NAME

router − zmailer message routing daemon

SYNOPSIS

router

[−**diksSVW**] [−**I smtpserver**] [−**L** *logfile*] [−**P** *postoffice*] [−**f** *configfile*] [−**n** *#routers*] [−**o** *zmshoptions*] [−**r** *routerdirloops*] [−**t** *traceflag*] [−**Z** *zenvfile*]

DESCRIPTION

The *router*(8zm) daemon makes all decisions affecting the processing of messages in **ZMailer**.

A mail message is submitted by placing it in a file in the *POSTOFFICE*/**router** directory. The *router*(8zm) scans this directory frequently for new files and will lock and process them as it finds them. The result is a message control file that gets linked into the *POSTOFFICE*/**scheduler** and *POSTOFFICE*/**transport** directories for use by the *scheduler*(8zm) in the next step of message processing. The original message file is then moved to the *POSTOFFICE*/**queue** directory.

The *router*(8zm)'s behaviour is controlled by a configuration file read at startup. It is really a *zmsh*(1zm) script that uses facilities provided builtin to the *router*(8zm).

OPTIONS

Invoking the *router*(8zm) without any arguments will do nothing (except make it reads its configuration file and promptly exit). The normal startup method is to run the *zmailer*(1zm) script, as in:

```
zmailer router
```

This will kill the possible previous incarnation of the *router*(8zm), and start a new as a daemon.

−d detach and run as a daemon.

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

−i run interactively, presenting a *zmsh*(1zm) session with the configuration file preloaded.

−I smtpserver
: Special operational mode while running under smtpserver. Implements interaction protocol in between router, and smtpserver.

−k kill the currently running router by sending it a **SIGTERM** signal.

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

−m *memtracefile*
: For debug purposes when compiled with XMEM option, not in general production setup.

−N don't syslog normal routing operation information, syslog only errors..

−n *#routers*
: starts the specified number of parallel router processes. The default is a single router process.

−o *zmshoptions*
: sets the option string passed on the the internal *zmsh*(1zm) invocation. The default is **-O**. Note that the leading '-' is mandatory. See *zmsh*(1zm) for the available options.

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

−r *routerdirloops*
: Process *routerdirloops* messages out of any alternate router directories, then check to see if any higher-priority jobs have been created. If undefined and alternate router directories are in use, the router will clear the entire directory before returning. See below and *zmailer*(3zm) about Z−Environment and *ROUTERDIRS*.

−S Can be used to turn off non-serious syslogging.

−s Turns *stability*-flag off and on. Without this flag, the search of new jobs will be done with (sometimes) timeconsuming care of organizing the job files into time order.

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

−t *traceflag*
sets trace options, one per -t switch, even before the configuration file is loaded. This is otherwise equivalent to the builtin **trace** command. The currently known options are: **assign**, **bind**, **compare**, **db**, **final**, **functions**, **matched**, **memory**, **on**, **regexp**, **resolv**, **rewrite**, **router**, and **sequencer**.

−V print version message and run interactively.

−W Warn about syntax errors in the RFC-822 headers when this option is on. Without this, bad headers are shown in their original form, not by rendering them thru any formatting RFC-822 engine. *This deviates from old behaviour of the ZMailer in a major way!*

To restart a *router*(8zm) daemon:

```
router -dk
```

To test an address, start up an interactive session:

```
router -i
```

or if the **ZMailer** *sendmail*(8zm) is installed:

```
sendmail -bt
```

Then just use the pre-defined functions.

CONFIGURATION FILE

The *router*(8zm) configuration file must provide some services so the message processing can proceed:

- There must be a way to translate an arbitrary syntactically valid address into a specification for how to deliver for that address.

  This is routing and must be implemented by a configuration file function with the name: **router**.

- There must be a way to make policy decisions about what to do with an address in the context of the particular message it came from. The most typical kind of policy decision is how to rewrite addresses in the message header for the immediate destination of a message envelope recipient address.

