NAME

scheduler − zmailer transport queue scheduler daemon

SYNOPSIS

scheduler

[−**divFHnQSVW**] [−**f** *configfile*] [−**E** *newentsmax[,newentstimemax]*] [−**L** *logfile*] [−**l** *statisticslog*] [−**N** *transpmaxfno*] [−**P** *postoffice*] [−**p** *channel/host-pair*] [−**R** *maxforkfreq*] [−**q** *rendezvous* [−**Z** *zenvfile*]

DESCRIPTION

The *scheduler* daemon manages the delivery processing of messages in the ZMailer.

The *router*(8zm) creates message control files in the *POSTOFFICE*/**transport** directory. These refer to the original message files in the *POSTOFFICE*/**queue** directory.

The *scheduler* reads each message control file from *POSTOFFICE*/**transport/**, translates the contained message and destination information into internal data structures, and unlinks the message control file.

Based on scheduling, priority, and execution information read from a configuration file, the *scheduler* arranges to execute *Transport Agents* relevant to the queued messages.

At the time scheduled for a particular transport agent invocation, the *scheduler* will start a transport agent (or use one from idle-pool), and tell it one by one which message control files to process. When all the destination addresses in a message have been processed, the *scheduler* performs error reporting tasks if any, and then deletes the message control file in *POSTOFFICE*/**transport** and the original message file in *POSTOFFICE*/**queue**.

All message delivery is actually performed by Transport Agents, which are declared in a configuration file for the *scheduler*. Each transport agent is executed with the same current directory as the *scheduler*. The *scheduler-transporter* interaction protocol is described later in this man-page.

The standard output of each transport agent are destination address delivery reports; either successful delivery, unsuccessful delivery, or deferral of the address. Each report uses byte offsets in the message control file to refer to the address. Reports may also include a comment line which will be displayed in the *scheduler*'s own reports.

Two types of reports are produced:

1. Error messages caused by unsuccessful delivery of a message are appended to its message control file. Occasionally, for example, when all addresses have been processed, the *scheduler* generates an error message to the error return address of the message (usually the original sender).

2. The *scheduler* binds itself to a well-known TCP/IP port (**MAILQ**, TCP port 174) on startup. Any connections to this port are processed synchronously in the scheduler at points in the execution where the state is internally consistent. The *scheduler* simply dumps its internal state in a terse format to the TCP stream. It is expected that the client program will reconstruct the data structures sufficiently to give a user a good idea of what the scheduler thinks the world looks like. The *mailq*(1zm) program serves this purpose.

OPTIONS

Invoking *scheduler* without any argument will start it as a daemon.

−d run as a daemon, usually used after −v to log daemon activity in great detail.

−E *newentsmax[,neweventtimelimit]*

when globbing new tasks from the directory, pick only first "newentsmax" of them, and leave rest for a new scan run, or do it at most for "newenttimelimit" seconds. Default

values are 40 000 messages, and 5 seconds.

This relates on how the system will scan the input queue areas for new messaes. Nominally in lightly loaded systems the scan for new messages will be done every 10 seconds, but if the queue in pre-scan area is sufficiently large, that interval is raised up to 20 seconds.

The "newenttimelimit" value can be set to 2 thru 15, and tells essentially for how long the scheduler is allowed to spend on the work to look for new things.

−f *configfile*
overrides the default configuration file *MAILSHARE*/**scheduler.cf**.

−F Freeze -- don't actually run anything, just do queue scanning. (For debug purposes..)

−H −HH
Use multi-level hashing at the spool directories. This will efficiently reduce the lengths of the scans at the directories to find some arbitary file in them. One 'H' means "single level hashing", two 'HH's mean "dual level hashing". "Hash" is directory which name is single upper case alphabet (A-Z).

When existing, ZENV variable *SCHEDULERDIRHASH* overrides the 'H' option.

−i run interactively, i.e., not as a daemon.

−L *logfile*
overrides the default log file location *LOGDIR*/**scheduler**.

−l *statisticslog*
starts the appending of delivery statistics information (ASCII form) into given file. No default value.

−M [1|2]
Version of the *mailq* protocol this server runs; essentially a test option, as existence of *PARAMauthfile= ".."* assignment at the *scheduler.conf* file turns the protocol into version 2.

−N *transmaxfno*
sets how many filehandles are allocated for the scheduler's started children (if the system has adjustable resources.)

