Sendmail X README

Claus Aßmann

June 22, 2005

Contents

1 Sendmail X

This distribution contains the source code for sendmail[1] X which implements a Message Transfer System (MTS)[2].

Please report bugs and provide feedback either to the developers list[Aßma] if you are subscribed or directly to[3]:

```
< smx + feedback (at) sendmailx . org >
```

See the LICENSE file before using the source code.

[1] sendmail is a trademark of Sendmail, Inc.
[2] often also called MTA: Message Transfer Agent
[3] Sorry for the obfuscation, replace `(at)` with `@` and remove the spaces.

1.1 Documentation

The document "Sendmail X: Requirements, Architecture, and Functional Specification" [Aßmb] provides the background about the sendmail X design, its architecture, as well as the functional specification, and details about the implementation.

1.2 Version

This document has been written for sendmail X.0.0.Alpha4.0, see also the greeting of the SMTP server and the version output of the main components. See Appendix A.2 for information about version naming.

1.3 Current State

There are still some error conditions which may not be handled gracefully, i.e., in case of some resource problems (e.g., out of memory or out of disk space) the system may abort. See Section 6.10.1 how to deal with those conditions. The software is running since 2004-01-01 as MTS on the main machine of the author without any significant problem, i.e., it never lost any mail.

Feedback about the code, the documentation, as well as patches and enhancements are highly appreciated; please send it to the address given above.

1.4 Main Components of sendmail X

The following programs are part of the sendmail X message transfer system (MTS):

- Queue manager (QMGR)
- SMTP server (SMTPS)
- SMTP client (SMTPC)
- address resolver (SMAR)
- main control program (MCP)

Information about each component will be given in the appropriate sections. Complete documentation and background information can be found in [Aßmb]. Section 10 describes the data flow in sendmail X.

This version of sendmail X does not come with a local delivery agent nor a mail submission program. See Sections 6.3.1 and 6.2 which programs can be used to achieve the desired functionality.

1.5 Typographical Conventions

In this documentation, a command written as

```
$ command
```

should be executed as an unprivileged user. Only a command written as

```
# command
```

should be executed as the superuser.

If a command contains components that need to be replaced by values that depend on the environment or the local configuration, then it is usually written as a macro, e.g., `$LOGFILE`.

2 Building sendmail X

sendmail X uses GNU autoconf for configuration. Hence you can build it (after verifying the distribution as explained in appendix A.1 and unpacking it) as follows:

```
$ mkdir obj.$OS && cd obj.$OS && $PATHTO/smX-$VERSION/configure $OPTIONS
```

Obviously you have to replace `$OS` and `$VERSION` as well as `$PATHTO`. It is also possible to build sendmail X in the source tree, however, this is discouraged:

```
$ ./configure && make && make check
```

Note: do *not* run this as `root`; this is not just a basic security measure (*only* use a privileged account if it is really required), but most of the programs refuse to run with `root` privileges.

2.1 Required Packages

sendmail X requires Berkeley DB 4.2 (or 4.1). Do not use 4.3.27 as it causes a crash in 64 bit mode on Solaris 5.8/9 at least[4]. If this software is not installed in a location that the compiler or linker use by default, you have to tell `configure` about it, e.g.,

```
$ B=/usr/local/BerkeleyDB.4.2
$ ./configure --with-bdb-libdir=$B/lib --with-bdb-incdir=$B/include
```

Suggestions how to let `configure` find this location are welcome! Note: sendmail X is supposed to ship with Berkeley DB which will probably happen after the alpha development stage.

2.2 Configuration Options

Beside the usual `configure` options like `--prefix` a few sendmail X specific configuration options are available:

–**enable-TLS** Enable check for STARTTLS support. The default is `yes`, i.e., `configure` tries to determine whether OpenSSL is available on the machine. See Section 2.3 (`SM_USE_TLS`) for details.

–**enable-SASL** Enable check for AUTH support. The default is `yes`, i.e., `configure` tries to determine whether Cyrus SASL v2 is available on the machine.

[4] "Private database environments on 64-bit machines no longer drop core because of 64-bit address truncation. [11983]' [Slea]

–**with-sasl-libdir=path** Path to directory containing Cyrus SASL v2 library.

–**with-sasl-incdir=path** Path to directory containing Cyrus SASL v2 include files.

–**with-bdb-libdir=path** Path to directory containing Berkeley DB library.

–**with-bdb-incdir=path** Path to directory containing Berkeley DB include files.

–**enable-pmilter** Enable policy milter protocol. Note: this is experimental, see Section 8.

To get the current list of configuration options, use `./configure --help`.

2.3 Compile Time Options

SM_USE_TLS Support STARTTLS. Requires OpenSSL 0.9.6 or newer [Pro]. Note: check the OpenSSL website [Pro] for security announcement and be aware that due to the complexity of the software it may cause (security) problems.

SM_USE_SASL Support SMTP AUTH. Requires Cyrus SASL version 2.1.18 or newer [SAS]. Note: check `http://asg.web.cmu.edu/cyrus/` and `http://asg.web.cmu.edu/sasl/` for security announcement and be aware that due to the complexity of the software it may cause (security) problems.

SM_HEAP_CHECK Turn on various heap checks and keep track of memory usage.

A simple way to set compile time options is to use:

```
$ CFLAGS="-DSM_USE_TLS" ./configure
```

A more complicated example is:

```
$ CFLAGS="-O -g -DSM_USE_TLS -I/usr/local/include" LDFLAGS="-L/usr/local/lib" \
 ./configure
```

Hint: it is useful to write the command line into a local file that can be reused for subsequent builds and versions.

For more compile time options see Appendix E.

Note: if configure has problems with OpenSSL because you do not have KerberosV installed, add

```
$ CPPFLAGS="-DSM_USE_TLS -DOPENSSL_NO_KRB5"
```

3 Test Programs

```
$ make check
```

will run all test programs; currently those tests take about one hour to run on a standard workstation. For each of the test programs one line is printed to denote whether the test succeeded, i.e., the output consists of lines with the marker `PASS:` or `FAIL:` and the name of the test program program. Additional output might be generated by the test programs themselves, e.g.,

`2 of 2 tests completed successfully`,

or some debug output. The debug output may even indicate an error, but only a final `FAIL:` indicates a test failure. Some tests depend on compilation options and are only conditionally enabled; others may depend on environment variables, see 3.1. For disabled tests `SKIP` is shown.

Since some of the tests may fail (see Section 3.2) and `make` will usually stop after encountering an error, it might be required to use

```
$ make -i check
```

to perform all tests.

3.1 Environment Variables used by Test Programs

Some of the test programs perform DNS lookups. These lookups may use domains that are under control of the sendmail X author. To disable those tests set the environment variable `SM_NO_DNS_TEST`. To set a different timeout than the default, the environment variable `SM_DNS_TIMEOUT` can be used, however, it may not be obeyed by all DNS test programs.

Most test scripts that perform multiple checks use the environment variable `STOPONERROR` to stop on the first error that occurs.

Setting `SM_NO_LOG_TEST` disables some tests that use `syslog(3)`.

Some tests that take a very long time may be disabled by setting `SM_NO_SLOW_TEST`.

3.2 Known Test Program Problems

The test programs which involve the full system (and of course sendmail X itself) will not work on an NFS mounted partition. You have to run the tests on a local disk (that is a restriction of Berkeley DB).

- `connctl.sh` will fail on systems that have neither `inet_pton(3)` nor `inet_aton(3)`. Fix: upgrade your OS or write a replacement function and put it into `librepl/`.
- `t-evthr-sig.sh` fails on Linux systems that use a thread implementation that is not POSIX compliant. The test is currently disabled on all Linux versions. Note: if you know a simple way to figure out whether the OS actually provides POSIX compliant pthreads, please let me know.
- `t-hostname` fails on systems where `gethostname()` does not return any FQHN at all (e.g., default SunOS 4/5 installations). Add the FQHN as alias to `/etc/hosts` to solve this problem, e.g.,

  ```
  10.1.2.3    myname myname.my.domain
  ```

- `t-smar-0.sh` and `t-smar-3.sh` may fail sometimes due to DNS timeouts. Run the tests again.

- `t-mts-icr.sh` and `t-mts-ocr.sh` rely on some timing issues; they try to test incoming/outgoing rate control.

For more information about possible test program problems that are not listed in the next section see Appendix F.

3.2.1 Known Test Program Problems specific to an OS/setup

FreeBSD systems when running in a `jail(8)` exhibit the following problems:

- The test programs for SMAR which perform DNS lookups can fail because UDP does not work in a `jail(8)` as expected. A workaround for this is to use the `-U` option for `smar` which can be achieved by setting the environment variable `SMAROPTS` to that value.
- Connections from localhost to the SMTP server do not have `127.0.0.1` as source IP address, but the IP address of a NIC. Hence relaying must be allowed for it by setting the environment variable `SM_SERVER_OPTIONS` to the option `-C` and the IP address, e.g., `-C 10.2.3.4`. Moreover, because the tests `chkmts/t-mts-icr.sh` and `chkmts/t-mts-ocr.sh` rely on connections coming from `127.0.0.1` they will fail too.

MacOS 10.3.4 has a problem with `sigwait(3)`, see Apple's bug 3675391; hence sendmail X does not work on this OS (and other versions that have the same bug).

4 Installing sendmail X

sendmail X needs several users to provide optimum separation of privileges and to maximize security. Currently there are four required accounts (the numbers for uid and gid are examples only); the last one listed below (`smx`) is not really required:

```
smxs:*:260:260:Sendmail X SMTPS:/nonexistent:/sbin/nologin
smxq:*:261:261:Sendmail X QMGR:/nonexistent:/sbin/nologin
smxc:*:262:262:Sendmail X SMTPC:/nonexistent:/sbin/nologin
smxm:*:263:263:Sendmail X misc:/nonexistent:/sbin/nologin
smx:*:264:264:Sendmail X other:/nonexistent:/sbin/nologin
```

with the corresponding groups:

```
smxs:*:260:
smxq:*:261:
smxc:*:262:smxs
smxm:*:263:smxs,smxq
smx:*:264:
```

A simple shell script to setup the directories, files, etc. as described below is available in `misc/sm.setup.sh.in`. This script is modified by `configure` to create `misc/sm.setup.sh` (in the build directory) which is invoked when

```
# make install
```

is called. Most defaults in the installation script `misc/sm.setup.sh` can be overridden with environment variables (default is listed in square brackets):

- `SMXCONFDIR`: [`/etc/smx`] configuration directory.
- `SMXQDIR`: [`/var/spool/smx`] queue directory; communication sockets are created in this directory by default too.
- `SMXLOGDIR`: [`.`] logging directory (relative to `SMXQDIR`). If logging is done via `syslog(3)` then this directory is not really used.
- `SMXS` [`smxs`] SMTP Server user and group
- `SMXC` [`smxc`] SMTP Client user and group
- `SMXQ` [`smxq`] QMGR user and group
- `SMXM` [`smxm`] address resolver (misc) user and group
- `SMX` [`smx`] generic (configuration etc) user and group
- `SMXLG` group for logfiles; the install program tries `operator`, `sysadmin`, and `root`.

Notes:

1. The users and groups *must* be created before `make install` is invoked.
2. `misc/sm.setup.sh` requires some programs that are only available if `make check` has been run before.
3. `misc/sm.setup.sh` will not overwrite existing files or directories, hence it does not work for upgrading a system if configuration files or directory/file owners need to changed.

4.1 Directories, Files, and Permissions

`make install` (i.e., `misc/sm.setup.sh`) will create all the required directories and files with the correct permissions provided the users and groups have been set up properly. This section shows explains what the created structure looks like.

The CDB[5] directories (`0-9`, `A-F`) must be owned by `smxs` and have group `smxq` with the permissions `0771`:

```
drwxrwx--x  2 smxs  smxq        0/
```

The main (DEFEDB[6]) and incoming queues (IBDB[7]) must belong to `smxq` and should not accessible by anyone else:

```
drwx------  2 smxq  smxq        defedb/
drwx------  2 smxq  smxq        ibdb/
drwx------  2 smxq  smxq        ibdb/ibdb/
```

[5] Content DataBase: the contents of mails are stored here; see Section 10.
[6] DEFerred Envelope DataBase
[7] Incoming Backup DataBase

Note: Do *not* use an NFS mounted partition for the deferred queue (`defedb`) (that is a restriction of Berkeley DB; already mentioned in 3.2).

