OpenEMM backend description

History

OpenEMM version	Version Date	Comment
1.0	23.09.2020	Initial implementation
1.1	26.01.2022	Update for LTS 22.04
1.1.1	10.02.2022	Document generalized
1.1.2	16.02.2022	Corrections incorporated
1.3	22.01.2025	Update to new features from version 25.04

Introduction

This document describes the backend components involved in sending a mail and the log data and tools available to track individual steps in the sending process.

If reference is made to database content, this refers to the "openemm" schema in which the application also operates.

Definitions of terms

This document uses terms that are commonly used in the EMM environment. To ensure that the meaning of the terms is known, a compact compilation of some terms is provided here:

- Admin mailing/test mailing: this refers to a mailing that is only to be sent to administrator or test recipients of the mailing list of a mailing. These are typically used during the creation phase of the mailing.
- Worldmailing: This refers to the final dispatch of a non-repeating mailing. This is the mailing that is sent once to all designated recipients.
- Date-based mailing: This refers to a mailing that is sent once a day to a selected group of recipients, defined by the mailing's target groups. To avoid such a mailing being sent repeatedly to all recipients, it is essential to assign a target group.
- Action-based mailing: These are individual mailings that are typically activated by a trigger (e.g. link click of a double opt-in mailing). These are also referred to as trigger mailings.
- XML package: This refers to the files in XML format that are generated for sending and contain all the information required to create the final mails.
- Frontend: This describes all systems on which a frontend process runs. This includes the components that are responsible for operating EMM (GUI), as well as the servers with which external services communicate (SOAP or RESTful web services) and servers that the recipients access (RDIR)

Conventions

If reference is made to a specific process within the document (e.g. user openemm, process generate3.py), this refers to the process with the specified name that is running under the designated operating system user. For Python processes, reference is made to the version running under Python3. If the process writes a log file with a different name, this is explicitly noted, e.g. user openemm, process java, log file mailout.

Structure of log files

In the backend area, the log files of EMM's own processes are generally structured according to a fixed scheme. They are located below the home directory of the respective service in the *var/log* directory. There are two types of log files here, outputs from processes and archived data.

The file name of the log files from processes follows this scheme:

- YYYYMMDD-<servername>-<processname>.log

The file name for archived data follows this scheme

- YYYYMMDD-<datenname>.log

where YYYYMMDD is the eight-digit representation of year-month-day, the <servername> is the name of the server on which the process is running without the domain, <processname> is the name of the process that wrote this file (if it is a Python process, the name is used without the ".py" suffix) and <datenname> is the type of data for archived data. This data is typically compressed after seven days, i.e. older files still have the ".gz" suffix and are deleted after 30 days.

In the case of archived data, the content of the file depends on the data itself and does not follow a fixed scheme, whereas the contents of the log files of processes follow this fixed scheme:

- [DD.MM.YYYY hh:mm:ss] <pid> <loglevel>/<area> <freetext>
- DD.MM.YYYY represents the current date.
- hh:mm:ss represents the current time.
- <pid> is the Linux process ID of the process that wrote this data.
- <loglevel> is the quality of the entry and ranges from DEBUG, NOTICE, INFO, WANING, ERROR, EXCEPTION, CRITICAL to FATAL.
- <area> is the (optional) area within the process that generated the output. If this specification is missing, the separating slash is also not present. The area can also contain further slashes, typically if the log outputs are nested in the program.
- <freitext> this is the actual message of the entry without a fixed scheme.

If a line does not follow this scheme, it is to be understood as a continuation of the previous line. This tends to be the exception and occurs, for example, when the output of an external program has been logged or in a stack trace when an exception is output. As a rule, however, multi-line outputs are also written according to the scheme defined here.

Structure of the file name of the XML packages

These files have a fixed structure to make it easier to assign them to the respective client and mailing. The general structure looks like this:

AgnMail-<LicenseID>=D<Timestamp>=<CompanyID>=<MailID>=<BlockID>=liaMngA.xml (for admin and test mailings)
or
AgnMail-<LicenseID>=D<Timestamp>=<CompanyID>=<MailID>=<BlockID>=liaMngA.xml.gz (for all other mailings)

where the individual elements are structured as follows:

- <LicenseID>: This is the static value 0.
- <Timestamp>: YYYYMMDDhhmmss (YYYY is the year, MM the month, DD the day, hh the hour, mm the minute and ss the second; the year here is four digits, all other values two digits, with a leading zero if required).
- <CompanyID>: This is the static value 1.
- <MailID>: This can be a combination of digits and capital letters, where (if you interpret the capital letters as separators of the digits) the *last* numeric value represents the MailingID of the mailing. Typically, the current structure of the MailID consists of a consecutive number, the value of the *maildrop_status_tbl.status_field* "for which the mailing was created, followed by the MailingID.
- <BlockID>: a unique ID for this block, this depends on the mailing type:
- action-based mailing: <consecutive number>CE<CustomerID>
- internal mailing to generate the provider preview: <consecutive number>CV<CustomerID>
- Other mailings: <consecutive number, at least four digits>

Overview of the mailing

This section is intended to provide a basic overview of the individual steps involved in sending a mailing without going into detail here. The creation of a mailing within EMM is not described here.