−n

Toggles configuration flag 'default_full_content', which defines what will be DSN RET parameter assumed value in case the originator didn't supply that parameter.

Default behaviour is similar to RET=FULL, while usage of this option is equivalent of RET=HDRS. This option does not override originator supplied DSN RET parameter value.

−p *channel/host*
A debug-type option for running selectively some thread under a single instance of the *scheduler*.

Use this with options: −v

−P *postoffice*
specifies an alternate *POSTOFFICE* directory.

−q *rendezvous*
on machines without TCP/IP networking, the rendezvous between *scheduler* and *mailq*(1zm) is done using a well-known named pipe. This option overrides the default location for this special file, either *RENDEZVOUS* or **/usr/tmp/.mailq.text**.

−Q The "Q"-mode, don't output the old style data into the queue querier, only the new-style one.

−S Synchronous startup mode, scans all jobs at the directory before starting even the first transporter.

−v be verbose about activity, and do not detach as a daemon.

−W be used in conjunction with −v to delay verbose logging start until after all the files have been parsed in, and it is a time for doing scheduling.

−Z *zenvfile*
passes on explicite non-compiled-in-default located ZCONFIG environment file.

−V print version message and run interactively.

CONFIGURATION FILE

The *scheduler* configuration file consists of a set of clauses. Each clause is selected by the pattern it starts with. The patterns for the clauses are matched, in sequence, with the *channel/host* string for each recipient address. When a clause pattern matches an address, the parameters set in the clause will be applied to the *scheduler*'s processing of that address. If the clause specifies a command, the clause pattern matching sequence is terminated. This is a clause:

```
local/*
        interval=10s
        expiry=3h
        # want 20 channel slots in case of blockage on one
        maxchannel=20
        # want 20 thread-ring slots
        maxring=20
        command="mailbox -8"
```

A clause consists of:

- A selection pattern (in shell style) that is matched against the *channel/host* string for an address.
- 0 or more variable assignments or keywords (described below).

There are several possible **PARAM**-assignments starting at column 0, more of them below.

If the selection pattern does not contain a '/', it is assumed to be a channel pattern and the host pattern is assumed to be the wildcard '*'.

The components of a clause are separated by whitespace. The pattern introducing a clause must start in the first column of a line, and the variable assignments or keywords inside a clause must not start in the first column of a line. This means a clause may be written both compactly all on one line, or spread out with an assignment or keyword per line.

If the clause is empty (i.e., consists only of a pattern), then the contents of the next non-empty clause will be used.

The typical configuration file will contain the following clauses:

- a clause matching all addresses (using the pattern ***/***) that sets up default values.

- a clause matching the local delivery channel (usually **local**).
- a clause matching the deferred delivery channel (usually **hold**).
- a clause matching the error reporting channel (usually **error**).
- clauses specific to the other channels known by the *router*, for example, **smtp** and **uucp**.

The actual names of these channels are completely controlled by the *router* configuration file.

Empty lines, and lines whose first non-whitespace character is '#', are ignored.

Variable values may be unquoted words or values or doublequoted strings. Intervals (delta time) are specified using a concatenation of: numbers suffixed with 's', 'm', 'h', or 'd' modifiers designating the number as a second, minute, hour, or day value. For example: 1h5m20s.

The known variables and keywords, and their typical values and semantics are:

interval (1m)
: specifies the primary retry interval, which determines how frequently a transport agent should be scheduled for an address. The value is a delta time specification. This value, and the *retries* value mentioned below, are combined to determine the interval between each retry attempt.

idlemax (3x interval)
: When a transport agent runs out of jobs, they are moved to "idle pool", and it a TA spends more than *idlemax* time in there, it is terminated.

expiry (3d)
: specifies the maximum age of an address in the *scheduler* queue before a repeatedly deferred address is bounced with an expiration error. The actual report is produced when all addresses have been processed.

