interface or other programs. Try to avoid writing your program in such a way that it depends on data values being constant between operations.

For example, let's say you've written a program that obtains a list of every user who has discarding enabled, performs some other processing for 15 minutes, and then opts those users out of filtering. (I don't know why you'd want such a program, but it's a simple example.) In the 15-minute interval between when your program obtained the list of users and when they were opted them out, several users could have enabled discarding via the web interface. Those users would not be opted out, since they didn't have discarding enabled when your program obtained the list of users.

To avoid such situations, try to keep your access to the user database as atomic as possible. In the above example, the program should be rewritten so the 15 minutes of processing occurs before or after the user database operations. If it isn't possible to perform user database operations that depend on previously obtained database information in a back-to-back fashion, try to run your program during periods of low system use.

Index