Mailertable, aliases map, and other maps for SMAR (see Section 5.8.3) should belong to `smxm` and can be readable as local conventions require:

```
-rw-r--r--  1 smxm  smxm         mt
-rw-r--r--  1 smxm  smxm         aliases.db
```

In general, maps should be owned by the user id of the program that uses them, e.g., `smxq` owns the QMGR configuration map `conf.db` (see Section 5.7.1).

The sendmail X configuration file can either belong to `root` or the generic sendmail X user:

```
-rw-r--r--  1 smx   smx          smx.conf
```

The directories in which the communication sockets between QMGR and the other programs are located must belong to `smxq` and be group accessible for the corresponding program:

```
drwxrws---  2 smxq  smxm        qmsmar/
drwxrws---  2 smxq  smxc        qmsmtpc/
drwxrws---  2 smxq  smxs        qmsmtps/
```

The directory in which the communication socket between MCP and SMTPS is located must belong to `smxs`:

```
drwxr-x---  2 smxs  smxs        smtps/
```

The logfiles must be owned by the corresponding user and may have relaxed group (or even world) read permissions:

```
-rw-r-----  1 smxq  operator   qmgr.log
-rw-r-----  1 smxm  operator   smar.log
-rw-r-----  1 smxc  operator   smtpc.log
-rw-r-----  1 smxs  operator   smtps.log
```

5 Configuration of sendmail X

Configuration of sendmail X can be done via command line parameters or via a configuration file (the latter is preferred, the former offers only a small subset of the available configuration options). If a configuration file and command line options are specified, then the options are currently processed in order, i.e., later settings override earlier ones for the same options. Information about the former is available by invoking a program with the option `-h` (MCP currently uses `syslog(3)` instead of `stderr`), it will show the usage as well as the default values. The syntax of the configuration files is specified in the following sections. To actually use a configuration file, the option `-f $CONFIGFILE` must be used, otherwise the programs use only the built-in default values, but not a configuration file. Option '`-V`' can be used to show version information, specifying '`-V`' multiple times shows more detail, e.g., '`-VVVVV`' will show the configuration data including the default value for (almost) every option.

Some configuration options can be set via Berkeley DB hash maps, these maps are: `conf` for QMGR (see Section 5.7.1) and `access` for SMTPS (indirectly via the address resolver, see Section 5.8.3).

5.1 Configuration File Syntax

The grammar for a sendmail X configuration file is very simple:

conf	::=	entries
entries	::=	entry *
entry	::=	option | section
section	::=	keyword [name] "{" entries "}" [";"]
option	::=	option-name "=" rhs
rhs	::=	value ";" | "{" value-list "}" [";"]

A configuration file consists of *entries*, each *entry* is either an *option* or a *section*. An option has a *name*, an equal sign, and a *value* terminated by a semicolon or a (bracketed) list of values separated by comma. A *section* consists of a *keyword*, an optional *name*, and a (bracketed) sequence of *entries*. Keywords and options are not case sensitive. The layout of a configuration file does not matter i.e., indentation and line breaks are irrelevant (in general, but see below for strings).

5.1.1 Configuration File Values

Values in a configuration file are usually strings or numbers. If a string is used, then it should be quoted, unless it contains no special characters which are treated specially by the grammar. If a string is very long it can be broken into substrings spread out over several lines (just like strings in ANSI C), e.g.,

```
somemessage = "this is a very long string which is spread "
   "out over several lines because otherwise it is too "
   "hard too read.";
```

In some cases it is possible to have *units* for values. Currently time and size values make use of this feature. Valid time units are `w` for weeks, `d` for days, `h` for hours, `m` for minutes, and `s` for seconds. Valid units for size are `B` for bytes, `KB` for kilo bytes, `MB` for mega bytes, and `GB` for giga bytes. It is allowed to specify a sequence of numbers and units, e.g., `1h 5m 12s`. Unless otherwise specified, the default units for times and sizes in a configuration file are `s` and `B`, respectively; for those values these units can be used.

5.2 Configuration for MCP

The installation script creates the file `smx.conf` in the configuration directory (`/etc/smx`, see Section 4). Check the comments in the file and edit it if required. A configuration file contains a sequence of sections, here is an abbreviated example:

```
smtps {
  port = 25;
  mcp_type = pass; pass_fd_socket = smtps/smtpsfd;
  user = smxs;
  path = /usr/libexec/smtps;
  arguments = "smtps -f /etc/smx/smx.conf";
}
qmgr {
  mcp_type = wait;
```

```
  user = smxq;
  restartdependencies = { smtps, smtpc, smar };
  path = "/usr/libexec/qmgr";
  arguments = "qmgr -f /etc/smx/smx.conf";
}
```

The valid options in an entry are:

1. `mcp_type`: one of `nostartaccept`, `accept`, `pass`, `wait` (required).
2. `port`: port number on which service should listen.
3. `address`: IP address on which service should listen [default: `INADDR_ANY`]
4. `socket`: This is a subsection that specifies the socket on which a service should listen.
   (a) `name`: path of Unix Domain socket on which service should listen.
   (b) `umask`: `umask` for socket.
   (c) `user`: owner of socket.
   (d) `group`: group of socket.
5. `min_processes`: minimum number of processes to start [default: 1].
6. `max_processes`: maximum number of processes to start [default: 1].
7. `pass_fd_socket`: path of Unix Domain socket to pass a file descriptor to the service.
8. `user`: user id to run service (process).
9. `group`: group id to run service (process).
10. `restartdependencies`: list of services that need to be restarted when this one is restarted (or crashes).
11. `path`: path to program to execute (required).
12. `arguments`: arguments (argv), must start with name of program, see `execv(2)` (required).
13. `pass_id`: option to use to pass a unique, numeric identifier to the spawned process via the command line. The option will be inserted as first argument. Example:

    ```
    smtpc { pass_id = "-i"; min_processes = 4; max_processes = 4;
      path = /usr/libexec/smtpc; arguments = "smtpc -f smx.conf"; }
    ```

    will cause MCP to start four `smtpc` processes, each with the options `-i` *ID* `-f smx.conf` where *ID* is replaced with a unique identifier.
14. `use_id_in_logfile_name`: if more than one process can be started then it might be useful to have unique logfiles unless the processes use `syslog(3)`. This option cause MCP to include a unique identifier (the same as for `pass_id`, which must be used too) in the logfile name. By default the logfile has the name of the section (or the section keyword if no section name is given), preceeded by the log directory (option `-L` for MCP), and `.log` appended. If `use_id_in_logfile_name` is turned on, then the numeric id is added before the extension, e.g., `/var/log/smx/mailer0.log` for `-L /var/log/smx/` and a section with the name `mailer`.

Notes:

- At most one of `port` and `socket` can be specified. This is for programs that run as servers and communicate via standard protocols, e.g., SMTP or LMTP, with clients.
- `pass_fd_socket` must be specified for `mcp_type = pass`, in this case MCP binds to the specified port and passes it via the Unix domain socket to the started process.
- For `mcp_type = nostartaccept` MCP waits for incoming connections, and then starts a process to handle a single connection.
- For `mcp_type = accept` MCP binds to the socket and then starts a process to handle the connections without waiting for an actual request.
- For `mcp_type = wait` MCP simply starts the requested number of processes without passing them any open connections. This is intended for services (processes) that do not communicate with external clients.

5.3 Example Configuration File

A configuration file for sendmail X contains several sections: a global section which specifies the locations of sockets and directories that are used by multiple components, and one section each for QMGR, SMAR, SMTP server, and SMTP client. Other sections may define services that are started by MCP, e.g., a local mailer.

```
CDB_base_directory = "/var/spool/smx/";

qmgr {
  AQ_max_entries = 8192;
  smtpc_initial_connections = 19;
  smtpc_max_connections = 101;
  smtps_max_open_connections = 5;
  smtps_max_connection_rate=160;
  max_errors_per_bounce=16;
  wait_for_server = 4; wait_for_client = 4;
  mcp_type = wait; user = smxq;
  restartdependencies = { smtps, smtpc, smar };
  path = "/usr/libexec/qmgr"; arguments = "qmgr -f /etc/smx/smx.conf";
}

smtps { flags = {8bitmime}; CDB_gid = 261; IO_timeout = 61;
  port = 25; mcp_type = pass; pass_fd_socket = smtps/smtpsfd;
  user = smxs; path = /usr/libexec/smtps;
  arguments = "smtps -f /etc/smx/smx.conf"; }

smtpc {
  Log_Level = 12; Debug_Level = 1; IO_timeout = 66; wait_for_server = 4;
  mcp_type = wait; user = smxc; path = "/usr/libexec/smtpc";
  arguments = "smtpc -f /etc/smx/smx.conf"; }

smar {
```

```
  Log_Level = 12;
  nameserver = {10.10.10.9, 127.0.0.1};
  DNS_timeout = 6;
  mcp_type = wait; user = smxm; restartdependencies = { smtps, qmgr };
  path = "/usr/libexec/smar"; arguments = "smar -f /etc/smx/smx.conf";
}
```

5.4 Common Global Configuration

All of the following options have defaults and should only be changed if necessary.

1. `hostname`: set the hostname to use for the various components. This can be set if `gethostbyname(3)` does not return a valid (fully qualified) hostname.
2. `CDB_base_directory`: base directory of CDB; this should either be empty (which is the default) or a path to a directory including a trailing slash; the CDB library currently simply appends the directory names (see Section 4.1) to it. It might be useful to move some subdirectories to different disks (by creating (symbolic) links (`ln(1)`)) to spread the I/O load.
3. `SMAR_socket`: socket created by the address resolver over which clients (SMTPS, QMGR) can send requests.
4. `SMTPC_socket`: communication socket between SMTPC and QMGR.
5. `SMTPS_socket`: communication socket between SMTPS and QMGR.

The sockets are currently Unix domain sockets only, hence the value is simply the (path)name of the socket.

5.5 Common Configuration Options

There is currently one configuration option which is the same across all modules but is not specified in the global section because it is specific to the individual modules.

1. `log`: this is a section with the following options:
   (a) `facility`: see `syslog(3)` for valid facilities, here are some valid options provided the OS offers them: `daemon`, `mail`, `auth`, `local0`, etc.
   (b) `ident`: identification string for `openlog(3)`, defaults to name of the modules. It might be useful to chose other identifiers, e.g., `smXmta` or `smxQMGR`.
   (c) `options`: options for `openlog(3)` (without the leading `LOG_` as provided by the OS, e.g., `pid` or `ndelay`.

   Example:

   ```
   qmgr { log { facility = daemon; ident=smX-qmgr; } }
   smtps { log { facility = mail; ident=smX-MTA; } }
   ```

   Note: debug output is currently sent to `stdout`; `syslog(3)` is not used for debugging.

All modules have an option to set the amount of logging (`log_level`) that should be done. The larger the value the more information is logged. For normal operation a value of 9 is recommended. During testing values of 12 to 14 are useful.

5.6 Pathnames for Files, Directories, and Maps

Most names of files (including maps) and directories in the configuration file have a default name (compiled into the binary) without an absolute path, e.g., `aliases.db`. If a pathname is not explicitly set in the configuration file or does not use a absolute path (i.e., begins with a slash), then the default is relative to either

1. the configuration directory: maps and configuration files, e.g., `aliases.db` and `cert_file`.
2. the main queue directory: pathnames of sockets, and databases to store envelope information (IBDB, DEFEDB) or message contents (CDB).

The paths for files mentioned in case 1 are taken relative to the path of the configuration file which is passed via the `-f` option to the various modules. For example: if SMAR is started as

```
/usr/libexec/smar -f /etc/smx/smx.conf
```

then the pathname used for the aliases map is `/etc/smx/aliases.db`. This applies to the SMAR maps aliases, mailertable, and access (5.8.2), the QMGR `conf` map (5.7.1), and the STARTTLS related files and directories used by the SMTP server (5.9) and client (5.10).

The paths for files mentioned in case 2 are taken relative to the execution directory. All sendmail X modules should be started (via MCP) in the main queue directory (default: `/var/spool/smx`, see Section 4).

See the various configuration options explained below how to override the defaults. Note: relative pathnames specified in the configuration file are (currently) always relative to the main queue directory.