  The expiry happens only, if there has been at least one delivery attempt in the lifetime of the scheduler process currently running.

expiry2 (0)
: specifies additional time beyond primary expiration by which time the message will be expired even without any delivery attempts having been made.

retries (1 1 2 3 5 8 13 21 34)
: specifies the retry interval policy of the *scheduler* for an address. The value must be a sequence of positive integers, these being multiples of the primary interval before a retry is scheduled. The *scheduler* starts by going through the sequence as an address is repeatedly deferred. When the end of the sequence is reached, the *scheduler* will jump into the sequence at a random spot and continue towards the end. This allows various retry strategies to be specified easily:

  brute force (or "jackhammer"):
  : retries=0

  constant primary interval:
  : retries=1

  instant backoff:
  : retries="1 50 50 50 50 50 50 50 50 50 50 50 50 "

  slow increasing (fibonacci) sequence:
  : retries="1 1 2 3 5 8 13 21 34"

  s-curve sequence:
  : retries="1 1 2 3 5 10 20 25 28 29 30 "

  exponential sequence:
  : retries="1 2 4 8 16 32 64 128 256 "

  etc.

maxta (0)

This is **global** parameter limiting the number of transport-agent processes that the sceduler can have running concurrently.

If retrying an address would cause the number of simultaneously active transport agents to exceed the specified value, the retry is postponed.

When no *maxta* parameter is used, or '0' is used, the *scheduler* (at operating systems supporting it) uses getrlimit(2)/setrlimit(2) to maximize the number of available file descriptors for the process. The thus maximized value minus about 20 is then used as *maxta* value.

At some operating systems there are no bidirectional pipes or sockets for the scheduler to talk with the transport agents, thus at those systems each TA process uses **two** *pipe*(2) file-handles, and thus system-wide limit on *maxta* halves from what was maximum derived above.

maxchannel (0)

This limits the number of concurrent transport agent processes for which the *channel* part of the address quad are same.

The default value is copied from the *maxta*.

It is advisable, although not mandatory, to have the same value for all *maxchannel* parameters at all clauses with matching channel parts.

Case where non-alike values might make sense is when you want to guarantee resources for a subset of destinations, e.g. for all other "smtp/" clauses have *maxchannel* of say 100, but for your very important domain domain have there a bit higher, say: 120.

maxring (0)

The recipients are groupped into "threads" by recipient channel+host parts, and threads matching same selector clause are groupped into "thread-rings", where same transport agent can be switched over from one recipient to another.

This parameter defines how many transport agents can be running at any time at the ring.

When no limit is given, then *maxta* value is used for default.

maxthr (1)

This limits the number of parallel transport agents within each thread; that is, using higher value than default "1" will allow running more than one TA for the jobs at the thread.

Do note that running more than one TA in parallel may also require lowering OVERFEED value. (E.g. having a queue of 30 messages will not benefit from more TAs, unless they all get something to process. Having OVERFEED per default at 150 will essentially feed whole queue to one TA, others are not getting any.)

overfeed (150)

Because the scheduler spins around a bit sluggishly to spot active TAs, it does make sense to feed more than one task to a TA, and then wait for the results, thus came about the "overfeed" mechanism.

This tells how many job specifiers to feed to the TA when the TA process state is "STUFFING."

skew (5)

Leftover from earlier scheduler internal structure. Does not make sense anymore.

user (root)
: is the user id of a transport agent processing the address. The value is either numeric (a uid) or an account name.

group (daemon)
: is the group id of a transport agent processing the address. The value is either numeric (a gid) or a group name.

command (smtp -srl ${LOGDIR}/smtp $host)
: is the command line used to start a transport agent to process the address.

  The program pathname is specified relative to the *MAILBIN*/**ta** directory.

  The string "$channel" is replaced by the current matched channel, and "$host" is replaced by the current matched host, from the destination address, and "${LOGDIR}" substitutes ZENV variable LOGDIR value there.

  It is strongly recommended that the "£host" is not to be used on a command definition, as it limits the usability of idled transporter.

  It is possible to place environment-string setting statements into the beginning of the line:

  command="MALLOC_DEBUG_=1 OTHER=var cmdname cmdparams"

queueonly
: a clause with *queueonly* flag does not auto-start at the arrival of a message, instead it **must** be started by means of *smtpserver*(8zm) command **ETRN** thru an SMTP connection.

  To have message expiration working, following additional entries are suggested: *interval=1h retries="24"*

For example, this is a complete configuration file:

```
# Default values
*/*
        interval=1m expiry=3d retries="1 1 2 3 5 8 13 21 34"
        maxring=0 maxta=0 skew=5 user=root group=daemon
# Boilerplate parameters for local delivery and service channels
local/*
        interval=10s expiry=3h maxchannel=2 command=mailbox
error
        interval=5m maxchannel=10 command=errormail
hold/*
        interval=5m maxchannel=1 command=hold
# Miscellaneous channels supported by router configuration
smtp/*.toronto.edu
smtp/*.utoronto.ca
        maxchannel=10 maxring=2
        command="smtp -srl $LOGDIR/smtp"
smtp
        maxchannel=10 maxring=5
        command="smtp -esrl $LOGDIR/smtp"
uucp/*
        maxchannel=5 command="sm -c $channel uucp"
```

The first clause (*/*) sets up default values for all addresses. There is no command specification, so clause matching will continue after address have picked up the parameters set here.