Settings for dispatch

For each mailing, there is an entry in the *maildrop_status_tbl* table within the database that provides the initial information for creating the mailings. Without such an entry, no mailing can be sent.

For admin, test and world mailings, a separate entry is created for each mailing. If the mailing is to be sent immediately or promptly (within the next hour), the mailing is initiated directly after the entry is created. If the mailing is to be sent at a later time, production is started by a separate process (user openemm, process generate3.py).

The dispatch is not initiated for date-based, on-demand and action-based mailings; these mailing types are activated and are active from then on. This means that they use their entry in the *maildrop_status_tbl* multiple times. The actual dispatch is started depending on the mailing type.

Date-based mailing

This checks every hour (user openemm, process generate3.py) whether a date-based mailing should be sent at the current hour. These mailings are then sent for this day.

Note: If, for example, a mailing is activated for dispatch at 10:00 and this is changed to 16:00 at 12:00, it will *not be* sent again on the same day as the mailing was already sent at 10:00.

On-demand mailing

To start this mailing, there must be an external process that initiates the start. This can be a logic within EMM (e.g. interval mailings) or provided by an external process (e.g. to generate a mailing directly after new data has been imported).

Action-based mailing

These mailings are linked to a trigger within EMM. If an event occurs that activates this trigger, the mailing is sent to the designated recipient.

Dispatch process

During dispatch, intermediate files are generated in XML format (XML packages), which contain all information required to generate the final mailings. Once a mailing has been sent, the handling of these intermediate files is described here.

In general, these XML packages are created directly by the "merger" service in the *var/spool/META* directory (user openemm, process java, log files merger and mailout).

The XML packages of admin and test mailings are converted into the final mails directly after creation (user openemm, process xmlback) and sent immediately.

All other XML packages are processed into the final mails by a separate process (user openemm, process pickdist3.py and xmlback) and transferred to the running MTA (mail transfer agent) as soon as their dispatch time is reached.

During dispatch, the MTA writes various information to its log file (/var/log/maillog), which contains details about the dispatch and the delivery status. This data is read (user openemm, process slrtscn3.py) and summarized and stored in the *log* directory for further processing.

Another process (user openemm, process update3.py) processes these files and writes them to the corresponding tables in the database so that the data is then available for analysis within EMM.

Delayed bounces

This is bounce information that is not generated directly during dispatch but is reported by email. These are received by the "mailloop" service (user openemm, process bavd3.py) and, analogous to the "mailer" service, prepared and collected and processed by the "merger" service.

Continuation of the newsletter dispatch

If the generation of the XML packages was aborted (user openemm, process java) and this is to be restarted, the first step is to determine why the generation was aborted.

Runtime error

In addition to the regular log file, the console output of the running process is also redirected to the *log//backend.log* file. This is necessary because a Java runtime error (e.g. OutOfMemory) cannot be written to the regular log file.

So, if the process (user openemm, process java) terminates unexpectedly, this log file should first be checked to see if there is a reference to the termination. If the error only indicates a temporary problem, you can proceed as described here, otherwise please contact AGNITAS Support.

If the generation was aborted by an abrupt shutdown of the server, you can proceed in the same way.

To resume generation, the "merger" service can simply be stopped and then restarted. The log file of the corresponding process (user openemm, process recovery3.py) can be used to track which mailings are then continued.

Programmatic termination

If generation is terminated by a programmatic abort, the cause must usually be eliminated before further action can be taken. If the error lies within the mailing, it is usually best to copy the mailing in question and then delete the original in order to correct the error in the copy.

There are roughly two reasons for this, either a program error (AGNITAS would then have to be informed) or an error that is so critical that it makes it impossible to continue the creation process (e.g. missing target group, referencing of non-existent profile fields).

A programmatic termination is typically preceded by an exception, which can be found in the log file (user openemm, process java, log file mailout).

Especially when several mailings are generated in parallel, it is not so easy to trace which log file entries belong to which mailing. Here, as soon as the information is available, there is a block of numbers in brackets in the <area> section that make it possible to assign the log file entries to a mailing, e.g:

```
[24.01.2022 14:17:00] DEBUG/ID:6/init/(610/33045/812684/1752184): Initial data valid
```

The number block 610/33045/812684/1752184 represents the client ID, the mailing list ID, the mailing ID and the mail drop status ID separated by a slash.

