mapSoN 2.3-beta-3 User Manual

Peter Simons

simons@computer.org

Table of Contents

1. Introduction

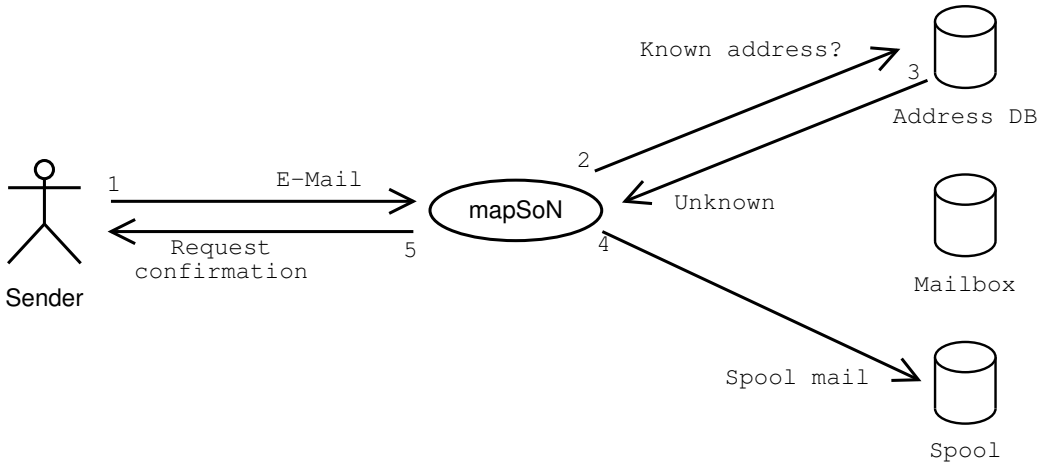
mapSoN is a spam filter that uses a pretty unique approach to keep unsolicited commercial e-mail out of your mailbox. Rather than using a set of configured “bad words”, a list of “know spammers”, or complicated scoring mechanisms to determine what is spam and what is not, it relies on “known senders” -- or rather “unknown senders”.

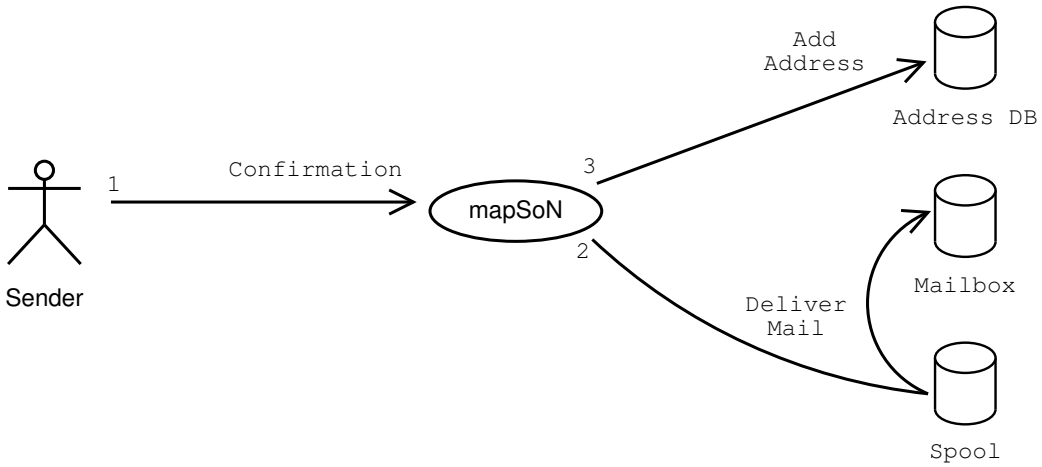
Every time you receive an e-mail, mapSoN will look-up the sender’s e-mail address in a small database file and check whether that address is in there. If it is, the mail is delivered to your mailbox, but if it is not, the e-mail will be stored in a spool directory in your home, using a cryptographic cookie as the filename. Then mapSoN will send a so called *request for confirmation* to the sender’s address, asking him to please confirm his addresses validity by replying and sending the cryptograpic cookie back. When mapSoN receives a mail with such a cookie in it, it will move the corresponding mail from the spool directory to your mailbox and add the sender’s address in the mail to the database.

This approach is based on the fact that spammers usually fake the sender address of the spam mail. (In fact, they *have to*, because sending unsolicited advertisement via e-mail is illegal in most countries.) But because their sender address is invalid, they will never see the request for confirmation, they will never reply, and their spam will sit in that spool file until hell freezes over or an apropriate cron job deletes it. Using this heuristic, mapSoN catches way above 95% of all spam mail I receive.