  Policy at this level is implemented by a configuration file function called **crossbar**.

- There must be a way to specify what should happen when message processing cannot continue due to a temporary resource unavailability (e.g., when the nameserver is acting up). There is a shell variable, **defer**, which may be set internally in the database lookup routines in the *router*(8zm) scripts.

  In case of a temporary resource failure, the variable will be set to a well-defined non-null string related to the failure. Temporary failures encountered during message header address rewriting may be dealt with by the **header_defer** function.

The interface specifications of these items exhaust the expectations the *router*(8zm) has of its configuration file. The *router*(8zm) contains useful builtin functions that will aid in implementing the required functionality. These functions are described in the section on *BUILTIN FUNCTIONS* below.

Optionally the *router*(8zm) may provide services to other programs. In particular, the *smtpserver*(8zm) program relies on the router to do address validation and verification when it is asked (and enabled) to provide this service during an SMTP conversation. The expected name for this function is **server**.

The following are the interface specifications for the functions mentioned above:

router *address attributes*

This function is applied to all message envelope addresses. The first argument to this function is a syntactically valid (understood by the parser) address. The second argument is a symbol whose value is a property list for the address. The property list consists of alternating keys and values, and is modified using the **lreplace** function.

The **router** function returns a list of address groups. Each address group is a list of mutually exclusive address tuples, i.e. delivering to one is equivalent to delivering to any other destination address in the group. An address tuple is a list of 4 elements (another name for this is a quad), which in order are:

- the delivery channel, designating a transport agent.
- the next host to be delivered to (an uninterpreted channel parameter).
- the address to present the next host.
- the symbol whose value is a property list for this address.

By convention, if either the channel or host is irrelevant, it is given as `"-"`.

For example, these are possible parameters and return value:

```
z# g0=(privilege 0 type recipient)
z# router rayan g0
(((local - rayan g0)))
```

This function also handles any alias expansion that may be necessary. The following example shows the expansion of a single address to multiple independent recipients:

```
z# router root g0
(((local - ken g0)) ((local - rayan g0)))
```

The **router** function is free to change the privilege of the address, or to add any other information to the property list for use by the **crossbar** function.

Note: in the current version of ZMailer, only the *router*(8zm) knows about mutually exclusive addresses. Therefore all quads must be the lone element of their address group.

If a resource deferral occurs during processing (e.g., the nameserver is busy or broken), the global variable **defer** will be set to a non-null string indicating the problem. This string is in a format interpreted by the *hold* transport agent when it sees a host name in a destination address.

crossbar *from to*

This function controls policy decisions that require context knowledge. It rewrites the next-address portion of an address quad to the proper form, and determines which style of message header rewriting is appropriate. The arguments are sender and recipient address quads, as returned by the **router** function. The return value is a list of three elements:

- the name of a function that will be used to rewrite all message header addresses
- rewritten sender address quad
- rewritten recipient address quad

The message header address rewriting function will be called with a single argument, the address in original form, and returns a string argument, the address rewritten.

header_defer *address*
This function rewrites a message header address that might not have been rewritten properly due to a resource deferral. In that case the **defer** variable will be set by the *router*(8zm) during execution of the message header address rewriting function specified by the **crossbar** function. To avoid repeatedly having to check for this at the end of such rewriting functions, this mechanism is provided as a convenience. Any resource deferral during the execution of this function is ignored. Typically **header_defer** would just quote the address and make it relative to the local host.

server *key ...*
This function is not used by the *router*(8zm) while processing mail. It is called from the *smtpserver*(8zm) when synchronous address validation is required. Since this function may need services provided by other parts of the normal configuration file, it is included by convention. The first argument is a keyword which describes the desired service, followed by parameters to that service. The known keywords are:

init which should be invoked before any other service.

to verifies its single parameter as a valid SMTP RCPT TO address.

from verifies its single parameter as a valid SMTP MAIL FROM address.

verify verifies its single parameter as a resolvable address.

expand
prints the alias expansion, if any, of the single address parameter.