There is also a special entry (here ID:6) after the log level (here DEBUG), which allows you to find related log data. If you have found a log file entry for which you would like to view the entire history of mail generation, the easiest way to do this is via this ID, e.g. with:

```
grep /ID:6/ ~/var/log/20220124-`uname -n|cut -d. -f1`-mailout.log
```

Prompt information about the program termination

As the termination of a mailing can be regarded as a critical event, there is a process (user openemm, process emerg3.py) that permanently scans the log file *log/backend.log* for program errors (Java exceptions) and known runtime errors (currently only OutOfMemory) and sends a mail to the Linux user "root" of the system on which the "merger" service is running if they occur. If these mails are to be sent to a different recipient, this can be changed via the system configuration (syscfg:exception-mail-recipient comma-separated list of e-mail addresses).

Relevant log data

The backend is made up of various individual components, each of which writes separate log files. Here is a list of the files and the context in which the underlying process is part of the processing.

Merger

- bav-relay3: This process forwards incoming mails to the "mailloop" service for bounce processing.
- emerg3: This process reads the log file of the process that generates the XML packages and sends mails if an unexpected event has occurred here, see also the chapter on program termination.
- enqueue3: This process is responsible for moving action-based mailings that are ready to be sent from the *var/spool/SIDESHOW* directory to the *var/spool/META* directory.
- generate3: This process initiates the sending of scheduled mailings and date-based mailings. If such a mailing is not sent as expected, it can be checked here whether the mailing was started. Mailing prioritization and dispatch time optimization are also processed within the process.
- mailout: This is the output of the generation of the XML packages. Problems can be found here if mail generation is aborted.
- pickdist3.py: This process generates the finished mails from the ready-to-send XML packages and transfers them to the MTA.
- servant3: (from version 25.04) this service provides supporting functions for internal processing.

- update3: This process writes the statistics data provided (user openemm, process fetch3) to the database. See also above for *fetch3*, if the statistics data is missing in EMM and *fetch3* does not show any errors, further research can be carried out here.
- trigger3: This service provides a minimalist HTTP server for internal communication between the services.
- xmlback: This processes the XML packages and generates the final mails from them.
- slrtscn3: This process interprets the log file of the MTA and generates prepared dispatch statistics from it. If these are missing within EMM, it can be checked here whether there are already problems processing the log files.
- bav-update3: This process reads the bounce filter configurations from the database and makes them available to the processing process (user openemm, process bav). If mails to a bounce address are not accepted, it can be checked here whether the configuration could be provided correctly.
- bav: This is the process that checks whether a bounce filter exists for incoming mails. Here you can check how the process behaves with incoming addresses.
- bavd3: This processes the incoming mails, handles bounces and forwards messages to set forwarding addresses if necessary. If an expected forwarding does not seem to work, but the mails have been accepted, you can look here for a possible cause.

Mail tracking by the system

It is useful to have the following values to hand here:

- Email address
- Customer ID
- Mailing ID
- Company ID (client)

Note: The tracking of admin and test mailings is described at the end of the list.

Creation successful

The first step is to check whether the XML package was generated successfully. This should be done in the "mailout.log" log file on all frontend servers for action-based mailings and on the merger for all other mailings. Here you can search for the lines for the mailing (as described above under "Programmatic termination") (the easiest way is to search for "/<MailingID>/"). If one of the last lines ends with the text "Successful end", then the creation of the XML package should be successfully completed.

Note: If the log level is set too low (other than DEBUG), this line is not output. In this case, the log level must be set accordingly in the database and the application restarted. The log level can be updated with the statement:

UPDATE config_tbl SET value = 'DEBUG' WHERE class = 'mailout' AND name = 'ini.loglevel';

If no line is updated here, it can be inserted with this statement:

INSERT INTO config_tbl (class, name, value) VALUES ('mailout', 'ini.loglevel', 'DEBUG');

Localize XML package

The "merger" service checks whether there are still XML packages for the mailing in the *var/spool/META* directory (see "Structure of the file name of the XML packages" above). These have not yet been sent. Otherwise, you can search for these files in *var/spool/ARCHIVE/<timestamp>* (where timestamp is in the format YYYYMMDD).

Once you have found the files, you can identify the appropriate XML package for a specific recipient with the command:

```
zgrep -l 'customer_id="<CustomerID>"' *<MailingID>=*xml.gz
```

If the recipient cannot be found in one of the XML packages, it can be assumed that they were not included in the mailing set. In this case, you should check whether the recipient is active on the mailing list of the mailing and whether they meet the set target group. You should also check whether the mailing was sent with a list split or as a follow-up mailing, which could exclude the recipient.