The third clause (error) has an implicit host wildcard of '*', so it would match the same as specifying error/* would have.

The fifth clause (smtp/*.toronto.edu) has no further components so it selects the components of the following non-empty clause (the sixth).

Both the fifth and sixth clauses are specific to address destinations within the TORONTO.EDU and UTORONTO.CA organization (the two are parallel domains). At most 10 deliveries to the **smtp** channel may be concurrently active, and at most 2 for all possible hosts within TORONTO.EDU. If $host is mentioned in the command specification, the transport agent will only be told about the message control files that indicate SMTP delivery to a particular host. The actual host is picked at random from the current choices, to avoid systematic errors leading to a deadlock of any queue.

CONFIGURATION FILE PARAM-ASSIGNMENTS

The scheduler can assign several of its internal parameters by having variable assignments beginning at column 0, and beginning with "PARAM" text. NOTE: There is no space in following assignment names!

PARAMmailqpath = "UNIX:/path/to/pf_unix/mailq/socket"

PARAMmailqpath = "TCP:mailq"

PARAMmailqpath = "TCP:174"

These define two different types of possible socket addresses for the mailq protocol; a UNIX socket, and a TCP socket. Default is "TCP:174".

PARAMauthfile = "/path/to/scheduler.auth"

Location of MAILQv2 autentication control file

PARAMglobal−report−interval = 15m

Interval by which all permanent reports accumulated into a message are reported by; sends out early reports of delivery failures, and does not force to wait for maximum queue timeout in case the message has more than once recipient.

PARAMmsgwriteasync = 1

If you desire to speed up the system a bit, and run slightly dangerously, enable (any non-zero numeric value will do) this parameter entry.

PARAMstore-error-on-error = 1

Enabling this parameter (any non-zero numeric value will do) will store the error messages into $POSTOFFICE/postman/ directory with suffix: ":error-on-error".

MESSAGE CONTROL FILE SYNTAX

A message control file contains all the information needed by delivery programs like *scheduler* and the transport agents. It is a terse presentation of the *router*'s decisions, along with some useful reference information.

The message control file consists of a number of fields.

All fields start in the first column (i.e., at the beginning of the file or just after a newline), and most fields extend to the end of line. The one exception is the message header field which extends till a double-newline terminator.

For all but this message header field, the second column is reserved for a tag byte. This position is used to lock the field and to indicate the status of past processing of the field. For example, the success or failure of delivery to a recipient address is indicated by a '+' or means the field has not been processed, or that processing has been deferred. A '~' indicates the field is locked because some transport agent is currently processing delivery for the address. The known field names and tags are defined in <**mail.h**>.

For all the *recipient* addresses, there is 6 characters space for transport-agent process-id so that a quickly restarted scheduler will not do double-delivery on some slowly running transporter.

The following fields are mandatory:

@ 0xHHHHHH
: Carries hex-encoded bitflags of what kind of format this dataset really is in.

  This is to ensure that "featurefullness" relation of: router <= scheduler <= TA-programs is not violated, and that the messages in the spool meet this criteria too.

  Rolling binaries back too far might break things.

i 123456−789
: the name of the message file in the *POSTOFFICE*/**queue** directory and of the message control file in the *POSTOFFICE*/**transport** directory. the system and therefore are named by their inode number.

o NNNNNNN
: the byte offset of the message body in the original message.

The following fields will frequently exist:

e user@some.domain
: is the return address for error messages, in a form that can be put in a To: header line.

l <mumble@jumble>
: is a string identifying this message in log entries. Typically the message id of the message would be used.

The following fields will occasionally appear:

x <jumble@mumble>
: is the log identification string (usually a message id) of an obsoleted message. The scheduler will purge any such identified message after running sanity checks.

v ../public/v_filename
: is the name of a file that the delivery system can appended log information to. This would appear as the result of running **sendmail -v** or **Mail -v**.