SMTP error codes and text may be **echo**'ed directly, and multiline output is handled by a filter in the *smtpserver*.

BUILTIN FUNCTIONS

The following builtin functions are provided in the *router* that do not exist in *zmsh*:

Functions that take a list argument:

channel
returns the channel (1st) component of an address quad.

host returns the host (2nd) component of an address quad.

user returns the next-address (3rd) component of an address quad.

attributes
returns the property list symbol (4th) component of an address quad.

Normal functions that take string parameters and return strings:

daemon
starts the *router* running in daemon mode, scanning the *POSTOFFICE*/**router** directory every few seconds for message files to process. This function is invoked automatically by other code in the *router* program and has no other purpose. *router* has a bit more complex directory semantics, than can be seen from above. See *zmailer*(3zm) for details.

basename *pathname* [*suffix*]
prints the base filename of the pathname. If a suffix is given and matches the filename, the suffix too is stripped from the filename.

db { add|remove|flush|owner|print|toc } [*database* [*key* [*value*]]]
is the access function to the database facilities in the *router*. The keyword arguments are:

add add a *key,value* entry to the database, if possible.

remove
remove a *key* entry from the database, if possible.

flush remove all entries from the database, if possible.

owner print the account name of the owner of the database, if possible. This is usually determined by the files associated with the database.

print print all entries of the database, if possible.

toc print a table of defined relations and their associated information. This table has five (5) columns, in order: the name of the relation, its type and subtype, cache entries and maximum size, flags, and associated files. See the **relation** function for more information.

The keywords may be abbreviated to their smallest unique prefix (usually a single character).

erraddron [*file*]
specifies a filename to append all address parsing error messages to. If there is no argument given, the logging is stopped. This is primarily for curious postmasters or other collectors of address trivia.

filepriv *file* [*uid*]
prints the numeric user id of the least privileged account that can modify the specified file. This is determined by an approximation that pessimistically assumes that any file or directory writable by group or others is insecure, and optimistically assumes that it is enough to check a file and its parent directory instead of all the way to the filesystem root. The reason for the latter is that if grandparent directories are insecure, the system is likely to have just as bad potential problems as can be created by using mail to run processes with forged powers (besides, doing the full check would be quite expensive).

If a second argument is given it is the numeric user id to assume for the file. This means only the parent directory will be checked for non-writability and for having the same (or a 0) uid.

gensym
generates and prints a new symbol name in the sequence **g0** to **g***N* every time it is called. The sequence is reset and any symbol values destroyed after the *router* has processed a message. This function is used to generate new symbols, to hold attached address property lists, during alias expansion.

groupmembers *groupname*
prints the accounts that are listed as members of a group in the system groups file, one per line. Note that accounts with the same login group id in but that are not listed in the groups file will not appear in this list.

homedirectory *user*
prints the home directory of the specified account.

hostname [*name*]
with an argument this sets the *router*'s idea of the system hostname, else the name as retrieved from the system is printed. The *router* has no preconceived notion of what the hostname is, so Message-Id and Received headers will only be generated if a hostname has been set using this function.

lappend *varname £values..*
Looks up named variable, and gives up if does not find it.

The variable data content is considered to be a simple list, and the values are appended to it as a chain.

listaddresses [-e *error-address*] [-E *errors-to-address*] [-c *comment*]
: filters an RFC822 address list on standard input to produce one normal form (no non-address tokens) address per line on its output. This function can be used to parse the alias file or .forward files or similar. If an error-address is specified, any syntax errors at list parsing will cause a report to be mailed to the given address. If a comment is specified, it will be inserted in the error report. If an errors occurs while messages are being delivered, the 'errors-to-address' can be used to force error message destination elsewere than to the default 'sender' of the message.

listexpand [-c *comment*] [-e *erroraddress*] [-E *errors-to*] *£attribute £localpart £origaddr* < listfile
: implements the most common pipeline where *listaddresses* was used with more efficient memory consumption handling.