5.7 Configuration for QMGR

The following configuration options are valid for QMGR:

1. `AQ_max_entries`: maximum number of entries in AQ (active queue) (unit: entries). Note: this value must be larger than the largest number of recipients accepted by a single transaction.
2. `SMAR_timeout`: timeout in address resolver, i.e., how long to wait for a result from SMAR (unit: s). Note: this value must be larger than the DNS timeout and it should take alias expansion into account.
3. `debug_level`: debug level (only if compiled with QMGR_DEBUG).
4. `control_socket`: specify path name of "control" socket (for querying and making requests). This socket can be used by the query/control program `qmgrctl`, see 6.6.3.
5. subsection `DEFEDB`. Note: The Berkeley DB documentation [Sleb] should be consulted before modifying any of these options (except the first two).
   (a) `base_directory`: Home directory for DEFEDB.

(b) `log_directory`: Log directory for DEFEDB. For better performance, this directory can be set to point to a different disk than the base directory of DEFEDB.

(c) `page_size`: DB page size.

(d) `cache_size`: DB cache size.

(e) `KBytes_written_for_checkpointing`: If non-zero, a checkpoint will be done if more than the amount of KBytes of log data have been written since the last checkpoint (unit: KB).

(f) `delay_between_2_checkpoints`: Minimum delay between two checkpoints (unit: s).

6. `delivery_timeout`: timeout for a single delivery attempt (unit: s). This value should be large enough that even big mails can be delivered over a slow link before the QMGR considers the delivery attempt a failure because the delivery agent did not return a result yet.

7. `DSN_max_delay`: maximum time for scheduling a DSN (unit: s).

8. `aliases`: select to which part of an e-mail address the aliases DB should be applied:

   (a) `localpart` only to the local part of local e-mail addresses,

   (b) `localdomains` to the full address of local e-mail addresses,

   (c) `all` to the full address of all e-mail addresses, even remote ones.

9. `flags`: configuration flags:

   (a) `reuse_connection`: try to reuse open SMTP connections for delivery. Note: this feature is still experimental.

   (b) `header_only_in_bounce`: include only the headers in a bounce message; by default the first bounce includes the entire message and subsequent ones include only the headers.

   (c) `DSN_in_MIME_Format`: Use MIME to structure a DSN. Note: this is not (yet) a DSN in the format specified by RFC 3464 [MV03].

10. subsection `IBDB`:

    (a) `max_commit_delay`: maximum time between commits to IBDB (unit: μs)

    (b) `size`: maximum size of each IBDB file (unit: B).

    (c) `max_open_TAs`: maximum number of open transactions in IBDB before a commit is performed (unit: entries).

    Note: the configuration file offers no way to specify a base directory for IBDB, however, the directory can be easily moved elsewhere and a (symbolic) link (`ln(1)`) can be added.

11. subsection `IQDB`:

    (a) `max_cache_entries`: maximum number of entries in IQDB cache (unit: entries).

    (b) `hash_table_entries`: size of hash table for IQDB (unit: entries).

12. `log_level`: logging level.

13. `max_errors_per_bounce`: maximum number of error messages (failed recipients) in a bounce (DSN) (unit: entries).

14. `min_disk_space`: minimum amount of free disk space (unit: KB). This value should be significantly larger than the maximum size of a message to be accepted by the SMTP server.

15. `OCC_max_entries`: size of open connection cache (unit: entries).

16. `ok_disk_space`: amount of free disk space at which normal operation continues (unit: KB). Must be larger than `min_disk_space`.
17. `queuereturn_timeout`: maximum time in queue (unit: s).
18. `queuedelay_timeout`: send delay warning ("delayed DSN") after this amount of time in queue (unit: s). To turn off delayed DSNs set this to a value bigger than `queuereturn_timeout`.
19. `retry_max_delay`: maximum time for retrying a delivery (unit: s).
20. `retry_min_delay`: minimum time for retrying a delivery (unit: s).
21. `smtpc_initial_connections`: initial number of outgoing connections to a single host (unit: entries).
22. `smtpc_max_connections`: maximum number of outgoing connections to a single host (unit: entries).
23. `smtps_max_connection_rate`: maximum incoming connection rate from a single host (unit: connections/60s).
24. `smtps_max_open_connections`: maximum number of open incoming connection from a single host (unit: entries).
25. `tests`: testing only.
26. `wait_for_client`: maximum amount of time to wait for a client to become available (unit: s)
27. `wait_for_server`: maximum amount of time to wait for a server to become available (unit: s)
28. `conf`: Name of configuration map (including extension), see Section 5.7.1 [default: `conf.db`]. See also Section 5.6.

5.7.1 Configuration Map for QMGR

QMGR implements a "slow start" algorithm to control the number of concurrent connections to one IP address. Initially, it will at most create a (small) number of open connections up to a specified initial limit. For each successful delivery, the allowed number is increased up to specified maximum limit.

For incoming connections, QMGR establishes two limits: the connection rate and the number of open connections.

The Berkeley DB hash map `conf.db` (the file should be owned by `smxq`) can have the following entries:

1. `oci:` this key specifies the initial number of concurrent outgoing connection to an IP address.
2. `ocm:` this key specifies the maximum number of concurrent outgoing connection to an IP address.
3. `octo:` specify the timeout for an entry in the outgoing connection cache.
4. `icr:` this key specifies the maximum rate for incoming connections (per 60s).
5. `ios:` this key specifies the maximum number of concurrently open incoming sessions.

`oci:`, `ocm:`, `icr:`, and `ios:` take an IP address/net as parameter such that the limits can be imposed per IP address/net. For example:

```
oci:127.0.0.1      5
ocm:127.0.0.1     10
oci:10            10
ocm:10            50
oci:               1
ocm:               4
icr:10             5
icr:127.0.0.1    100
ios:127.0.0.1    120
```

Note, however, that the limits apply only to single IP addresses, they are not aggregated for nets. That is, for the example every single host in the IP net 10.x.y.z can have a maximum incoming connection rate of 5 messages per minute.

The default values for these configuration options are set in the binary and can be changed via command line options or the configuration file (see Section 5.7):

1. `-C n` maximum number of concurrent connections to one IP address [default: 100]
2. `-c n` initial number of concurrent connections to one IP address [default: 10]
3. `-O R=n` maximum connection rate per 60s (SMTPS) [default: 100]
4. `-O O=n` maximum number of open connections (SMTPS) [default: 100]

5.8 Configuration for SMAR

5.8.1 Declaring Maps for SMAR

In general, maps must be declared before they can get used. Each map declaration in a configuration file is a named section – the name is used for later references – `map` with the following options:

1. `type`: type of the map; currently one of `hash` (Berkeley DB hash), `sequence`, `socket`, and `passwd`.
2. `file`: the pathname of the `db` file (including the extension) (type `hash` only).
3. `mapname`: name of the map used in the protocol (type `socket` only).
4. `path`: path of Unix domain socket (type `socket` only).
5. `address`: IPv4 address of inet socket. (type `socket` only).
6. `port`: port for inet socket (type `socket` only).
7. `maps`: list of map names to use in the map (type `sequence` only).

Note: for `socket` maps either a Unix domain socket (`path`) or an inet socket (`address` and `port`) must be specified.

Example:

```
map localusers { type = hash; file = "/etc/smx/localusr.db"; }
map otherusers { type = hash; file = "/etc/smx/otherusr.db"; }
map password { type = passwd; }
map seq1 { type = sequence; maps = { localusers, otherusers }; }
map seq2 { type = sequence; maps = { password, otherusers }; }
```

5.8.2 Configuration Options for SMAR

The following configuration options are valid for SMAR:

1. `dnsbl`: specify a DNS based blacklist[8]. This section can be specified multiple times[9]; it has the following required options:

   - `domain`: specify the domain to use for DNS lookups, e.g., `dnsbl.tld`.
   - `tag`: specify the tag to use for lookups in the access map (which must be enabled, see Section 5.9, 4g).

   The client IPv4 address *A.B.C.D* is looked up via DNS as *D.C.B.A.*`domain` querying for an A record. If an A record *W.X.Y.Z* is found, then it is looked up in the access map as `tag`:*W.X.Y.Z.* for temporary and permanent DNS lookup failures the entries that will be checked in the access map are `tag:temp` and `tag:perm`, respectively.

   Notes:

   - DNS lookups in blacklists can be disabled via entries in the access map using the tag `cltaddr`, see Section 5.8.3.
   - currently a colon is added as delimiter after `tag`, this may be removed in later versions to allow for more flexibility; e.g., the configuration option itself can include a delimiter.

   The access map entry should have one of the usual rejection RHSs as explained in 5.8.3. Example: configuration file:

   ```
   smar { dnsbl { domain = dnsbl.tld; tag = dnsbltld; } }
   ```

   access map:

   ```
   dnsbltld:127.0.0.1  error:550 5.7.1 listed at dnsbl.tld as open relay
   dnsbltld:127.0.0.2  error:550 5.7.1 listed at dnsbl.tld as spam source
   dnsbltld:127.0.0.9  error:451 4.7.1 listed at dnsbl.tld as suspicious
   dnsbltld:temp       error:451 4.7.1 temporary lookup failure at dnsbl.tld
   ```

   If multiple DNS based blacklists are specified, the DNS queries are made concurrently but the lookups in the access map are performed in the order in which the blacklists are given; the first successfull lookup is used as result, no further priorization is performed.

2. `DNS_flags`: valid flags are:

   (a) `use_TCP`: use TCP instead of UDP for connections to a nameserver.

   (b) `use_connect`: use `connect(2)` even if using UDP.

[8] This option is modelled after `dnsblaccess` written by Neil Rickert for sendmail 8.

[9] Compile time option `SM_MAX_DNSBL`: currently set to 8.

3. `DNS_timeout`: timeout for DNS requests (unit: s).
4. `log_level`: logging level.
5. `nameserver`: list of up to four IPv4 addresses[10] of nameservers.
6. `local_user_map`: specify a (name of a) map of valid local addresses; the map must have been declared as explained in Section 5.8.1.
7. `address_delimiter`: Delimiter (one character) for address extensions in local part, [default: `'+'`].
8. `aliases`: This is a subsection that specifies the parameters for aliases:
   (a) `name`: Name of aliases map (including extension) [default: `aliases.db`].
   (b) `flags`:
      i. `localpart`: the aliases map contains only localparts of addresses and those are only looked up for local addresses.
      ii. `local_domains`: the aliases map contains fully qualified addresses which are only looked up for local addresses. This can be used similar to virtual users in sendmail 8, e.g.,
      ```
      vuser1@virt1.tld:   user1
      vuser2@virt1.tld:   user2
      vuser3@virt2.tld:   user3
      ```
      iii. `all_domains`: the aliases map contains fully qualified addresses which are only looked up for any domain.
      iv. `replace_macros`: Replace macros in the RHS of the map entries by the appropriate value, see Section 5.11.3.
      v. `preserve_domain`: If the RHS of an entry is an unqualified address, do not append the local hostname to it but the domain of the original address, i.e., preserve the original domain.
9. `mailertable`: This is a subsection that specifies a mailertable, currently it has only one valid entry: `name`: Name of mailertable. [default: `mt`].
10. `access`: This is a subsection that specifies the access control map, currently it has only one valid entry: `name`: Name of access map (including extension) [default: `access.db`].

5.8.3 Configuration Maps for SMAR

SMAR requires a mailertable, and it can make use of an alias map as well as an access map, all of which are described in the subsequent sections.