If the recipient is contained in a file that is still under *var/spool/META*, this package has not yet been processed. The timestamp in the file name can be used to determine whether the package should already be sent (this must be the same as the current time or in the past) or whether it is not yet ready. In the latter case, you simply have to wait until the time is reached, otherwise you have to check whether the "npickup3" service is running and view its log file if necessary.

If the XML package can be found under *var/spool/ARCHIVE/<timestamp>*, then the package has at least already been transferred to a mailer. To do this, search for the file name of the XML package in the log file of the "npickup3" service for the day of dispatch.

Example: Assuming the file name of the XML package is

```
AgnMail-1=D20200407151951=610=609515=0001=liaMngA.xml.gz
```

then the call

```
grep AgnMail-1=D20200407151951=610=609515=0001=liaMngA.xml.gz var/log/20200407-*-
npickup3.log
```

leads to this result:

```
[07.04.2020 15:19:58] 14081 INFO/npickup: Sent /home/merger/var/spool/META/AgnMail-
1=D20200407151951=610=609515=0001=liaMngA.xml.gz to 127.0.0.1:INCOMING
[2020-04-07 15:19:58] 14081 INFO/npickup: Sent 00-AgnMail-
1=D20200407151951=610=609515=0001=liaMngA.xml.gz to 127.0.0.1:INCOMING
```

This means that this package has been copied to the server "127.0.0.1" (this sample data is taken from a test system that has only one active "mailer" service running on the same system).

Note 1: The second file, 00-AgnMail-1=D20200407151951=610=609515=0001=liaMngA.xml.gz, serves as a marker for the "mailer" service that the actual file "AgnMail-1=D20200407151951=610=609515=0001=liaMngA.xml.gz" has been transferred in full and can be processed.

Note 2: If the log file has already been compressed, the search call must look like this:

```
zgrep AgnMail-1=D20200407151951=610=609515=0001=liaMngA.xml.gz var/log/20200407-*-
npickup3.log.gz
```

Check dispatch

If the processing of the XML package was successful, the dispatch can now be checked. The easiest way to do this is to use the "maillog-search" command. This searches the log files of the MTA (by default /var/log/maillog*) for the recipient's entry.

Example: If the recipient has the CustomerID 1234 and the mailing has the ID 567, this command can be used to search for the corresponding information:

```
maillog-search 'customer:1234 and mailing:567'
```

If you know (roughly) in which log file the data can be found, you can specify this with the "-p" option, e.g.

```
maillog-search -p /var/log/maillog 'customer:1234 and mailing:567'
```

This is useful if the log files are larger and the search would take longer.

These outputs should then reveal the final dispatch status.

If you call the command "maillog-search" with the option "-?", a help text about the possible uses is displayed.

Check admin and test dispatch

As these are sent directly on the "merger" service, the log file of the MTA on this server can be consulted directly.

Checking the processing of statistical data

As soon as a mailing has been processed on the "mailer" service, statistical data is collected for display in EMM and written to the database. This data contains information about the mailings generated (account.log) and the dispatch status (extbounce.log). Depending on the application, there may also be other files with the extension ".log".

If the statistics are not available in EMM, the services and log files of the processes can be checked in the order described here.

Collection

The data for the generated mailings is written to the "mailer" service by the processing process "xmlback". This data is stored in the first step under *log/account.log.* The dispatch statuses are determined on the "mailer" service by the "slrtscn3" process from the log files of the MTA and stored under *log/extbounce.log.* In the case of delayed bounces, this data is obtained in the "mailloop" service by the "bavd3" process and also stored there under *log/extbounce.log.*

Processing the data

The "update3" process of the "merger" service processes the merged files in the *log* directory and writes them to the corresponding tables in the database.

Tools

There are some tools that have been created specifically for the backend area of EMM. These have arisen from internal requirements and thus primarily cover the needs that have arisen during operation at AGNITAS. All commands not listed here are either only used internally by other programs, are incomplete or no longer up-to-date and will be removed in a future version.

- lsi (list incoming, service "merger"): Lists the status of the respective *INCOMING* directory for all "mailer" services. This allows you to centrally view the current load at XML package level.
- lsq (list queue, service "merger", "mailer"): Lists the contents of the queues of the service and the MTA.

- maillog-search ("mailer" service): Searches and organizes the log files of the MTA.
- server-from-database ("merger" service): Displays the servers of a service configured in the database, e.g. server-from-database mailer
- slog ("merger" service): Displays an overview of the mailings generated.