login2uid *login*
: Prints the uid associated with the specified account name, if any. A side-effect is to add the GECOS name field of the account to the **fullname** in-core database, to add the login name to uid mapping to the **pwnam** in-core database, and to add the uid to login name mapping to the **pwuid** in-core database.

lreplace *listvarname fieldindex £newvalue*
: This modifies the content of named *listvarname* (property-list) variable by replacing:

  with numeric index
  : the value of indexed element; say "1" would replace second data item in simple chain. If the index goes beyond the chain, the new value is added at the tail of the chain.

  with property-element name
  : the value of possibly existing property-element pair (say: "name1" "value1" "name2" "value2") is replaced with a new one. If the name can not be found, name and its value are appended to the variable.

process *messagefile*
: is the protocol switch function. It is called by the **daemon** function to process a message found in the *POSTOFFICE*/**router** directory. This function will in turn call an internal protocol-specific function which knows the syntax and semantics of the message file. The current version knows about messages submitted using the **MSG_RFC822** parameter to *mail_open*(3zm). For that case, the protocol function is called **rfc822**. *router* has a bit more complex directory semantics, than is stated above. See *zmailer*(3zm) for details. Although the **process** function is provided built in, it is usually overridden by a defined function in the *router* configuration file.

recipient
: is a boolean function that returns the value of the statement "executing a header rewriting function and the address is a recipient address in a message header ".

recase [−u | −l | −p] *string*
: is a case-mapping function that prints the parameter string in either all-uppercase, all-lowercase, or capitalized (pretty).

relation −t *dbtype[,subtype]* [−f *file* −C *file* −e# −s# −bilmnNu% −d *driver*] *name*
: informs the *router* of the existence of a database, and how to access it. It also creates a builtin function with the specified name, which is used to retrieve the value associated with a key in the database. The options are:

  −t *dbtype*
  : is one of the known types of databases, currently:

incore
: is a database maintained in virtual memory (using splay trees). This type should not be used for any database that must periodically be flushed, since not all occupied memory can be freed.

unordered
: is a file with key-value pairs on every line, separated by whitespace. (See about "*−m*"-option!)

ordered
: is a file with key-value pairs on every line, separated by whitespace, sorted by key. (See about "*−m*"-option!)

hostsfile
: is the *hosts*(5) file.

bind
: is the BIND implementation of a Domain Name System resolver. The subtype for this type is the name of a Resource Record type in the **IN** class.

btree
: Is *SleepyCat BSD DB version 2.x* B-TREE database.

bhash
: Is *SleepyCat BSD DB version 2.x* HASH database.

ndbm
: is the new DBM library. The *BSD4.4* has a thing called **db**, which is different thing, but it can be used in place of ndbm via its interface library. (The BSD4.4-db does have only one database file, not two, like ndbm does.)

  This db has a 1024 byte size limit in that the key size plus the data size must be below that limit!

gdbm
: is the GNU implementation of the new DBM library. **Note: GDBM uses ONE file, which is named exactly as you parametrize it, this is unlike NDBM, which appends .dir and .pag to the supplied name!**

dbm
: is the old DBM library. There can be only one DBM open at the time, and this system keeps them all open all the time... Avoid if you can!

  Depends upon systems, some early instances of DBM didn't have close call at all - but encountering such systes is unlikely...

  This db has a 1024 byte size limit in that the key size plus the data size must be below that limit!

yp
: is the Network Information Service from Sun Microsystems Inc. (Latter renamed to be NIS, the still newer NIS+ is not supported)

header
: is an database type used to store RFC822 header semantics information. It is unlikely to be used for anything else. See the HEADERS section below.

ldap
: defines binding to LDAP database. More data available below in "LDAP" section.

−f *file* is a file associated with the database, typically the file containing the data, or the basename of DBM files or something similarly relevant to the database access routines.