Access Map To activate the access map the flag `access` (see Section 5.9, 4g) (or the option `-a`) must be given to the SMTP servers. All entries consist of a left hand side (LHS, key) which in turn has a tag and a (partial) address and a right hand side (RHS, value). Valid tags are:

[10] 4 is the default value for the compile time option `SM_DNS_MAX_TSKS`

Tag	refers to
`from:`	envelope sender address (MAIL)
`to:`	envelope recipient address (RCPT)
`cltaddr:`	client IPv4 address
`cltname:`	client host name
`cltresolve:`	result of forward and reverse client lookup
`mxbadip:`	IPv4 addresses that are not allowed for MX - A records
`certissuer:`	DN of CA cert that signed that presented cert
`certsubject:`	DN of presented cert

Valid addresses for `from:` and `to:` are RFC 2821 addresses without the angle backets (`localpart@domain`) as well as partial addresses in the form `localpart` and `@domain`, i.e., domains must be preceeded with an at (`@`) sign. Valid addresses for `cltaddr:` and `mxbadip:` are IPv4 addresses and (sub)nets, and for `cltname:` host names. The client host name is determined by performing a reverse lookup (PTR record) for its IP address. The resulting names are looked up as A records. Only if one of the A records matches the client IP address, the host name is set. The result of these lookups can be used for `cltresolve:` where the following keys are valid:

`ok`	reverse and forward lookup match
`no`	reverse and forward lookup do not match
`tempptr`	reverse lookup (PTR) caused a temporary error
`tempa`	forward lookup (A) caused a temporary error

Valid values for RHS are

`relay`	allow relaying; currently only for `to:`, `cltaddr:`, `certissuer:`, and `certsubject:`
`ok`	accept
`error:`*XYZ A.B.C.D text*	return an error consisting of SMTP reply code *XYZ*, enhanced status code *A.B.C.D*, and *text*, i.e., the part after `error:` is returned to the client.
`reject`	same as `error:550 5.7.0 Rejected.`
`discard`	accept command but silently discard its effects.

Some tags may allow for other RHS values, these are explained when those tags are discussed in more detail.

Optionally a RHS can be preceeded by the modifier `quick:`. For an `error:` entry it causes an immediate rejection when the entry matches. Otherwise rejections can be delayed to the `RCPT` stage – if SMTPS is configured appropriately, see Section 5.9 – and can be overridden using the modifier `quick:` together with `ok` or `relay` in the access map for the recipient address with the `to:` tag. Using the modifier `quick:` together with `relay` for an entry with the `cltaddr:` tag causes it to override all other access map checks. `quick:ok` for an entry with the `cltaddr:` tag causes it to override other access map checks unless they are necessary to allow relaying.

Domain names (`@domain`) must have an exact match, subdomain matching can be specified with a leading dot, i.e., `@.domain`, see Section 5.11.1.

Examples:

```
cltresolve:tempptr        error:451 4.7.1 reverse lookup failed
mxbadip:127.0.0.1         error:551 5.7.1 Bad IP address 127.0.0.1 in MX/A list
mxbadip:192.168.255.255   error:551 5.7.1 Bad IP address 192.168.255.255 in MX/A list
from:@spammer.domain      error:551 5.7.1 No spammers
from:@.spammer.domain     error:551 5.7.1 No spammers in subdomains either
to:root                   error:551 5.7.1 No mail to root
to:abuse                  quick:ok
cltaddr:10                error:551 5.7.1 No direct mail from 10.x.y.z
cltname:spammer.domain    quick:error:551 5.7.1 No mail from spammers
to:@primary.domain        relay
cltaddr:10                relay
cltaddr:127.0.0.1         quick:relay
```

Discard The effect of `discard` depends on the protocol stage in which it is returned. If it is returned for a session, e.g., when a client connects, all transactions in the session are discarded. If it is returned for `MAIL` only that transaction is discarded. If it is returned for `RCPT` only that recipient is discarded; however, if no valid recipients are left, the entire transaction is discarded. Moreover, if `quick:discard` is returned for one recipient the entire transaction is discarded too.

Mailertable The address resolver implements an asynchronous DNS resolver and it uses a file called `mt` (mailertable) which consists of domain parts of e-mail addresses and corresponding IP addresses (in square brackets) or domain/host names separated by one or more whitespace characters. The syntax for an entry in mailertable is:

entry	::=	lhs " "+ rhs
lhs	::=	["."] hostname | "."
rhs	::=	[[port "^"] [mailer ":"] hostlist
port	::=	integer
mailer	::=	"lmtp" | "esmtp"
hostlist	::=	host [" " hostlist]
host	::=	"[" IPv4-address "]" | hostname

An entry consists of a LHS and a RHS which are delimited by at least one space. The key (LHS) is a hostname or a dot (denoting the default entry), the value (RHS) consists of an optional port number, a optional mailer and a list of hosts which are separated by spaces. A host is either a hostname (which is subject to MX lookups) or an IPv4 address in square brackets.

Example:

```
localhost            lmtp:
SPAM.FILTER.DOMAIN   2525^esmtp:[127.0.0.1]
MY.DOMAIN            esmtp:[10.1.2.3]
ANOTHER.DOMAIN       esmtp:MTA.SERVER
.TLD                 esmtp:GATE.WAY
.                    esmtp:SMART.HOST
```

Note: currently this file must exist, even if there are no entries (it is created by `make install`).

Aliases To specify aliases for local addresses the Berkeley DB hash map `aliases.db` is used. The key in the map must be

- the local part of a valid (local) e-mail address,
- or a complete local e-mail address,
- or any e-mail address,

based on the `aliases` option (see Section 5.7, 8). The value (RHS) for an alias entry is a list of one or more RFC 2821 addresses (including the angle brackets) separated by spaces (*not* commas). If the RHS has only a single address which does not have an '@' sign, then it is converted into an RFC 2821 address by SMAR, i.e., SMAR will append the hostname of the machine and put angle brackets around the string. Example:

```
myalias:  localuser
mylist:  <user1@my.dom> <user2@my.dom> <localuser@local.host>
owner-mylist:  someuser
```

For mailing lists, the `owner-` notation is supported, i.e., if there are aliases *list* and *owner-list* then mail sent to *list* will use *owner-list* as envelope sender address; the original domain will be preserved.

Example for the option `aliases = localdomains`. Let two domains be local, i.e., in mailertable:

```
first.dom   lmtp:
second.dom  lmtp:
```

and these entries be in aliases:

```
myalias@first.dom:  user1
another@second.dom:  user2
```

Then mail to `<myalias@second.dom>` and `<another@first.dom>` would be rejected while mail to `<myalias@first.dom>` or `<another@second.dom>` would be accepted.

Aliases can be nested (currently up to 5 levels, see `smar/rcpts.c`).

5.9 Configuration for SMTP Server

1. `CDB_gid`: (numeric) group id for CDB, i.e., the group id of `smxq`, see Section 4.1.
2. `daemon_address`: address for daemon to listen on; this should not be used in normal operation. Current (preliminary) format is: `host:port`, `:port` (listen on 0.0.0.0) `host` (port defaults to 8000). Up to 16 addresses[11] can be specified. See the notes below.
3. `pass_fd_socket`: socket to pass file descriptor from MCP to SMTPS.
4. `flags`:
   (a) `8bitmime`: offer `8BITMIME`: sendmail X is 8 bit transparent, but it does not perform any conversion, so this option should only be used if all communication partners can deal with 8 bit data.
   (b) `background`: `fork(2)` after start; this should not be used in normal operation.
   (c) `delay_checks`: delay acceptance check until `RCPT` stage (unless explicitly overridden, see Section 5.8.3).

[11] Compile time option `SS_MAX_BIND_ADDRS`

(d) `lmtp_does_not_imply_relaying`: even if a domain in the mailertable has `lmtp:` as RHS do not implicitly allow relaying to it, i.e., do not consider the domain as "local" with respect to relaying. This is useful for an MSA to avoid external mail to local domains without authentication.

(e) `serialize_accept`: serialize `accept(2)` calls.

(f) `softbounce`: change permanent (5xy) SMTP error replies into temporary (4xy) errors. This is a useful feature for testing to avoid bounces due to misconfigurations.

(g) `access`: use access map (in SMAR). Note: currently this flag is required to perform a reverse lookup for a client IP address to get the hostname of the client which then can be used for logging and the `Received:` header.

5. `id`: unique identifier for SMTP server (0); see Section 5.9.1.
6. `io_timeout`: timeout for SMTP operations.
7. `max_threads`: maximum number of threads.
8. `max_wait_threads`: maximum number of waiting threads.
9. `min_wait_threads`: minimum number of waiting threads.
10. `max_bad_commands_per_session`: maximum number of bad, i.e., unknown, SMTP commands per session accepted by server [default: 3]. After this limit is reached the connection is terminated with an 421 error.
11. `max_recipients`: deprecated, use `max_recipients_per_session` instead.
12. `max_recipients_per_session`: maximum number of recipients per session.
13. `max_recipients_per_transaction`: maximum number of recipients per transaction.
14. `max_hops`: maximum number of hops (Received: headers) [default: 21[12]].
15. `max_message_size`: maximum message size (unit: KB) [default: 8MB[13]].
16. `processes`: number of processes to start.
17. `max_transactions`: maximum number of transactions per session.
18. `tls`: This is a subsection that specifies the parameters for `STARTTLS` support. It is only available if the SMTP server has been compiled with the option `SM_USE_TLS`, see Section 2.3. See appendix D for some background information about these options.

    (a) `cert_file`: File with certificate in PEM format.

    (b) `key_file`: File with private key for certificate in PEM format.

    (c) `CAcert_file`: File with CA certificate in PEM format.

    (d) `CAcert_path`: Directory with (symbolic links for) CA certificates in PEM format.

    (e) `flags`: Some flags are available to influence the behavior of the SMTP server with respect to STARTTLS.

        i. `allow_relaying_if_verified`: If the client presented a certificate that can be verified by the CA certificates that are available to the server (see above: `CAcert_file` and `CAcert_path`), then relaying is allowed for the SMTP session.

[12] Compile time option `SM_MAXHOPS`
[13] Compile time option `SM_MAX_MSG_SZ_KB`

ii. `check_access_map_for_relaying`: If this flag is set then the access map (which must be activated, see 4g) is checked to see whether relaying should be allowed for a client which presented a certificate that has been verified (see above). For this purpose, the DN of the cert issuer is looked up in the access map using the tag `certissuer:`. If the resulting value is `relay`, relaying is allowed. If it is `cont`, the DN of the cert subject is looked up next in the access map using the tag `certsubject:`. If the value is `relay`, relaying is allowed; every other value is currently ignored.
To avoid problems with the DN names in map lookups, they are modified as follows: each non-printable character and the characters '<', '>', '(', ')', '"', '+', ' ' are replaced by their hexadecimal ASCII value with a leading '+'. For example:
`/C=US/ST=California/O=endmail.org/OU=private/CN=`
`Darth Mail (Cert)/emailAddress=darth+cert@endmail.org`
is encoded as:
`/C=US/ST=California/O=endmail.org/OU=private/CN=`
`Darth+20Mail+20+28Cert+29/emailAddress=darth+2Bcert@endmail.org`
Examples:
To allow relaying for everyone who can present a cert signed by
`/C=US/ST=California/O=endmail.org/OU=private/CN=`
`Darth+20Mail+20+28Cert+29/emailAddress=darth+2Bcert@endmail.org`
simply use:

```
 certissuer:/C=US/ST=California/O=endmail.org/OU=private/CN=
 Darth+20Mail+20+28Cert+29/emailAddress=darth+2Bcert@endmail.org   relay
```

To allow relaying only for a subset of machines that have a cert signed by
`/C=US/ST=California/O=endmail.org/OU=private/CN=`
`Darth+20Mail+20+28Cert+29/emailAddress=darth+2Bcert@endmail.org`
use:

```
 certissuer:/C=US/ST=California/O=endmail.org/OU=private/CN=
 Darth+20Mail+20+28Cert+29/emailAddress=darth+2Bcert@endmail.org   cont
 CertSubject:/C=US/ST=California/O=endmail.org/OU=private/CN=
 DeathStar/emailAddress=deathstar@endmail.org                      relay
```

Notes:

- line breaks have been inserted after `CN=` for readability, each tagged entry must be one (long) line in the access map.
- if OpenSSL 0.9.6 is used then the `emailAddress=` part of a DN is replaced by `Email=`.

19. `auth`: This is a subsection that specifies the parameters for `AUTH` support. It is only available if the SMTP server has been compiled with the option `SM_USE_SASL`, see Section 2.3.

    (a) `flags`: flags for SMTP AUTH
    See the Cyrus SASL documentation for the meaning of these flags: `noplaintext`, `noactive`, `nodictionary`, `forward_secrecy`, `noanonymous`, `pass_credentials`, `mutual_auth`.

    (b) `trusted_mechs`: list of SASL mechanisms for which relaying is allowed if a client successfully authenticated using one of those

20. `pmilter`: This is a subsection that specifies the parameters for pmilter support (see Section 8). It is only available if it has been enabled during `configure` (`--enable-pmilter`, see Section 2.2).