In order to avoid annoying more “real” people, who are trying to contact you, than necessary, you can import the addresses from your mail archive into the mapSoN database. Furthermore, you can set mapSoN up in a way that will let any mail pass automatically, that is a reply to a mail or a news posting of yours: If you sent someone an e-mail and he replies back, mapSoN won’t bother him. It would be pretty inpolite, if it did.

This illustration will probably help not all and only add to the confusion, but I made it, and now I have to include it here -- helpful or not!





2. How effective is mapSoN

You may wonder how effective mapSoN is -- and rightfully so. Of course, being the principal author of this tool, I am biased. But I tried my best to conduct an objective study, determining how many spam mails have been caught, how wany mapSoN did not catch, and how many mails were delayed that weren't spam. The numbers presented below are somewhat skewed, because the logfile I analyzed includes the various test mails I piped into mapSoN in order to test and to debug it, but if at all, they tend to make the result look worse because I did not test the case that a mail comes in and is approved, I tested the case that a mail is *deferred*.

Anyway, I activated mapSoN for my private mail account on January 11th, 2002. This analysis was made on April 10th the same year, so we have a test period of 90 days -- almost three months. Here are the numbers:

Address Database entries, imported:	4407
Address Database entries, today:	4606
Total number of mails received:	7632
Mails coming from mailing lists:	2104
Number of processed mails:	1561
Number of passed mails:	395
Number of mails deferred:	998
Number of mails confirmed:	67
Number of mails unacknowledged "real" mails:	14
Number of definite junk mails:	862

How did I get those numbers? First of all, I counted all logfile entries coming from sendmail, which contained the strings "mailer=local" and "to=<simons". This I assumed to be the number of total mails received. On those entries, I counted how many of those contained the string "to=<simons+" -- what should largely be the mails I receive via mailing lists because I use procmail's argument feature when I subscribe to mailing lists.

All those mails, addressed to "simons+something" bypassed mapSoN automatically. Furthermore, all mails that contained certain headers like `In-Reply-To` or `References` bypassed mapSoN entirely because I assume those to be replies to messages of mine. This explains why the number of mails that mapSoN actually processed is much smaller than the number of total mails received: Apparently only 20% of all incoming e-mail were processed by mapSoN at

all!

Then I counted the entries in mapSoN's logfile that sayed "passed", which turned out to be 25% of all mails mapSoN saw. The mails that mapSoN did not let through amount to 63% of all mails processed by mapSoN, and I determined those by counting the logfile entries that said "Spooling e-mail".

Of those deferred e-mails, only 6% were confirmed later! So then I waded through the spool directory and moved all "regular" e-mails to a separate folder and counted them: It turned out that regular 14 mails were *not* confirmed by the sender, that's 0.14% of all deferred mails and 0.0018% of the total number of mails received.

When I looked at those mails in detail, it turned out that 3 of the 14 mails lying in the spool "unwarrantedly" actually *had* been acknowledged, but that the other person was too dumb to reply correctly: Two of the three sent a *new* e-mail, which did of course not contain the cookie, and the third person replied to the request for confirmation, but erased the cookie from the mail manually. It's no wonder that the mails they sent me turned out to be of the kind that I don't want anyway.

The remaining 11 mails that were not delivered to me but were not spam either, were some kind of replies I got from Internet sites like amazon.com, other customer service stuff and automated reminders. No personal e-mail. They were delayed because they were sent with incorrect sender addresses that bounced when mapSoN sent the request for confirmation back or that were apparently not read by the other end at alll.

Once I noticed the problem, I got it fixed quickly by using "user+something" addresses at sites like amazon.com, too, so that their mails bypass mapSoN to begin with.

Of course there was a certain amount of spam that got past mapSoN one way or the other. Some used addresses that actually were in my database because I had imported them from my mail archives when I set mapSoN up. Some others were routed past mapSoN by my procmail configuration because they looked like bounces or postings coming from mailing lists. Unfortunately, I cannot determine any exact number without wading through my mail archive manually, and I honestly don't want to do that. After all, the whole point of writing mapSoN was that I do *not* have to see spam!

To summarize: In the 90 days of testing, mapSoN caught 862 definite spam mails: That's about 10 per day. It reduced the amount of spam in my mailbox to one or two mails once a week. At the same time, only 0.008 percent of all e-mail I receive had to be acknowledged, what I think is an acceptable level of inconvenience for my communication partners, especially given the fact that obviously they were contacting *me*, not the other way round.