Since all programs need to refer to the same file, on mail clusters it is recommended that this be a relative path naming a file within the *POSTOFFICE* directory hierarchy.

A message control file must contain at least one address "group". Each group consists of a sender address field, one or more recipient address fields, and a message header that goes along with these.

An address field is a string containing a space-separated 4-tuple (quad) as follows:

channel
: is the name of the delivery channel for this address. This must be a contiguous word.

host
: is the name of the next destination host for this address. This too must be a contiguous word.

user
: is the address to be handed to the destination host for further delivery. This string may contain space. It is distinguishable because the last component cannot contain spaces.

privilege
: is the numeric uid representing the privileges associated with this address.

The address group components are:

s <address quad>
carries a sender address field in address-quad form.

r PPPPPPDDDD <address quad>
is a recipient address field in address quad form, but also contains fields for transprort agent pid number (PPPPPP), and a four character space for delay reporting by the scheduler (DDDD).

N string
are the delivery-status-notification parameters for the previous recipient.

n string
is the delivery-status-notification environment id data for the previous recipient.

R string
The DSN RET-mode setting (HDRS/FULL) for the previous recipient.

X nnnnnn
is an XOR recipient address field. The first element is a tag (a class number) to identify collections of recipient addresses which are equivalent (and therefore mutually exclusive). This is followed by an address field as described above.

m carries the message header for this address group.

After one or more of these address groups, the error messages for addresses are appended to the message control file. This is done by the *scheduler* as it receives error reports from transport agents.

d <diag-string>
Is storage format for diagnostics for recipient addresses.

Structure of the diagnostic string is:
d *id*:*headeroffset*:*drptoffset*::*diagtime* \t *notarydata* \t *message*
d 172:347:226::964917927 ...

The *id* field tells the byte offset to tge "r" of the receiver definition line.

The *headeroffset* gives the byte offset of the first byte of the *headers* associated with the address group.

The *drptoffset* points to the possible "N" line.

The *diagtime* is just system time of the event.

The *notarydata* and *message* are explained further below, and are identical to the transport-agent to the scheduler communication protocol objects of the same name.

For example, this is a typical message control file (it is a snapshot taken while a transport agent was running):

```
i 15582
o 60
l <90Jun3.165355edt.15582@neat.cs.toronto.edu>
e Rayan Zachariassen <rayan>
s local - rayan 7
r~23456   local - rayan 7
m
Received: by neat.cs.toronto.edu id <15582>;
          Sun, 3 Jun 1990 16:53:55 -0400
From:   Rayan Zachariassen <rayan>
To:     rayan
```

```
Subject: a typical message control file
Message-Id: <90Jun3.165355edt.15582@neat.cs.toronto.edu>
Date:   Sun, 3 Jun 1990 16:53:54 -0400
```

TRANSPORT AGENT INTERFACE

The transport agent interface follows master-slave -model, where the TA informs the *scheduler* that it is ready for the work, and then the scheduler sends it one job description, and awaits for diagnostics. Once the job is finished, the TA notifies the scheduler that it is ready for a new job.

A short sample session looks like this:

S: *(start the transport agent)*
T: **#hungry**
S: *spoolid* \t *hostspec*
T: **diagnostics**
T: **#hungry**
(etc. active work)
T: **#hungry**
S: *#idle*
T: **#hungry**
(the scheduler moved the TA into IDLE pool)
S: *spoolid* \t *hostspec*
(the TA was reactivated from the IDLE pool, doing work)
T: **#hungry**
S: **EOF**
(the scheduler determined that the TA should be killed)
T: *(exits)*

("S" = Scheduler, "T" = Transport agent)

Normal diagnostic output is of the form:

id / *offset* \t *notarydata* \t *status* SPC *message*

Where:

id is the inode number of the message file,

offset is a byte offset within its control file where the address being reported on is kept,

notarydata
: is a Ctrl-A separated quintet/sextet carrying delivery-status-notification information for the recipient. *Status* is one of **ok**, **ok2**, **ok3**, **error**, **error2**, **deferred**, **deferall**, or **retryat**, and the

status The exit status is a code from <**sysexits.h**>.

message
: is descriptive text associated with the report. The text is terminated by a linefeed. Any other format (as might be produced by subprocesses) is passed to standard output for logging in the **scheduler** log. The **retryat** response will assume the first word of the text is a numeric parameter, either an incremental time in seconds if prefixed by +, or otherwise an absolute time in seconds since epoch.