−C *file* The SleepyCat database can have an auxiliary configuration file defined by this options. See below for its syntax.

−e# is the default time-to-live on cached information. When the information has been in the cache for this many seconds, it is discarded. The default is 0.

−s# sets the cache size to the specified number of entries. The default is usually 10, depending on the database type.

−b if the key exists in the database, return the key as the value.

−i if the key exists, its value is a byte offset into a file named by the subtype for this database. The value then becomes the concatenation of the data on the lines following that offset which start with whitespace. This is used for the aliases file.

−l map all keys to lowercase before searching.

−m

check for file content modification before every access. Reopen the file when a change is detected. This option is used when the router should discover changes to a database underfoot so it need not be restarted to use new data. This is warmly recommended on relations which use unordered, or ordered datasets (aliases, routes, ...), and especially if the system is configured to use mmap(2) facility.

Updateing such databases should preferrably use *mv* command to move a new version of the database in place of the old one. (**do not use copy!**)

−n if the key exists in the database and the value is null or a list, return the key as value. Otherwise return the value retrieved, if any.

−N *Negative Cache*, if the key is not found from the backend DB, place the result into the cache with configured TTL. (See *-e* option.)

−u map all keys to uppercase before searching.

−d *driver*

specifies a search driver that allows searching for structured keys using special knowledge. The argument to this option must be a known driver.

Currently known drivers are:

pathalias

The lookup sequence for “foo.bar.edu” is:

```
foo.bar.edu
.foo.bar.edu
.bar.edu
.edu
.
```

pathalias.nodot

The lookup sequence for “foo.bar.edu” is:

```
bar.edu
edu
```

longestmatch

The lookup sequence for “foo.bar.edu” is:

```
foo.bar.edu
.bar.edu
```

```
.edu
.
```

−%

Marks that the database results containing '%0' thru '%9' are to be replaced with positional arguments to the database call (via *name*).

The '%0' is always the full key, '%1' **may** be "wildcarded" portion of the key in case any of the driver routines is used, and the original full key does not give a match.

Nominally '%1' thru '%9' are positional options to *name*:

```
result=$(name $keyval $opt1 $opt2 ... $opt9)
```

However like is above mentioned, "wildcarding" lookup drivers may change the indices making "$opt1" above to be '%2', and eight option would become '%9' making ninth inaccessible.

name is the name bound to the lookup function of defined database.

rfc822 *messagefile*
This function controls the parsing and processing of a message file in RFC822/976 format. It is called by the **process** function.

rfc822date
prints the current time in RFC822 format.

runas *user function* [*arguments...*]
changes the current effective user id of the *router* process to that given (which may be numeric or an account name), then runs the specified function with the specified arguments, then switches the effective user id of the process back (to root).

sender is a boolean function that returns the value of the statement "executing a header rewriting function and the address is a sender address in a message header ".

rfc822syntax *address*
This is a simple interface to the address parser. If the command line argument is a syntactically valid RFC822 address, this command is silent and returns 0 as exit status. If there is a parse error, a verbose error message is printed to stdout and the function returns a non-0 exit status.

squirrel { [-]*event* }
sets the kinds of events that cause a message to be copied into the POSTOFFICE/postman directory. The events are: breakin, badheader, illheader, nochannel, nosender. Whether or not a '-' is necessary for an event depends on the current state of the event's flag. The usage message will indicate what to do to toggle the event flag.

stability { on | off }
determines whether the router will process incoming messages in arrival order (when on), or in random order determined by position in the *router* directory. The *router* will by default do the first queue scan in stable mode, and subsequent scans in unstable mode. The name of this command is the name for a similar characteristic of sorting algorithms.

trace *key1 ... keyN*
Enables tracing of the specified items. The valid keywords are:

all turns on all tracing options. You only do this to test the I/O capabilities of your system.

except
flips the sense of tokens following this one.