    (a) `socket`: path of Unix domain socket to communicate with policy milter.

    (b) `timeout`: maximum amount of time to wait for a reply from a policy milter.

Notes: only one of `daemon_address` and `pass_fd_socket` should be specified. In normal operation it is almost always `pass_fd_socket` because the SMTP server cannot bind to privileged ports, hence the file descriptor must be passed from MCP.

5.9.1 Multiple SMTP Servers with different Configurations

The normal way to run multiple SMTP servers is to let MCP start several SMTP servers. Each SMTP server must given a unique identifier (see Section 5.9, item 5) and each SMTP server section in `smx.conf` must have a unique name (e.g., MTA and MSA), which is passed via the option `-N` *name* to `smtps`. Example: `smx.conf`:

```
smtps MTA {
  port = 25;
  type = pass; pass_fd_socket = smtps/mtafd;
  user = sm9s;
  path = /usr/libexec/smtps;
  arguments = "smtps -N MTA -f /etc/smx/smx.conf";
  log { facility = mail; ident=smX-MTA; }
}

smtps MSA {
  port = 587;
  type = pass; pass_fd_socket = smtps/msafd;
  user = sm9s;
  path = /usr/libexec/smtps;
  arguments = "smtps -N MSA -f /etc/smx/smx.conf";
  log { facility = mail; ident=smX-MSA; }
  trusted_auth_mechs = { CRAM-MD5, DIGEST-MD5 };
  auth_flags = { noplaintext }; }
```

For tests it is also possible to let MCP start only one SMTP server which creates several copies of itself if multiple daemon addresses are specified (see Section 5.9, item 2). Note: this only works for unprivileged ports because the SMTP server does not run as root.

5.10 Configuration for SMTP Client

1. `debug_level`: debug level (only if compiled with SMTPC_DEBUG).
2. `io_timeout`: timeout for SMTP operations (unit: s).
3. `LMTP_socket`: Unix domain socket to use for LMTP [default: `lmtpsock`].
4. `log_level`: logging level.
5. `max_wait_threads`: maximum number of waiting threads.
6. `min_wait_threads`: minimum number of waiting threads.
7. `remote_port`: port to which connections should be made [default: 25]. Note: if multiple SMTP clients are specified, all of them *must* use the same value for `remote_port`. Currently the scheduler requires that all SMTP clients behave the same. If different ports are required, then those must be listed in mailertable entries.

8. `processes`: number of processes to start.
9. `wait_for_server`: maximum amount of time to wait for a server (QMGR) to become available (unit: s).
10. `tls`: This is a subsection that specifies the parameters for `STARTTLS` support. It is only available if the SMTP client has been compiled with the option `SM_USE_TLS`, see Section 2.3. See appendix D for some background information about these options.
    (a) `cert_file`: File with certificate in PEM format.
    (b) `key_file`: File with private key for certificate in PEM format.
    (c) `CAcert_file`: File with CA certificate in PEM format.
    (d) `CAcert_path`: Directory with (symbolic links for) CA certificates in PEM format.

5.11 Lookup Orders

5.11.1 Lookup Orders in Maps

In many cases an item is not just looked up verbatim in a map, but it may be split into logical parts and then less significant parts are iteratively removed and the remaining data is looked up until either a match is found or the data is empty; in the latter case a default key may be looked up depending on the map.

For domain names of the form "`sub2.sub1.tld`" the lookup order is "`sub2.sub1.tld`", "`.sub1.tld`", "`.tld`", and "`.`" (without the quotes), the last lookup is only done if the map type requests it, e.g., mailertable. Obviously this schema is extended if more components are specified. As the sequence shows there is no implicit "match all subdomains" lookup, instead entries in a map must have a leading dot for subdomains matches. To reiterate: "`sub2.sub1.tld`" does neither match the entry "`sub1.tld`" nor "`tld`".

For IPv4 addresses of the form "`A.B.C.D`", the lookup order is "`A.B.C.D`", "`A.B.C`", "`A.B`", and "`A`" (without the quotes). In contrast to domain lookups, no trailing dots are required (nor checked) to denote subnet matches, because the number of components of an IPv4 address is fixed (and known) in contrast to the number of components in a host name or domain name.

For RFC 2821 addresses of the form "<`user+detail@hostname`>" (where "`+detail`" is optional and "`+`" is the `address_delimiter`, see Section 5.8.2) the lookups are done according to the following sequence:

1. Repeat the following lookups for each subdomain of hostname (as explained above):
   (a) "`user+detail@hostname`" if "`+detail`" exists; this is a verbatim match.
   (b) "`user++@hostname`" if "`+detail`" exists and "`detail`" is not empty; this matches any non-empty "`+detail`". Note: the second "`+`" character is a fixed metacharacter, it does not depend on `address_delimiter`; it is a modelled after the "`+`" operator in regular expressions etc to denote a non-empty sequence of items.
   (c) "`user+*@hostname`" if "`+detail`" exists; this matches any "`+detail`" (including just "`+`").
   (d) "`user*@hostname`"; this matches "`user@hostname`" as well as "`user+detail@hostname`" ("`detail`" can be empty). *Note*: "`*`" is not a generic metacharacter here, it matches only a token beginning with `address_delimiter` or an empty sequence, it does *not* match any other character sequence. For example: the input "`user1@hostname`" does not match the LHS "`user*@hostname`".

(e) "user@hostname"; this does *not* match if "+detail" exists.

(f) "@hostname".

2. If nothing has been found and the map type requests it, then try localpart only (with the same meaning as above):

   (a) "user+detail" if detail exists

   (b) "user++" if detail exists and is not empty,

   (c) "user+*" if detail exists,

   (d) "user*",

   (e) "user"

5.11.2 Lookup Orders for Anti-Spam Measures

Map lookups for anti-spam measures are performed according to the (E)SMTP dialogue, i.e., connection information (cltaddr: and cltname:), MAIL command (from:), and RCPT command (to:). Whether a rejection has an immediate effective depends on the result of the lookup, e.g., the quick: modifier, and whether the option delay_checks is set.

5.11.3 Macro Replacements in RHS

The alias map allows the use of macro in the right hand side of map entries. Macros have the form "${name}" (without the quotes). Available macros are: user, detail, domain, tag, delimiter, subdomain, extension. They have the obvious meaning; subdomain refers to the part of the domain before the dot, i.e., if the pattern is @.domain and the input is user@host.domain then subdomain refers to host, extension is the delimiter and the detail together (provided the address contains them).

Example:

```
alias*@.domain      user${extension}@${subdomain}.domain
```

provides the following mappings:

```
alias@host.domain              user@host.domain
alias+detail@host2.domain      user+detail@host2.domain
```

6 Running sendmail X

6.1 MCP

Start MCP as root in the directory /var/spool/smx (i.e., the main queue directory, see Section 4: SMXQDIR) using

```
# ./mcp.sh
```

The script contains the runtime path for MCP based on the data used by `configure`.

To stop the entire sendmail X MTS simply terminate the MCP, it will forward the signal to all processes it started.

The MCP provides some restart functionality: if a process crashes, it will restart it unless the exit code indicates that a restart is useless, e.g., `EX_USAGE`. Moreover, the processes listed in the restart dependencies will be stopped and started too.

6.2 Using sendmail X only for Outgoing Mail

sendmail X can be used in combination with a MUA that speaks (E)SMTP directly or with the sendmail 8 MSP for outgoing mail. For the latter add this to your sendmail 8 `submit.mc` file (see also `misc/sm8.submit.mc`):

```
LOCAL_RULE_0
R$* + X<@$*>    $#smx $@ localhost $: $1 <@$2>

LOCAL_RULESETS
SHdrToSMTP
R$+                     $: $>PseudoToReal $1            sender/recipient common
R$+                     $: $>MasqSMTP $1                qualify unqual'ed names
R$* + X<@$*>            $: $1 < @ $2 >
R$* < @ *LOCAL* > $*    $: $1 < @ $j . > $2

MAILER_DEFINITIONS
Msmx,   P=[IPC], F=kmDFMuXa, S=EnvFromSMTP/HdrFromSMTP, R=EnvToSMTP/HdrToSMTP,
        E=\r\n, L=990, T=DNS/RFC822/SMTP,
        A=TCP $h 2009
```

and run the SMTP server of sendmail X as listener on `localhost:2009`. Then mail to `<local+X@domain>` will be sent via sendmail X. After initial testing the `relay` mailer can be changed to use port 2009 by default hence the local additions shown above can be removed.

6.3 Using sendmail X for Incoming Mail

6.3.1 Specifying Local Domains

If the domain of a recipient address matches an entry in mailertable (see Section 5.8.3) with the right hand side `lmtp:`[14] then SMTPC talks LMTP over the local socket `lmtpsock` (see 5.10). If you have an LDA that runs as daemon and can talk LMTP over a local socket you can use it for local delivery. It is also possible to use `procmail` in LMTP mode and start it from `mcp`, see `smx.conf`. See `contrib/procmail.lmtp.p0` for a patch for procmail 3.22 to allow handling of addresses with details (`+extension`) in LMTP mode. A mailertable `mt` for local delivery via LMTP should look like this:

```
localhost     lmtp:
MY.DOM        lmtp:
HOST.MY.DOM   lmtp:
```

[14] currently internally encoded as `127.0.0.255`

By default mail to addresses whose domain part is listed in mailertable with RHS `lmtp:` is allowed, i.e., those domains are considered local and hence relaying (even though technically this might not be called relaying) to them is allowed. This behavior can be turned off (see Section 5.9, item 4d) in which case it is necessary to also allow relaying to these domains which can be done either via the access map (see Section 5.9, 4g), or the command line option `-T` for SMTPS. This allows for treating (some of) these domains as *private* by not allowing relaying to them, hence they will be only reachable from systems from which relaying is allowed.

6.3.2 Specifying Valid Local Addresses

To validate addresses for local domains, SMAR uses the Berkeley DB hash map `aliases.db`, which can be created using makemap (from sendmail 8) or `libsmmap/createmap` or a map specified by the option `local_user_map` (see Section 5.8.2, item 6). The key in the map must be the local part of a valid (local) e-mail address. If the local part cannot be found in either map, the address is rejected.

To list valid local addresses in the alias map the right hand side must be the string "`local:`", e.g.,

```
postmaster:   <user@host.domain>
abuse:        user+abuse
user++:       local:
user:         local:
```

6.4 Using sendmail X as Gateway

sendmail X can easily be used as an internet gateway. To override routing, mailertable entries (see Section 5.8.3) can be specified. A list of valid addresses can be made available via the access map by allowing relaying to those addresses instead of entire domains, e.g.,

```
to:user1@my.domain          relay
to:user2@my.domain          relay
to:postmaster@my.domain     relay
cltaddr:10.12               relay
```

6.5 Using sendmail X as Backup MX Server

The previous section showed how to specify valid remote addresses if all of them are known. However, for systems that act as backup MX servers it might not be simple to always keep such a list up to date. In that case, a default entry for a domain should be made, e.g.,

```
to:user1@other.domain       relay
to:user2@other.domain       relay
to:postmaster@other.domain  relay
to:@other.domain            error:451 4.3.3 Try main MX server
cltaddr:10.12               quick:relay
cltaddr:127.0.0.1           quick:relay
```

The last two entries allow local systems to send mail to any user at `other.domain`; without those entries mail to unlisted users will be (temporarily) rejected and hence cannot be delivered via this system.

6.5.1 Note about Backup MX Servers

It is not a good idea to run a backup MX server `B` for a host `A` that has stronger anti-spam measures; if mails are sent to `A` via `B`, then `B` may accept them for delivery, but `A` may reject them and hence `B` has to sent bounces, which, in case of spam, are most likely to forged addresses, hence those bounces will only cause additional problems. The opposite case (`B` has stronger anti-spam measures than `A`) can cause the rejection of mail that `A` actually wanted to receive. Hence `B` and `A` should have the same anti-spam measures; i.e., a system that acts as backup MX server for another one should perform the same anti-spam checks as the main MX server(s).

6.6 Miscellaneous Programs

6.6.1 Do not run programs as `root` User

Almost all sendmail X programs (except for MCP) refuse to run with `root` privileges. To run a program as a different user the utility `misc/runas` can be used, e.g., after installation in `/usr/local/bin/`

```
# /usr/local/bin/runas smxq mailq -V
```

(specify `-h` to see the usage).