The notarydata has Control-A separated sub-fields, five or six of them:

Final-Rcpt-Address DSN-Action ENH-Status Report-String WTT-Host [WTT-TAid]

Final-Rcpt-Address
: This is the final form of the recipient address used at the final delivery of which the diagnostic data is report about. Compare with ORCPT= data!

DSN-Action
: One of: "delivered", "failed", relayed", "delayed", "expanded". See RFC 1892.

ENH-Status
: Enhanced status code per RFC 2034.

Report-String
: A free-form text (one line, no e.g. CRs enbedded).

WTT-Host
: For SMTP systems to produce "Remote-MTA:" header contents.

WTT-TAid
: ZMailer specific extension to report the process name and PID number of the transport agent producing this report. Helps to hunt for the syslogged data relating to this diagnostics instance.

STATISTICS LOG FORMAT

The statistics log reports condenced performance oriented information in following format:

Old format:

timestamp	fileid	dt1	dt2	state	$channel/$host
812876190	90401-2	0	5	ok	usenet/-
812876228	90401-1	0	7	ok	usenet/-
812876244	90401-1	0	1	ok	local/gopher-admin
812876244	90401-1	0	5	ok	smtp/funet.fi
812876559	90401-1	0	21	ok	smtp/utu.fi

New format:

spoolid	dt1	dt2	state	$channel/$host
S258367AbTCLWiT	0	5	ok	usenet/-
S258367AbTCLWiT	0	7	ok	usenet/-
S258387AbTCLWiT	0	1	ok	local/gopher-admin
S258397AbTCLWiT	0	5	ok	smtp/funet.fi
S258467AbTCLWiT	0	21	ok	smtp/utu.fi

where the fields are:

spoolid
: Spoolid (encoded inode and spool file mtime), which can be used to correlate with syslogged data with these same ids. The spoolid uses nanosecond resolution version of mtime, if such is supported in the system, and will in all cases encode nanoseconds as last five characters of the spoolid. Systems which do not internally support nanosecond timers in files will often result with same spoolid value for two messages.

timestamp
: The original spoolfile ctime (modification time) stamp in decimal.

fileid Spoolfile name after the router has processed it

dt1 The time difference from spoolfile ctime to scheduler control file creation by the router

dt2 The time difference from scheduler file ctime to the delivery that is logged on

state What happened? Values: ok, error, expiry

$channel/$host
Where/how it was processed

MAILQ(v1) PROTOCOL

Upon accepting a TCP connection on the **MAILQ** port (TCP port 174), the *scheduler* dumps data to the TCP stream in the following format and immediately closes the connection:

The TCP stream syntax is:

version *id*\n
data in id-dependent format<*close*>

The first line (all bytes up to an ASCII LF character, octal 12) is used to identify the syntax of all bytes following the line terminator **LF**. The first 8 characters of the first line are **"version "** as a check that this is indeed a **MAILQ** port server that has been reached, the remaining bytes are the real data format identification. The data is interpreted according to that format until the terminating connection close.

Format identifiers should be registered with the author. Currently identified ones are: **"zmailer 1.0"**, and **"zmailer 2.0"**.

For the **"zmailer 1.0"** data format, the syntax of the data following the first **LF** is:

Vertices:\n
(<*key*>**:**\t><*msg-file*>\t><*n-addrs*>**;** <*off1*>(**,**<*offN*>)*\t>[#<*text*>]\n)*
(**Channels:**\n
(<*word*>**:**\t>(><*key*>)+\n)+
Hosts:\n
(<*word*>**:**\t>(><*key*>)+\n)+)?
End:\n
<mailq -Q thread and status report >

Where:

\n is an ASCII linefeed

\t is an ASCII tab

key is an unsigned decimal number

msg-file
is a contiguous string (it is the message file name relative to a known directory)

n-addrs
is an unsigned decimal number (number of addresses)

off1...offN
are unsigned decimal numbers (address byte offsets)

text
is a string not containing an ASCII linefeed (status message)

word
is a contiguous non-space string

For example, here is sample output from connecting to the **MAILQ** port:

```
version zmailer 1.0
Vertices:
311424:  3714  1; 116
```

```
311680:  6472  2; 151,331    #128.100.8.4: Null read! (will retry)
312192:  6347  1; 152        #128.89.0.93: connect: Connection timed out (will retry)
Channels:
smtp:    >311424>311680>312192
Hosts:
scg.toronto.edu:       >311424
mv04.ecf.toronto.edu:  >311680
relay1.cs.net:         >312192
```

This is sufficient information to be able to reconstruct the transport queues as seen by the scheduler process, and to find more information than what is shown here by actually looking up the message control and data files referred to.