The *rtrace* routine at the default router scripts is defined as:

trace all except rfc822 regexp

assign print shell variable assignments.

bind prints various information from the code that calls the DNS resolver.

compare
print **sift** statement pattern-selector comparisons.

db print database lookups, including cache search and update information.

final prints the message envelope information after processing each message.

functions
print shell function calls and return values, with nesting indicated by indentation.

matched
print **sift** statement pattern-selector matches.

on same as **functions**.

regexp
prints regular expression matching execution.

resolv turns on the **RES_DEBUG** flag in the **BIND** resolver library, and prints various information from the code that calls the DNS resolver.

rewrite
prints the tokenized addresses sent through the message header address rewriting functions.

router
prints the tokenized addresses sent through the **router** function.

sequencer
prints procedural steps taken during message processing.

memory
prints memory allocation information after each message.

uid2login *uid*
Prints the first account name associated with a specified user id, if any, or **uid#***uid* if no account exists with that user id. It has the same side-effects as the *login2uid* function.

untrace *key1 ... keyN*
Complement of the *trace* function.

In addition any defined relations will create a builtin function with the same name as the relation.

SleepyCat DB parameter file

With SleepyCat DBs, there is an auxiliary parameter for the *relation* call, namely *-C* which points to file defining additional parameters to *SleepyCat DB* opening, like in *Concurrent Data Store* application mode.

Config file syntax for SleepyCat DB 3.x/4.x:

```
envhome = /path/to/envhome/directory
envflags = CDB, CREATE, RO
envmode  = 0644
tempdir  = /path/to/tmp/dir
```

The *envflags* set, can have any/all of:

CDB The database is supposed to have full *Concurrent Data Store* machinery at its disposal. The database may be read-only, but CDB requires internal read-write access to the environment for proper transaction locks to be maintained.

CREATE
Create the database (or environment) if it does not exist, even for read-only access of it...

RO Access is purely read-only.

LDAP DB parameter file

The LDAP setup configuration file is passed to *relation* definition in *-f* option parameter file.

Empty lines, and lines with first non-whitespace character being a '#' are comments. Otherwise lines begin with a keyword (which may have unspecified amount of whitespace in front of it), whitespace after the keyword is skipped, and first non-whitespace character starts the parameter, which continues until end of line.

```
base     ...
attr     ...
filter   ...
uri      ...
ldaphost ...
ldapport ...
protocol [ 2 | 3 ]
scope [base | one | sub]
binddn   ...
debug    ...
passwd   ...
start_tls    [y|n] (or: n= 0|of(f)|f(alse), y=1|on|t(rue) )
authmethod [simple | sasl ]
sasl_secprops ...
sasl_real     ...
sasl_mech     ...
sasl_authc_id ...
sasl_authz_id ...
```

Example:

```
#
# You must at least define "base", "ldaphost", "filter" & "attr".
#

base            o=fooooo,c=fi
ldaphost        10.11.12.13
ldapport        389
binddn          cn=admin,o=fooooo,c=fi
protocol        2
authmethod      simple
passwd          pwpwpwpwpw
filter          (uid=%s)
attr            mail
scope           one
```

GLOBAL VARIABLES

The following shell variables are defined by the *router*(8zm):

defer is set by the database lookup routines to indicate a temporary failure of some kind. The value assigned to this variable is a valid host parameter to the *hold* channel's transport agent.

envelopeinfo
is set by the rfc822 function as soon as possible after the message has been validated, and before the first call to the *router* shell function. It is a property list of at least the following elements:

file The message file name.

message-id
The message-id.

uid The user id of the owner of the message file.

gid The group id of the owner of the message file.

size The message size (header + body) in bytes.

headersize
The message header size in bytes.

bodysize
The message body size in bytes.

now The time the router started processing the message (in seconds since epoch).

delay The number of seconds the message had been queued.

resent A "yes" or "no" indicating whether Resent- headers exist.

trusted
A "yes" or "no" indicating whether file owner is trusted.