6.6.2 Displaying Content of Mail Queues

The program `mailq` displays the content of the mail queues (`defedb` and `ibdb`). Currently its output is in a similar format as the sendmail 8 version. The option `-h` shows how to use the program; see the previous section about using `runas` for `mailq`. Note: the output of this program might not be accurate due to internal buffering by QMGR. Moreover, this program reads DEFEDB in such a way that only entries that have been *checkpointed*[15] are shown. This is done to avoid interference with the operation of QMGR.

6.6.3 Interacting with QMGR

The program `qmgrctl` allows to interact with the QMGR via the control socket (see Section 5.7, item 4). Invoke `qmgrctl -h` to see the available options. By default the program will show the current status of QMGR.

Enhancement to this program are welcome to provide more functionality.

6.7 Reloading Maps

Maps (for SMAR and QMGR) can be reloaded by moving the old db file out of the way, creating a new file and then sending a `USR1` signal to the appropriate process to reopen the map.

```
# mv $MAP.db $MAP.old.db
# createmap -F $MAP.db < $MAP
# kill -USR1 $PID
```

[15] See Section 5.7 about options for checkpoints.

Note: for QMGR it is also possible to use `qmgrctl -r` instead, see Section 6.6.3.

6.8 Logging

Logging is done via `syslog(3)` (see Section 5.5, 1) or to `stdout/stderr`, which is redirected by the default MCP configuration to `PROG.log`. The logging format is not yet completely consistent across programs. Moreover, the logging entries might not be easy to understand because they contain some details which are not interesting to a potential postmaster, but to developers. Nevertheless, the logging entries should show the flow of mail through the system. See Appendix C.2 for an explanation of the format of logfile entries.

Note: logfiles *must* exist with the proper owner and permissions to be used. Neither MCP nor the modules will currently create logfiles. This is done by `make install`, i.e., `misc/sm.setup.sh`, which parses `smx.conf` to extract the section titles/names and `user` entries to create the logfiles with the correct name and owner. This does not (yet) properly work if unique logfile names are created, see Section 5.2, 14.

6.8.1 Logfile Rotation

Unless `syslog(3)` is used (see Section 5.5, 1), logfile rotation can be achieved by copying the existing logfile to a backup file, e.g.,

```
# cp qmgr.log qmgr.log.0
```

and sending a `USR2` signal which will cause the processes to rewind the logfile. Note: the author is aware that this is not an optimal solution, however, using `syslog(3)` will usually provide a better way.

6.9 Regular Checks

There are at least two things that should be done regularly:

1. Check the logfile for errors:

   ```
   $ egrep 'sev=(ALERT|CRIT|ERR|FAIL)|assert' $LOGFILE
   ```

2. Keep track of the size of the processes, e.g,

   ```
   $ date >> $SMXPROCS
   $ ps axuww | grep '^smx' | sort >> $SMXPROCS
   ```

   If one of the processes continuously grows then sendmail X chould be compiled with `-DSM_HEAP_CHECK` (see 2.3) and a heap dump should be taken regularly by sending the USR1 signal to the process. By comparing subsequent heap dumps it should be possible to locate a possible memory leak.

Please report problems that cannot be resolved locally, see Section 1.

6.10 Dealing with Errors

6.10.1 Resource Problems

Resource problems in certain parts of the code can lead to a stop of the involved program. In such a case it will be restarted automatically but if the resource problem has not been taken care of the MTS may stop again. In that case manual interaction is required. The simple solution to a resource problem is of course to add more resources (RAM/disk) or to free up some resources, e.g., stopping programs that do not need to run or deleting unused files. There are also ways to control resource usage within sendmail X:

- memory shortage: the memory usage of sendmail X can be controlled by restricting the size of various data structures, see Section 5.7, e.g., `AQ_max_entries`, `IQDB`, and `OCC_max_entries`. However, setting these values too low will result in a very slow MTS that may operate in a degraded state which is not acceptable.
- disk shortage: sendmail X has options that let it decide how much free disk space is needed for operation, see Section 5.7: `ok_disk_space` and `min_disk_space`. However, if there is not enough space to store the envelope databases (DEFEDB and IBDB) then the system cannot work, hence sufficiently free disk space is essential for proper operation.

6.10.2 Database Problems

See Section 10 for some background information about the usage of the various databases before trying to fix any possible problems.

If the deferred database is corrupted then the Berkeley DB utilities to deal with such situations should be tried [Sleb], e.g., `db_recover`.

Currently messages stored in CDB have the transaction identifier (`ss_ta`, see Section C.2) as filename. In the worst case, i.e., if IBDB or DEFEDB are destroyed, this allows to reconstruct the envelope data together with the logfile entries. See the script `misc/rcvrenvfromlog.sh` for an example, here is a description of its operation. First, check which messages are still in CDB: in the CDB directory (5.4: `CDB_base_directory`) issue:

```
# ls -1 [0-9A-F]/S*
```

Then search for each of those transaction ids (`$TAID`) in the logfile (`$LOG`):

```
$ egrep "ss_ta=$TAID, (mail|rcpt)=" $LOG | \
  sed -e 's;^.*\(mail=<.*>\), .*;\1;' -e 's;^.*\(rcpt=<.*>\), .*;\1;'
```

will show the sender (`mail=`) and the recipients (`rcpt=`). Based on this data it is possible to resend the messages.

Note: contributions in this area are welcome, e.g., better scripts that perform more checks and maybe allow for completely automatic recovery.

7 Caveats

The following problems exist in this version of sendmail X.0:

- The address resolver seems to leak memory if an old, outdated resolver library (BIND 4.9.x) is used, however the builtin memory debug module does not show any leak, hence either the process status output is misleading or the leak is in a system library. Help in tracking down this problem is appreciated.
- If the system runs out of memory then the MTS may not act gracefully in all cases, see Section 6.10.1.
- If a disk that is used for one of the queues becomes full, some errors may not be handled gracefully, see Section 6.10.1. To avoid this, sendmail X has some limits for the amount of available disk space that is required to accept mail (see Section 5.7).

8 Policy Milter

This version of sendmail X has experimental support for a policy milter. Experimental means that the API (which is documented in Appendix B) may change over time.

A policy milter is similar to a milter in sendmail 8. The most important difference is that a policy milter in sendmail X can (currently) not modify any part of an e-mail, it can "only" decide whether to accept or reject an e-mail.

9 Miscellaneous

9.1 Strict RFC 2821 Compliance

The SMTP server currently enforces fairly strict RFC 2821 compliance. For example, a `MAIL` command must be given in the following format

```
MAIL From:<user@some.domain>
```

i.e., the angle brackets are required, there must be no space after ":", etc. This has the useful side effect of catching some spam programs:

```
5.5.0 Syntax error., input=MAIL FROM: <blafwhoyqjywvu@asia.com>
```

Moreover, the server requires that lines end in CRLF (\r\n), it will not accept command input without the correct line ending, i.e., trying to do that will cause a read error.

9.2 Security Checks

There are currently no additional security checks when creating/accessing files or directories besides those provided by the operating system. This could be a problem if MCP is misconfigured because it runs as

`root`. Hence it will simply overwrite existing files if those are specified in the configuration file. The other modules run as non-privileged users, hence the OS provides sufficient access checks – unless the system is misconfigured and the sendmail X accounts are misused for other purposes too.

9.3 Restrictions

Besides the obviously missing functionality there are some other things that may restrict the use of sendmail X in certain environments. Here is an incomplete list:

- DNS lookups currently use only UDP by default, hence answers that exceed the UDP paket size will cause problems. However, such DNS pakets are really rare (because they cause operational problems in various places, e.g., some firewalls may block TCP for port 53). A possible workaround might be to force TCP (see Section 5.8.2, item 2a), the correct way is a change in the DNS library to retry with TCP, but this has not yet been implemented.
- Map lookups convert keys to lower case before checking an entry. In general this is not a problem unless local addresses rely on preserving the case of the local parts of addresses. That is, local addresses which require upper case characters do not work.

Everything that is not described in the documentation does either not exist in the current version of sendmail X, or is unlikely to work. However, there may be omissions in the documentation, please inform the author of such bugs.

9.4 Code Review, Enhancements, Patches

Source code inspection as well as patches and suggestions are very welcome.

Enhancements and extensions are very welcome too, especially to extend the basic functionality of the current sendmail X release.

9.5 Porting

Porting to currently unsupported platforms including non-Unix systems is encouraged. Note that the destination system must support statethreads [SGI01] and Berkeley DB 4.x. It might be necessary to port those first.

10 Data Flow in Sendmail X

This Section explains how Sendmail X stores information about messages that are transferred. It gives some background information which is useful for troubleshooting. Details about the operation of sendmail X can be found in [Aßmb].

Sendmail X uses two different databases on disk to store envelope information (sender and recipients): IBDB: incoming backup database, DEFEDB: deferred envelope database, and one database to store message contents: CDB: content database. See Section 4.1 about the location and layout of these

databases[16]. The queue manager additionally uses two internal envelope databases: IQDB (Incoming Queue DataBase) and AQ (Active Queue).

Incoming messages are accepted by the SMTP servers which store the content in the CDB (complete messages including headers in the format as received). The envelope information, i.e., sender (`MAIL`) and recipients (`RCPT`), is stored by the queue manager in IQDB and written to IBDB which is just a log of envelope data and what happened to it. That is, the files in IBDB are written sequentially and are continuously growing. If a file reaches its size limit (see Section 5.7: `IBDB`), then it is closed and a new file is opened. For a delivery, the envelope information must be transferred into AQ. For incoming mail this happens as soon as a transaction is accepted, in which case the data is moved from IQDB to AQ. A transaction is only accepted if the message is safely written to CDB and the envelope information has been committed to IBDB, i.e., all information is committed to persistent storage[17].

The scheduler in QMGR takes recipient envelopes from AQ and creates transactions which are given to the SMTP clients for delivery. An SMTP client takes the transaction information and tries to send a message whose content is read from CDB. After a successful delivery attempt a record is written to IBDB that logs this information. A cleanup task removes periodically old IBDB files which contain only data that is no longer referenced.

The deferred envelope database is only used if a message cannot be delivered during the first attempt. In that case the appropriate envelope data is added to DEFEDB and a record is written to IBDB stating that the data has been transferred to DEFEDB. Entries in DEFEDB contain a timestamp called next-time-to-try at which QMGR reads them from the database into AQ and the scheduler tries another delivery attempt. If that succeeds, the entries are removed from DEFEDB, otherwise they are either requeued with a new next-time-to-try (in case of a temporary error) or a DSN (bounce message) is generated (in case of a permanent error).

A Miscellaneous about the Source Code

A.1 Verifying the Source Code Distribution

The source code is distributed as (compressed) tar file and is accompanied by a PGP signature file which has the same name as the tar file plus the ending `.sig`. To verify the integrity of the source code you need to have a copy of PGP [Cor] or GPG [Gnu] and the sendmail PGP signing key [Con]. Then you can verify the integrity of the source code distribution by using:

```
$ gpg --verify smX-$VERSION.tar.gz.sig
```
or:
```
$ pgp smX-$VERSION.tar.gz.sig smX-$VERSION.tar.gz
```

See the documentation for the software that you use for further information.

A.2 Version Naming

Each sendmail X version has a name in the following format:

sendmail X.major.minor.[qualifier]qualifier-version.patchlevel

[16] the term *database* is used loosely here, only DEFEDB is a real database, the others are just ways to store some information and access them in some way.

[17] If non-persistent storage is used for these databases mail can of course be lost.

The major number changes between releases when new features are introduced (*major* changes, but see below about the development phases). The minor number changes when no new features are introduced, but bugfixes and (portability) enhancements are made. That is, no configuration changes are needed when going from one minor version to the next. The patchlevel number is used for intermediate patches between releases, e.g., if something is broken but it is not important enough for a new release because it is barely used or encountered.

There are several different qualifiers:

1. PreAlpha: This means the software is not feature complete and hence might be missing some functionality that is considered important by different users. Additionally, there is most likely no compatibility in data structures stored on disk between different pre-alpha versions, e.g., when upgrading from PreAlpha16 to PreAlpha17 the main queue format may have changed without checks in the software for this. Hence old queues must be drained before upgrading. Moreover, the protocols used for communication between sendmail modules may have changed without providing backward compatibility, therefore modules from different releases must *not* be used together. Such incompatibilities are usually stated in the list of changes `ChangeLog`.