3. Building mapSoN

Compiling the software should be pretty straight-forward; all you'll have to do is the usual routine:

```
./configure
make
make install
```

The last step may require super-user privileges, so have the root password ready if you're going for a system-wide installation.

Be adviced that mapSoN needs a fairly recent C++ compiler because it makes full use of the new ISO C++ language features. If you're using the GNU C Compiler (http://gcc.gnu.org/) version 2.95 or later, you won't have any problems. But other compilers are known not to be ISO C++ compatible. If you're having trouble, send me an e-mail and I'll see what I can do.

The `configure` script is a standard GNU Autoconf (http://www.gnu.org/software/autoconf/) script, which supports all the usual options. If you are not familiar with these scripts, please refer to the "Running `configure` Scripts" section of the Autoconf user manual, which is available at:

http://www.gnu.org/manual/autoconf/html_chapter/autoconf_13.html#SEC129

The following list will show only those options that are specific to mapSoN.

`--with-mailboxdir=`*`DIR`*

In order to deliver incoming e-mail to your mailbox, mapSoN needs to know where the user mailboxes are located on your system. It tries to figure that out automatically by checking for the existance of the directories `/var/mail` and `/var/spool/mail`, but depending on your setup, you might want to choose another path here.

`--with-mta=`*`PATH`*

mapSoN needs to know the complete path to your system's mail transport agent (MTA) in order to send out requests for confirmation. The `configure` script will assume that you have sendmail installed and look for it in various locations, but you can (and may have to) set the right choice manually using this option.

`--with-debug`

Per default, mapSoN comes with a couple of additional debug messages, which you can enable on the command line or in the configuration file if you feel something's going wrong. But in order for these log messages to be available, the binary must have been compiled with the DEBUG flag. Using this option, you can manually decide whether you want these log messages compiled in or not.

Any of the paths you configure here are only used as defaults. You can override them at run-time in the configuration file mapSoN reads at startup.

If your using a fairly recent gcc version to compile this, you might want to try

```
CXXFLAGS=-frepo LDFLAGS=-frepo ./configure
```

to configure the build. This will result in an about 30% smaller binary because unused template instances are optimized out.

On some platforms, largefile support doesn't work yet. If you're getting compiler errors, give the configure script the option *`--disable-largefile`* and try again.

4. How to Activate mapSoN

Assuming, you have built and installed the mapSoN package sucessfully, you must do two things to activate it: Create the directory "`.mapson`" in your home directory and tell your Mail Transport Agent to pipe incoming local mail into mapSoN rather than to deliver it to the mailbox directly.

The first step should be manageable without further instructions, but installing mapSoN as local mailer is non-trivial. The rest of this chapter is divided into separate sections that will describe the various possible setups. How mapSoN must be installed depends entirely on the Mail Transport Agent you use, so if your configuration is not discussed in this manual, please consult your MTA's user manual instead. And if you found out how to do it, please write a short paragraph about it and let me know so that I can include it in the next version!

4.1. Using sendmail's `.forward` mechanism

Before I start, let me give you one piece of advice: Using mapSoN without any further tool installed that will allow you to filter and to redirect incoming mail into different folders will *not* make you happy. This kind of installation is nice to figure out whether mapSoN is useful to you, but you *should* install procmail or a similar program as soon as possible. Trust me.

Anyway, sendmail (http://www.sendmail.org/) uses a simple mechanism to forward incoming local mail into application programs: A file named `.forward`, that must be located in your home directory. To activate mapSoN, all you have to do is to create that file and put the following line into it:

```
"|exec /usr/local/bin/mapson"
```

Apparently, on some systems the home directory must grant execute-permission to "other" for the sendmail to evaluate that file. So if you created the `.forward` file as shown above and still there's no sign of any mapSoN activity, execute **`chmod 711 $HOME`** and try again.

Another potential obstacle is that some sendmail installations use the restricted shell (smrsh) for the execution of the local mailer. This shell will not allow users to execute arbitrary commands in the `.forward` file. If your system uses smrsh, you must create a link from `/usr/local/bin/mapson` to `/usr/adm/sm.bin/mapson` in order to enable mapSoN. (The paths may vary from system to system, obviously.)