MAILQv2 PROTOCOL

The MAILQv2 protocol is interactive autenticating protocol, unlike its predecessor (v1). The system begins with a greeting telling version, and then giving one line of challenge to be used in subsequent autentication command:

```
version zmailer 2.0\n
MAILQ-V2-CHALLENGE: 942665308.906504.3\n
```

Protocol commands are:

AUTH username hexauthenticator
: The "login" of the mailq session. The hexauthenticator is lowercase hexadecimal character printout of MD5 checksum ran over the catenate of the challenge string (without its ending newline character), and the user's password. This algorithm is essentially the same what APOP scheme uses.

PERLish example with above challenge:

```
 $authen = MD5hex("MAILQ-V2-CHALLENGE: 942665308.906504.3"."my-passwd")
```

SHOW SNMP
: Implements 'mailq −QQQ'.

SHOW QUEUE SHORT
: Implements 'mailq −QQ'.

SHOW QUEUE THREADS
: Implements 'mailq −Q'.

SHOW THREAD channel host
: Reports details usable to implement mailq-v1 like interface. The details are TAB separated fields in a line until an LF. Fields are:

1) filepath under $POSTOFFICE/transport/
2) line number within a group, 0 starts a group
3) error address in brackets
4) recipient line offset within the control file
5) message expiry time (time_t)
6) next wakeup time (time_t)
7) last feed time (time_t)
8) count of attempts at the delivery
9) "retry in NNN" or a pending on "channel"/"thread"
10) possible diagnostic message from previous delivery attempt

ETRN etrn_string
: Supports ETRN-cluster subsystem at smtpserver.

KILL MSG spoolid
: Unimplemented.

KILL THREAD channel host
: Unimplemented.

Responses are written out to same socket in POP-like manner:

```
AUTH .... \n
+OK  or  -LOGIN FAILED \n
SHOW SNMP\n
+OK until LF.LF\n
text\n
text\n
```

If the output text contains a dot at the beginning of the line, it is duplicated in SMTP (and POP) style.

Of various commands, the "SHOW" class implements multiple text-line outputs, others only "+OK" (or "-ERR...").

MAILQv2 AUTHENTICATION FILE

For autenticating MAILQv2 protocol users, system can use *PARAMauthfile="/path/to/file.auth"* PARAM-assignment to identify file containing the data, and with the file to authenticate and parametrize what user can do thru the MAILQv2 port.

```
#
# APOP-like authentication control file for the ZMailer scheduler.
#
# Fields are double-colon (':') separated, and are:
#   - Username
#   - PLAINTEXT PASSWORD (which must not have double-colon in it!)
#   - Enabled attributes (tokens, space separated)
#   - IP ACLs
#
# Same userid CAN appear multiple times, parsing will pick the first
# instance of it which has matching IP address set
#
# The default-account for 'mailq' is 'nobody' with password 'nobody'.
# Third field is at the moment a WORK IN PROGRESS!
#
# SECURITY NOTE:
#   OWNER:      root
#   PROTECTION: 0600
#
# Attribute tokens:
#       ALL     well, a wild-card enabling everything
#       SNMP    "SHOW SNMP"
#       QQ      "SHOW QUEUE SHORT"
#       TT      "SHOW QUEUE THREADS", "SHOW THREAD channel host"
#       ETRN    "ETRN etrn_string"
#       KILL    "KILL THREAD channel host", "KILL MSG spoolid"
```

```
#
#       SMTPIP  Smtp-server's  "Z-REPORT" ACL.
#
# -- "nobody" via loopback gets different treatment from
#    "nobody" from anywhere else.
#
nobody:nobody:SNMP QQ TT ETRN:  [127.0.0.0]/8 [ipv6.0::1]/128
nobody:nobody:SNMP ETRN:        [0.0.0.0]/0   [ipv6.0::0]/0
#watcher:zzzzz:SNMP QQ TT ETRN: [127.0.0.0]/8 [192.168.0.1]/32
#root:zzzzzzz:ALL:              [127.0.0.0]/8 [192.168.0.2]/32
```

Z–ENVIRONMENT VARIABLES

LOGDIR
: defines location of log files. Example: **LOGDIR=/var/log/mail**

MAILBIN
: Defines where executable transport-agent binaries exist under $ **MAILBIN**/ta/ directory.