   Do not run this on a production server unless you are aware of the possible consequences. The software is still under development and not fully functional. Moreover, it may not be sufficiently tested.
2. Alpha: In this state the software is ready for public testing but its features may still change.
3. Beta: Feature changes are unlikely, but still possible if required. Usually only bugfixes occur between beta versions.
4. Gamma: This is a release candidate. Usually only critical bugfixes occur between gamma versions. There might be no gamma versions at all if beta testing was considered succesful and sufficient.
5. A release version does not have an explicit qualifier.

The qualifier-version is used to distinguish between different version of the same qualifier, e.g., PreAlpha16 and PreAlpha17. It is 0 for a release version.

Examples for version names: sendmail X.0.0.PreAlpha19.0, sendmail X.0.0.0.0 (this is the name of the first release).

See the file `include/sm/version.h` how the version string is converted into a 32 bit number that denotes the version number.

A.2.1 Snapshots

From time to time snapshots may be made available. Those are marked with a date in the distribution file name, e.g., `smX-0.0.16.0-20040928.tar.gz`. The name indicates that it is a snapshot of what will become version `smX-0.0.16.0`, i.e., the next release will have the given version number (without the date). The only other indication in the distribution is the inclusion of an `s` in the version number that is shown in the version output of the main components. A snapshot did not go through the usual release cycle and is made available as *technology preview*.

B Native Policy Milter API

Note: this API is still experimental and may change.

Naming conventions: A *policy milter* (also called *pmilter* is a program that uses the API provided by *libpmilter*. The latter interacts with the SMTP servers via an internal protocol, i.e., this protocol can be changed without changing the visible API and should not directly be accessed by a user application.

B.1 Data Structures

libpmilter itself uses three context structures all of which must be treated by a milter as opaque.

1. `pmg_ctx`: "global" libpmilter context (only one per process).
2. `pmss_ctx`: libpmilter context per SMTP server that connects to this instance.
3. `pmse_ctx`: libpmilter context per SMTP session.

Any of the libpmilter functions takes one of these contexts as parameter; e.g., all SMTP session oriented functions have a parameter of type `pmse_ctx_P`.

A milter can have its own contexts for each of these three environments.

B.2 Start and Stop

The functions in this section return `SM_SUCCESS` (0) on success and a negative value in case of an error.

First libpmilter must be initialized; a pmilter must specify a variable `pmg_ctx_P pmg_ctx;` which is passed per reference to the initialization function:

```
sm_ret_T sm_pmfi_init(pmg_ctx_P *pmg_ctx)
```

The pmilter global context must be treated as opaque data structure, it is passed to subsequent libpmilter function calls.

Next it is started, the milter passes a description of its requirements and functionality:

```
sm_ret_T sm_pmfi_start(pmg_ctx_P pmg_ctx, pmilter_P pmilter)
```

A milter can stop by calling:

```
sm_ret_T sm_pmfi_stop(pmg_ctx_P pmg_ctx)
```

There are various functions to set some options. To set the path of the Unix domain socket over which the SMTP servers and libpmilter communicate:

```
sm_ret_T sm_pmfi_setconn(pmg_ctx_P pmg_ctx, const char *path)
```

The backlog parameter of the `listen(2)` function can be set:

```
sm_ret_T sm_pmfi_setbacklog(pmg_ctx_P pmg_ctx, int backlog)
```

The debug level of libpmilter might be set via (this requires knowledge of the internals of the library which can be acquired by looking at the source code):

```
sm_ret_T sm_pmfi_setdbg(pmg_ctx_P pmg_ctx, int debuglevel)
```

To set the communication timeout:

```
sm_ret_T sm_pmfi_settimeout(pmg_ctx_P pmg_ctx, int timeout)
```

B.3 New SMTP Server

Whenever an SMTP server connects to a milter an option negotiation is performed (similar to ESMTP itself). A pmilter can check whether server capabilities are acceptable and return the options that it wants:

```
sm_ret_T pmfi_negotiate(pmss_ctx_P pmss_ctx, uint32_t srv_cap, uint32_t srv_fct, uint32_t srv_feat,
uint32_t srv_misc, uint32_t *pm_cap, uint32_t *pm_fct, uint32_t *pm_feat, uint32_t *pm_misc)
```

Currently only the capabilities field is used: `srv_cap` is set by the SMTP server to a list (implemented as bit field) of phases of the ESMTP dialogue that can be passed to a pmilter. In turn the pmilter must set `*pm_cap` to includes those phases of the ESMTP dialogue that it wants to receive. For details, see `include/sm/pmilter.h`. For each of those phases a callback is invoked (see Section B.4) which must be set by the pmilter in its description structure `struct pmilter_S` (see `include/sm/pmfapi.h`).

B.4 SMTP Session and Transaction

The protocol steps from ESMTP are "replayed" to the policy milter which can decide to accept or reject them.

- New SMTP session:

  ```
  sfsistat_T pmfi_connect(pmse_ctx_P pmse_ctx, const char *hostname, sm_sock_addr_T *hostaddr)
  ```

  `hostname`: host name, as determined by a reverse lookup on the host IP address; `hostaddr`: host address, as determined by a getpeername call on the SMTP socket.

- SMTP `HELO`/`EHLO` command:

  ```
  sfsistat_T pmfi_helo(pmse_ctx_P pmse_ctx, const char *helohost)
  ```

  `helohost`: Value passed to HELO/EHLO command, which should be the domain name of the sending host.

- `MAIL` (envelope sender):

  ```
  sfsistat_T pmfi_mail(pmse_ctx_P pmse_ctx, const char *mail, char **argv)
  ```

  `mail`: envelope mail address; `argv`: null-terminated MAIL command arguments.

- `RCPT` (envelope recipient):

  ```
  sfsistat_T pmfi_rcpt(pmse_ctx_P pmse_ctx, const char *rcpt, char **argv)
  ```

  `rcpt`: envelope recipient address; `argv`: null-terminated RCPT command arguments.

- `DATA`:

  ```
  sfsistat_T pmfi_data(pmse_ctx_P pmse_ctx)
  ```

- unknown/not implemented SMTP command:

  ```
  sfsistat_T pmfi_unknown(pmse_ctx_P pmse_ctx, const char *cmd)
  ```

  `cmd`: SMTP command.

- For each body chunk:

  ```
  sm_ret_T pmfi_body(pmse_ctx_P pmse_ctx, unsigned char *bodyp, size_t bodylen)
  ```

  There may be multiple body chunks passed to the filter. End-of-lines are represented as received from SMTP (normally Carriage-Return/Line-Feed; CRLF). `bodyp`: pointer to body data; `bodylen`:

length of body data. Note: the last body chunk contains the final dot of the SMTP transmission, i.e., "CRLF.CRLF"

- End of message (final dot of message has been received):

  `sfsistat_T pmfi_eom(pmse_ctx_P pmse_ctx)`

- Message is aborted outside of the control of the filter, for example, if the SMTP client issues an RSET command.

  `sm_ret_T pmfi_abort(pmse_ctx_P pmse_ctx)`

  If `pmfi_abort` is called, `pmfi_eom` will not be called and vice versa.

- `QUIT` (end of an SMTP session):

  `sm_ret_T pmfi_close(pmse_ctx_P pmse_ctx)`

  This is called when an SMTP session ends.

B.5 Set and Get pmilter Contexts

As explained in Section B.1 a milter can have a "global" context `pmilter_g_ctx`, a context per SMTP server `pmilter_ss_ctx`, and a context per SMTP session `pmilter_se_ctx`. The following functions are provided to set and get these contexts.

Set the "global" context `pmilter_g_ctx`:

`sm_ret_T sm_pmfi_set_ctx_g(pmg_ctx_P pmg_ctx, void *pmilter_g_ctx)`.

This must be done after libpmilter has been initialized but before control is transferred to it.

To retrieve the "global" context invoke:

`void *sm_pmfi_get_ctx_g(pmg_ctx_P pmg_ctx)`

Note: this requires the "global" libpmilter context which is not usually passed to pmilter functions in callbacks. See below how to access the "global" context `pmilter_g_ctx` from other places.

To set the pmilter context per SMTP server `pmilter_ss_ctx` use:

`sm_ret_T sm_pmfi_set_ctx_ss(pmss_ctx_P pmss_ctx, void *pmilter_ss_ctx);`

to retrieve it call:

`void *sm_pmfi_get_ctx_ss(pmss_ctx_P pmss_ctx)`

The "global" pmilter context `pmilter_g_ctx` can be retrieved from the libpmilter context per SMTP server:

`void *sm_pmfi_get_ctx_g_ss(pmss_ctx_P pmss_ctx)`

At the lowest level a context per SMTP session `pmilter_se_ctx` can be set via:

`sm_ret_T sm_pmfi_set_ctx_se(pmse_ctx_P pmse_ctx, void *pmilter_se_ctx)`

and retrieved by:

`void *sm_pmfi_get_ctx_se(pmse_ctx_P pmse_ctx)`.

Just as before there is a function to retrieve the pmilter context per SMTP server `pmilter_ss_ctx` from the libpmilter context per SMTP session:

`void *sm_pmfi_get_ctx_ss_se(pmse_ctx_P pmse_ctx)`

Note: if a pmilter uses these contexts, then it is useful that each "lower level" context contains a link to its "higher level" context. That is, each pmilter context per SMTP session `pmilter_se_ctx` should have a pointer to its pmilter context per SMTP server `pmilter_ss_ctx` which in turn should have a pointer to the "global" pmilter context `pmilter_g_ctx`. This allows access from a function that is specific to a SMTP session to each relevant context.

B.6 Accessing MTA Symbols

A pmilter can set a list of symbols it wants to receive from the MTA by calling

`sm_pmfi_setmaclist(pmss_ctx_P pmss_ctx, uint where, ...)`

during the option negotation, i.e., in `pmfi_negotiate()`. The parameter `where` denotes the stage of the ESMTP dialogue when the value of the symbol should be sent. It must be one of

PM_SMST_CONNECT	Session start
PM_SMST_EHLO	EHLO or HELO command
PM_SMST_MAIL	MAIL command
PM_SMST_RCPT	RCPT command
PM_SMST_DATA	DATA command
PM_SMST_DOT	Final dot of mail body

A sequence of up to `PM_MAX_MACROS` macros can be requested which must end with `PMM_END`. Valid values are:

PMM_SRVHOSTNAME	hostname of SMTP server
PMM_SEID	session id
PMM_MAIL_TAID	transaction id
PMM_DOT_MSGID	Message-Id

`PMM_MAIL_TAID` cannot be requested before `PM_SMST_MAIL` and `PMM_DOT_MSGID` can only be requested at stage `PM_SMST_DOT`.

To retrieve the value of a symbol the function

`sm_pmfi_getmac(pmse_ctx_P pmse_ctx, uint32_t macro, char **pvalue)`

can be used in the various callback functions of the ESMTP dialogue. If the macro was not in the request list, an error will be returned. If the macro has not yet been received, `*pvalue` will be `NULL`. Otherwise `*pvalue` will point to the value of the macro. Note: the string to which `*pvalue` points must *not* be changed.

B.7 Further Capabilities

In addition to selecting which SMTP commands to send to pmilter (see Section B.3), there is one other capability available:

`SM_SCAP_PM_RCPT_ST` causes the MTA to send `RCPT` information even if the command has been rejected, e.g., because the recipient is unknown, the recipient has been rejected due to access map checks, or relaying has been denied. Note: `RCPT` commands that are rejected for other reasons, e.g., because the address is syntactically invalid, or some limit (maximum number of recipients) is exceeded, will not be sent to pmilter.

The function

```
sm_ret_T sm_pmfi_getstatus(pmse_ctx_P pmse_ctx, sfsistat_T *pstatus)
```

should be used in that case to access the current SMTP reply code for the command. This functionality is useful for a pmilter that wants to keep track of all recipients, not just those which are accepted, e.g., to deal with dictionary attacks.