4.2. Using procmail

Most systems these days use procmail (http://www.procmail.org/) to deliver local mail. This means, that you can configure your local mailer by adding cryptic recipes in an entirely undocumented syntax to the file `.procmailrc` in your home directory:

```
ARGUMENT="$1"

# Have mapSoN accept anything that is a reply to a
# message of mine.
#
:0 w
* ^(In-Reply-To|References|Message-Id):.*example.org
| /usr/local/bin/mapson --accept

# Confirmation mails go into mapSoN.
#
:0
* ARGUMENT ?? [a-f0-9]...repeat 32 times...[a-f0-9]
| /usr/local/bin/mapson --cookie $ARGUMENT

# Forward the mail into mapSoN for approval unless
#   - it has an argument,
#   - is a bounce,
#   - comes from a mailing list, or
#   - is an automatically generated mail.
#
:0 w
* !ARGUMENT ?? ..*
```

```
* !^FROM_DAEMON
* !^Precedence: (list|bulk|junk)
* !^Auto-Submitted:
| /usr/local/bin/mapson
```

Don't panic, I know this recipe looks like hell, and to be perfectly host, it took me hours to get it working the way I wanted it. But all you have to do is to copy it into the `.procmailrc` file in your home directory ... Be sure, though, to customize the parts marked *`replaceable`* for your system -- in particular the path to the mapSoN binary and the domain name of your e-mail address.

In case you want to know what this thing does, though, read on!

procmail has an incredibly useful feature that this recipe makes use of: The argument. If your address is, say, `user@example.org`, then you can receive e-mail under the address `user+foo@example.org`, too. You can append any string to your username by a plus sign as long as the result is still a valid e-mail address. procmail will still deliver these mails to `user`'s mailbox; the parameter is effectively ignored. But you can use that parameter to sort mail being sent to different addresses into different folders reliably!

If you subscribe to the mailing list "cat-lovers", for example, you could subscribe the address `user+cat-lovers@example.org` instead of your ordinary address. That mail would still reach you, but with the recipe

```
:0
* ARGUMENT ?? cat-lovers
/var/spool/mail/user2
```

you can easily sort the list's articles into a different folder!

In the recipe shown above, all mail that has such an argument will bypass mapSoN, and for a good reason: You don't want mapSoN to process mail that is delivered to you via a mailing list! It would be incredibly unpolite to request a confirmation from someone who posted to the mailing list and did not mail *you* at all -- at least not deliberately. To make matters worse, the request-for-confirmation mail would not even reach the poster, but would be sent to the mailing list administrator because most mailing lists re-write the envelope of the mails delivered via them to that address.

So, to avoid all that mess: Subscribe under a `user+something` address and you won't have any problems. Plus: You can sort mail from different lists into different folders easily, if you want that.

Furthermore, any mail that just looks remotely as if it's not coming from a human sender will bypass mapSoN, too. This means that you'll receive a spam mail from time to time, but this is essential to avoid infinite mail looks, for instance. You don't want mapSoN to send a request for confirmation in response to a bounce mail that has been created because a former request for confirmation could not be delivered, etc.

Another nice thing is that the first rule in the recipe will make mapSoN accept any mail that looks as if it is a reply to an article of yours. If someone is replying to an e-mail or a news posting of yours, his mail reader will (hopefully) add the `In-Reply-To` header pointing to the message id of your article. And since message ids contain the hostname of the site that created the article, the first rule will recognize this and let the mail pass and add his address to the database.

The second rule will ensure that confirmation mails are processed correctly. In order to take advantage of that, you will have to edit your request-for-confirmation template (see Section 7) so that the `From:` line looks like this:

```
From: user+${MD5HASH}@example.com (Real Name's Anti-Spam-Tool)
```

In essence, this means that the person replying to the request will have the cookie put into procmail's argument automatically! If you don't want to use this, just don't -- mapSoN will find the cookies in the mail headers or body, too, but this approach is very error resistent. You wouldn't belive how many people are too dumb to understand "please reply and include that string in the mail: [...]".

In case you're wondering: The string "`[0-9]`" in the recipe must indeed be repeated exactly *32* times, because a cookie consists of 32 characters in the range of `a` to `f` or `0` to `9`. Some regular expression libraries allow to shortcut this expression as `[a-f0-9]{32}`, but apparently the one shipped with procmail is not one of them. At least on my machines, I was not able to make that work.

One more general advice: Obviously the mapSoN-related recipes must be at the end of your `.procmailrc` file. Once mapSoN ran, the mail is processed and any recipes following below won't be invoked unless you do some heavy procmail magic. You have been warned.

4.3. Using fetchmail

Not everybody gets his e-mail delivered via SMTP, thus, not everybody can install mapSoN on the machine that actually accepts the incoming mail. If you use POP3 or IMAP for example, your mail will have been accepted by your mail server already and you just fetch it from there.

Luckily, you can still use mapSoN, but you'll have to use a tool like fetchmail (http://www.tuxedo.org/~esr/fetchmail/). fetchmail will fetch the mails lying on your mail server via POP3, IMAP, or whatever and then invoke sendmail locally to actually deliver the mail to your mailbox. Hence, you can use the installation desribed in Section 4.2, too.