Furthermore every envelope header without address semantics will be added to the list, typically:

with The RFC822 Received header "with" value.

via The RFC822 Received header "via" value.

rcvdfrom
The host of the last MTA to touch the message.

etc.

HEADERS

The predefined database **headers** is used to specify to the RFC822 message header parser which headers it should pay attention to and what their semantics are. The database keys are lowercased header field names. The values are strings of three fields separated by a colon (':') character. The first field is the name of the semantic rule in the parser that should be used to parse the header. The second field is either **Sender** or **Recipient** or null (indicated by a **-**), corresponding to whether the header contains addresses and if so which type. The third field is used for a **Resent** flag, which should be given if the header has a proper *Resent-* prefix, otherwise null.

There are two kinds of entries in the **headers** database: mandatory, and optional. The mandatory entries are for those headers that are absolutely necessary for the proper relaying of mail. Typically these are purely address headers. The optional entries are for headers that are not essential to the relaying of mail, but whose semantics are in fact specified in RFC822. For example, here is a representative sample of the contents of the database:

sender	Mailbox:Sender:-	(permanent)
bcc	Addresses:Recipient:-	(permanent)
from	AMailboxList:Sender:-	(permanent)
message-id	MessageID:-:-	(permanent)
reply-to	AddressList:Sender:-	(permanent)
resent-from	AMailboxList:Sender:Resent	(permanent)
resent-sender	Mailbox:Sender:Resent	(permanent)
return-receipt-to	AddressList:Sender:-	(optional)
return-path	Mailbox:Sender:-	(optional)
date	DateTime:-:-	(optional)
encrypted	Encrypted:-:-	(optional)
errors-to	AddressList:Sender:-	(optional)

keywords PhraseList:-:- (optional)

You may decide to add or remove header definitions from this database. This is done using the normal database interface function, **db**. For example, if you want to disable the automatic checking (and rewriting) of *Date* message headers, you would do "db delete headers date". Such actions should never be done lightly, since it will likely cause violation of the RFC822 protocol when transferring mail to other mailers.

SIGNALS

SIGHUP:

execution of the shell trap handler is deferred until a sequence point. This makes it easier to do log rollovers entirely in the configuration file, without fear of data corruption.

SIGTERM:

exit cleanly (immediately if idle, otherwise after finishing with the message being processed).

Other signals may be handled by shell traps.

Z−ENVIRONMENT VARIABLES

MAILVAR

MAILSHARE

these two tell the directories where the configuration files and routing databases are located at. The *MAILVAR* files are host dependent, while *MAILSHARE* files can be common to all of the campus.

NROUTERS

tells how many parallel routers there may be running.

NOBODY

this tells the cornerstone of the system security -- who is *nobody*. It can tell either a numeric userid, or account name.

LOGDIR

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

LOGLEVEL

**** document missing ****

POSTOFFICE

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

ROUTERDIRS

defines a ':' separated list of alternate router directories. If these are defined at all, they **must** exist, if alternate queueing priority mechanism is desired to be used.
Example: **ROUTERDIRS=router1:router2:router3:router4**

ROUTERNOTIFY

defines an *AF_UNIX/DGRAM* type local notification socket into which a receiving client *may* inform the *router*(8zm) that there is some new job available.

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.

SYSLOGFLG
**** document missing ****

FILES

/opt/mail/zmailer.conf	(ZCONFIG)
/var/spool/postoffice/.pid.router	(POSTOFFICE/.pid.router)
/var/spool/postoffice/.router.notify	(POSTOFFICE/.router.notify)
/var/spool/postoffice/.scheduler.notify	(POSTOFFICE/.scheduler.notify)
/var/spool/postoffice/router	(POSTOFFICE/router)
/var/spool/postoffice/{ROUTERDIRS}	(POSTOFFICE/{ROUTERDIRS})

SEE ALSO

mailq(1zm), *zmailer*(3zm), *scheduler*(8zm), *zmsh*(1zm), *zmailer.conf*(5zm).

AUTHOR

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