MAILSHARE
: Location of scheduler configuration files

PATH
: What PATH environment variable to give to transport-agent subprograms.

POSTOFFICE
: defines directory where all POSTOFFICE functions are under.
Example: **POSTOFFICE=/var/spool/postoffice**

SCHEDULERNOTIFY
: defines an *AF_UNIX/DGRAM* type local notification socket into which the *router*(8zm) sends a message for the *scheduler*(8zm) that there is a new job available.

SCHEDULERDIRHASH
: Carries a numeric value of "1" or "2" (if defined at all), which will then override possible "-H" option at the *scheduler*(8zm). Existence of this ZENV-variable tells the *router*(8zm) to send messages directly to the scheduler's hash subdirectories, thus eliminating a few directory operations which the scheduler would otherwise do, and at the same time limiting the size of the directory files.

  On high-load systems, value of: *SCHEDULERDIRHASH=2* is highly recommended!

SYSLOGFLG
: Existence of "c" or "C" character in value string enables syslogging of some events as seen by the scheduler.

ZCONFIG
: Gives location of *zmailer.conf* file (and filename). Actually this one variable does **not** override compiled-in location of this configuration file.

TCP–WRAPPER AND MAILQ

Do note that this applies only when the mailq is using old version 1 mode!

If the ZMailer system is configured with *tcp-wrapper* code, then service-id `"mailq"` is looked for all those addresses that are allowed to do queries.

Usually files *hosts.allow*, and *hosts.deny* contain following kind of entries:

/ETC/*hosts.allow*

```
mailq : ALL@1.2.3.0
smtp-receive: ALL@ALL
```

/ETC/*hosts.deny*

```
ALL : ALL@ALL
```

(*Do note that smtpserver* (8zm) *has also tcp-wrapper support, which becomes active simultaneously with scheduler's tcp-wrapper code!*)

SIGNALS

SIGHUP:
close and reopen the stdout/stderr log file.

SIGTERM:
exit cleanly.

SIGQUIT:
exit cleanly, but at first order transporter childs to shut down, and collect their status reports.

SIGALRM:
check pending work.

SIGUSR1:
reread the *scheduler* configuration file.

SIGUSR2:
dump state information to the *rendezvous* file.

FILES

/opt/mail/zmailer.conf	(ZCONFIG)
/var/spool/postoffice/.pid.scheduler	(POSTOFFICE/.pid.scheduler)
/var/spool/postoffice/.scheduler.notify	(POSTOFFICE/.scheduler.notify)
/var/spool/postoffice/scheduler	(POSTOFFICE/scheduler)
/var/spool/postoffice/transport	(POSTOFFICE/transport)
/var/spool/postoffice/queue	(POSTOFFICE/queue)

SEE ALSO

mailq(1zm), *mailq-m*(5zm), *router*(8zm), *smtpserver*(8zm), *mailbox*(8zm), *hold*(8zm), *smtp*(8zm), *errormail*(8zm), *sm*(8zm), *expirer*(8zm), *reroute*(8zm), *manual-expirer*(8zm), *manual-rerouter*(8zm), *zmailer.conf* (5zm).

RFC 822/2822	The basic Internet email format specification
RFC 1123	Various 822 parameter clarifications

Several extended SMTP facilities are implemented:

RFC 1341/1521/2045	MIME specification (body, formats)
RFC 1342/1522/2047	"MIME-2" specification (headers)
RFC 1425/1651/1869	ESMTP EHLO framework
RFC 1428	Basic MIME conversion rules
RFC 1891/3461	ESMTP DSN
RFC 1892/3462	The Multipart/Report Content Type
RFC 1893/3463	Enhanced Mail System Status Codes
RFC 1894/3464	Extensible Message format for DSNs
RFC 1985	ESMTP ETRN
RFC 2034	ESMTP ENHANCEDSTATUSCODES

RFC 2852 ESMTP DELIVERBY (incomplete inplementation)

AUTHOR

This program authored and copyright by:
Rayan Zachariassen <no address>
A plenty of changes and several real bugfixes by:
Matti Aarnio <mea@nic.funet.fi>