B.8 Miscellaneous Functions

To set a reply text in an SMTP session or transaction oriented callback in addition to the reply code use:

```
sm_ret_T sm_pmfi_setreply(pmse_ctx_P pmse_ctx, const char *reply)
```

Note: the reply string *must* contain the full SMTP reply, i.e., it must be of the form

```
XYZ D.S.N text\r\n
```

where `XYZ` is a valid SMTP reply code (see RFC 2821 [Kle01]) which *must* match the return code of the function from which `sm_pmfi_setreply()` is called, `D.S.N` is an enhanced status code as defined in RFC 3463 [Vau03] and the rest is an explanation of the status including CRLF (`\r\n`).

Return version number of libpmilter:

```
sm_ret_T sm_pmfi_version(pmg_ctx_P pmg_ctx, uint32_t *major, uint32_t *minor, uint32_t *patchlevel)
```

This can be used to compare the version number of the library against which pmilter is linked with the version number against which pmilter is compiled. The major version numbers must match otherwise the program will not run.

Signal handler function:

```
sm_ret_T pmfi_signal(pmg_ctx_P pmg_ctx, int sig)
```

This will be called when a USR1 or USR2 signal is received; it is not called within a signal handler, i.e., the code does not have to be signal-safe. Note: this is not yet implemented.

B.9 Return Values

SMTP Session and transaction oriented functions use `sfsistat_T` as return type. Allowed values for this type are (as defined in `include/sm/smreplycodes.h`):

- `SMTP_R_OK`: accept command.
- `SMTP_R_DISCARD`: discard effect of command.
- `SMTP_R_CONT`: continue other checks.
- `SMTP_R_SSD`: shut down SMTP session.
- `SMTP_R_TEMP`: reject command with a temporary error.
- `SMTP_R_SYNTAX`: syntax error.
- `SMTP_R_PERM`: reject command with a permanent error.
- other valid SMTP reply codes [Kle01].

Additionally return values can be modified by using `SMTP_R_SET_QUICK(returnvalue)`. See Section 5.8.3 for the effects of this.

C Format Specifications

C.1 Format of Session/Transaction Identifiers

The format of session and transaction identifiers is specified in `include/sm/mta.h`. For the SMTP server it consists of a leading 'S', a 64 bit counter and an 8 bit "process" identifier, both of which are printed in hexadecimal format. For the SMTP client it consists of a leading 'C', an 8 bit "process" identifier, a 32 bit counter, and a 32 bit thread index, all of which are printed in hexadecimal format.

Examples: `S00000000407CE49200`, `C010000137D00000000`.

SMTP server session/transaction identifiers are unique until the 64 bit counter wraps around, SMTP client session/transaction identifiers are unique only within a single invocation of QMGR.

C.2 Logfile Format

The general format of entries in a logfile is a sequence of named field which are separated by commas. Each field consists of a name, an equal sign, and a value. If the value is a text field that is received from an external (untrusted) source, then all non-printable characters, commas, and percent signs are shown as their two digit hexadecimal ASCII representation with a leading percent sign. For example, the text

```
550 5.7.1 no, not now, 99% usage
```

is encoded as

```
550 5.7.1 no%2C not now%2C 99%25 usage
```

This encoding allows a logfile analyser to use the comma symbol as a delimiter of fields without having to perform complicated parsing, e.g, the Unix `awk` utility can be used with comma as field separator. Note: suggestions for a better encoding or different solution for the problem are welcome (more details can be found in [Aßmb]).

Logfiles use the identifiers described earlier such that transactions and sessions can be easily recognized. For the following examples logfile entries have been slightly edited and line breaks have been inserted.

Here is one example of a session in an SMTP server:

```
ss_sess=S00000000407EAE3800, client_ipv4=127.0.0.1,
  client_name=localhost.endmail.org.
ss_sess=S00000000407EAE3800, where=connection, starttls=successful
ss_sess=S00000000407EAE3800, ss_ta=S00000000407EAE4E00,
  mail=<SENDER@sendmail.org>, stat=0
ss_sess=S00000000407EAE3800, ss_ta=S00000000407EAE4E00,
  rcpt=<RECIPIENT@sendmail.org>, idx=0, stat=0
ss_sess=S00000000407EAE3800, ss_ta=S00000000407EAE4E00,
  rcpt=<SOMEONE@SOME.DOMAIN>, idx=1, stat=0
```

```
ss_sess=S00000000407EAE3800, ss_ta=S00000000407EAE4E00,
  msgid=<20040916050457.GG54961@endmail.org>, size=1177, stat=0
```

The first entry shows a successful session creation including the IPv4 address and the hostname of the client as well as whether the client is allowed to relay. The second entry indicates that STARTTLS has been used. A new transaction is shown in the third entry and two recipients are given thereafter (along with the index `idx`). The last entry shows that the transaction was successful (`status=0`; 0 is used instead of 250 or other SMTP reply codes that indicate success) and the size of the received mail as well as its Message-Id.

Here is one example of a session in an SMTP client:

```
da_sess=C01000006C800000002, status=connected, port=25, addr=64.81.247.36
da_sess=C01000006C800000002, where=connection, starttls=successful
da_sess=C01000006C800000002, da_ta=C01000006C900000002,
  ss_ta=S00000000407EAE4E00, mail=<SENDER@sendmail.org>, stat=0,
  reply=250 2.5.0 MAIL command succeeded
da_sess=C01000006C800000002, da_ta=C01000006C900000002,
  ss_ta=S00000000407EAE4E00, rcpt=<RECIPIENT@sendmail.org>, stat=0,
  reply=250 2.1.5 RCPT ok
da_sess=C01000006C800000002, da_ta=C01000006C900000002,
  ss_ta=S00000000407EAE4E00, where=final_dot, size=1177, stat=0
```

This is very similar to the format of the entries entries in the SMTP server and should not require an explanation. In addition to the delivery agent session and transaction ids (`da_sess` and `da_ta`) the SMTP server transaction id (`ss_ta`) is logged too. This makes it simple to track a message through the MTS. Obviously `ss_ta` can be used for multiple outgoing messages if the incoming message has been sent to multiple recipients (maybe indirectly via an alias), hence this is not a unique identifier in the SMTP client log.

C.3 Format of Received Header

The format of the `Received:` header added by the SMTP server is specified in `smtps/smtps.c`.

```
Received: from EHLO-NAME (CLIENT-NAME [CLIENT-ADDR])
        by HOST-NAME (SM-X-VERSION) with PROTOCOL
        id SMTP-TA-ID; DATE
```

where `PROTOCOL` is `ESMTP`, `ESMTPS`, `ESMTPA`, `ESMTPSA`, or `SMTP` [New04]. If STARTTLS is active, then `(TLS=TLSVERSION, cipher=CIPHERSUITE, bits=CIPHERBITS, verify=VERIFYRESULT)` is placed before `id`, where TLSVERSION is the TLS protocol version, e.g., TLSv1, SSLv3, SSLv2; CIPHERSUITE is the cipher suite that was in use, e.g., AES256-SHA, EDH-DSS-DES-CBC3-SHA, EDH-RSA-DES-CBC-SHA, CIPHERBITS denotes the effective keylength (in bits) of the symmetric encryption algorithm of the TLS connection, and VERIFYRESULT is one of the following:

OK	verification succeeded.
NO	no cert presented.
NOT	no cert requested.
FAIL	cert presented but could not be verified, e.g., the signing CA cert is missing.

Note: the name of the client is only shown if the access map feature is activated (see Section 5.9, 4g), otherwise the time-consuming DNS lookups (PTR and A records) are not performed.

C.4 Format of DSNs

DSNs (*bounces*) are currently not compliant to RFC 1891ff. The format looks like this:

```
From: Mailer-Daemon@HOST.NAME
Subject: Undeliverable mail

Hi! This is the sendmail X MTA. I'm sorry to inform you that a mail
from you could not be delivered. See below for details.
```

and then a list of recipients and the reasons for the failure, e.g.,

```
Recipient:
<user@example.com>
Remote-MTA:
10.2.3.4
Reason:
550 5.7.1 <user@example.com>... Access denied
during RCPT
```

D Certificates for STARTTLS

When acting as a server, sendmail X requires X.509 certificates to support STARTTLS: one as certificate for the server, at least one root CA (`CAcert_file`), i.e., a certificate that is used to sign other certificates, and a path to a directory which contains certs of other CAs (`CAcert_path`). The file specified via `CAcert_file` can contain several certificates of CAs. The DNs of these certificates are sent to the client during the TLS handshake (as part of the CertificateRequest) as the list of acceptable CAs. However, do not list too many root CAs in that file, otherwise the TLS handshake may fail; e.g.,

```
error:14094417:SSL routines:SSL3_READ_BYTES:
sslv3 alert illegal parameter:s3_pkt.c:964:SSL alert number 47
```

You should probably put only the CA cert into that file that signed your own cert(s), or at least only those you trust. The directory specified via `CAcert_path` must contain the hashes of each CA certificate as filenames (or as links to them). Symbolic links can be generated with the following two (Bourne) shell commands:

```
C=FileName_of_CA_Certificate
ln -s $C `openssl x509 -noout -hash < $C`.0
```

An X.509 certificate is also required for authentication in client mode, however, sendmail X will always use STARTTLS when offered by a server. The client and server certificates can be identical. Certificates can be obtained from a certificate authority or created with the help of OpenSSL. The required format for certificates and private keys is PEM. To allow for automatic startup of sendmail X, private keys must

be stored unencrypted. The keys are only protected by the permissions of the file system, hence they should not be readable by anyone but the owner. If server and client share the same key it is ok to make the key group readable however. Never make a private key available to a third party.

E Debugging Options

There are several compile time parameters to turn on debugging. Doing so will enable the output of debug data (to stdout/stderr or in some cases to a logfile). Since currently no logging abstraction is in use, the output is done on a per-module basis (whatever is simplest for the individual module).

The compile time options are:

SC_DEBUG	SMTPC
SSQ_DEBUG	SMTPS - QMGR communication
SS_DATA_DEBUG	SMTPS DATA stage
QMGR_DEBUG	QMGR
SMAR_DEBUG	SMAR
SM_LIBDNS_DEBUG	libdns

For details see the source code.

Note: it is possible to set different debug levels for different debug categories in QMGR. For a list of categories see `include/sm/qmgrdbg.h`. To set a debug level n for a category c use the option `-x`$c.n$. The general syntax for the parameters is:

debugoptions	::=	debugoption ["," debugoptions]
debugoption	::=	range ["." level]
range	::=	first ["-" last]

If `level` is omitted, it defaults to `1`. Example: `-x1-3.4,5.3,9-11`

F Other Potential Problems with Test Programs

Some of the test programs may generate warnings, e.g., most of the tree related programs cause compilers on 32 bit systems to emit a warning `integer constant too large` which can be ignored.

References

[Aßma] Claus Aßmann. Sendmail X. `http://www.sendmail.org/%7Eca/email/sm-9-rfh.html`.

[Aßmb] Claus Aßmann. Sendmail X: Requirements, Architecture, Functional Specification, Implementation, and Performance. `http://www.sendmail.org/%7Eca/email/sm-X/`.

[Con] Sendmail Consortium. PGP keys. `ftp://ftp.sendmail.org/pub/sendmail/PGPKEYS`.

[Cor] PGP Corporation. PGP. `http://www.pgp.com/`.

[Gnu] GnuPG. GNU Privacy Guard. `http://www.gnupg.org/`.

[Kle01] Simple mail transfer protocol. RFC 2821, Internet Engineering Task Force, 2001.

[MV03] K. Moore and G. Vaudreuil. An Extensible Message Format for Delivery Status Notifications. RFC 3464, Internet Engineering Task Force, 2003.

[New04] Chris Newman. ESMTP and LMTP Transmission Types Registration. RFC 3848, Internet Engineering Task Force, 2004.

[Pro] OpenSSL Project. OpenSSL. `http://www.openssl.org/`.

[SAS] Cyrus SASL. Project Cyrus. `http://asg.web.cmu.edu/cyrus/`, `http://asg.web.cmu.edu/sasl/`.

[SGI01] SGI. State threads for internet applications, 2001. `http://state-threads.sourceforge.net/`.

[Slea] Sleepycat. Berkeley DB 4.4.XX Change Log. `http://www.sleepycat.com/update/4.4.XX/if.4.4.XX.html`.

[Sleb] Sleepycat. Berkeley DB Tutorial and Reference Guide, version 4.2.52. `http://www.sleepycat.com/docs/`.

[Vau03] G. Vaudreuil. Enhanced mail system status codes. RFC 3463, Internet Engineering Task Force, 2003.