If you don't want to bother setting up sendmail -- and I could understand that --, tell fetchmail to call procmail as the delivery agent and you're fine. Use the following entry in your `.fetchmailrc` file:

```
poll mailserver.example.org mda "/usr/local/bin/procmail -d username"
```

Unfortunately, you cannot use procmail's argument feature in this setup, unless you can talk your e-mail provider into using procmail himself. If he does not, the `user+foo` username will yield an "unknown user" on his mail server otherwise.

5. Command Line Syntax

mapSoN understans several optional parameters on the command line, which allow you to override the compiled-in default or the settings in the config file. The standard Unix synopsis line is:

mapson [-h | --help] [--version] [-d | --debug] [-a | --accept] [--cookie *cookie*] [-c *config* | --config-file *config*] [--dont-scan] [*mail*...]

Here is a list of all options together with a short description of what the respective option does:

`-h`
`--help`

Show mapSoN usage information.

`--version`

Show mapSoN's version string.

`-d`
`--debug`

Enable debugging. Please note that debugging is only available if mapSoN has been compiled with the define DEBUG. Otherwise, the debug code is not included in the binary.

`-a`
`--accept`

Accept the incoming e-mail unconditionally and add the sender's addresses to the database.

`--cookie` *`cookie`*

Using this parameter, you can specify a cookie on the command line. mapSoN will then try to approve the corresponding mail from the spool. If the cookie turns out to be incorrect, mapSoN will continue to process the mail as if none had been specified. That means, though, that if a valid cookie is found in the mail itself, it will approve the corresponding mail nonetheless.

`-c` *`config`*
`--config-file` *`config`*

Use the configuration file *`config`* rather the default.

`--dont-scan`

Do not scan for cookies in the incoming e-mail. This is useful in case you're using procmail (or some similar mechanism) to direct cookies to special addresses and thus can use the --cookie option rather than to have mapSoN look through the mail for one.

`mail ...`

If any parameter is specified on the command line that is not an option, mapSoN will go into *gather addresses* mode. The parameters are interpreted as filenames, each of the files containing an e-mail that mapSoN will parse. Any sender address mapSoN finds in these mails will be added to the database of known addresses. This mode is meant to import addresses from your mail archive to the database.

6. The mapSoN Configuration File

At startup, mapSoN will try to read its configuration file at `${HOME}/.mapson/config` first of all. If that file does not exist, mapSoN falls back to the system-wide file `/usr/local/etc/mapson.config`. This means that if mapSoN has been installed well, you don't need a configuration file of your own. It also means, that you may *have* a configuration file of your own nonetheless.

The config file is read line by line. Empty lines are ignored, as are lines starting with the commend delimiter "`#`". Anything else is supposed to start with one of the keywords listed below, followed by one or more whitespace characters, followed by the actual data part.

The data part may contain environment variables, which you can use to have one configuration file fit all of your users! See the sample configuration file installed at `/usr/local/share/mapson/reqmail.template-sample` for such an example.

Valid keywords in the configuration file are:

`Mailbox` *`file`*

This directive sets the complete path of the mailbox file, where mapSoN stores approved mails. Unless, of course, the parameter configured here starts with a pipe sign ("`|`"), as in "`|/usr/sbin/sendmail foo@example.org`". In this case, mapSoN will pipe the mail to the standard input channel of this command rather than to write it to a file.

`SpoolDir` *`directory`*

This directive sets the complete path to the directory, in which deferred mails will be spooled until a confirmation arrives for them.

`AddressDB` *`file`*

This directive sets the complete path of the file mapSoN uses to store the "known" addresses.

`AddressDBAutoAdd` *`boolean`*

If this directive is set to `true` (the default), then mapSoN will add so-far unknown addresses from which mail has been accepted to the database automatically, so that in future mails from these addresses will pass. You can disable this behavior, though, in case you want to maintain that database manually or by other means.

`ReqConfirmTemplate` *`file`*

This directive sets the complete path to the request-for-confirmation template file mapSoN uses to generate the request-for-confirmation mail sent to first-time originators.

An arbitrary number of alternate paths can be specified, if they're separated by colons, for example:

```
$HOME/.mapson/reqmail.template:$DATADIR/reqmail.template:...
```

In this setup, mapSoN would first try to load the file `$HOME/.mapson/reqmail.template`. If that failed, it would try `$DATADIR/reqmail.template`, and so on, until one of the files can be loaded successfully.

This is an extremely useful feature if you are a system administrator who wishes to allow all users of the system to use mapSoN without having to create a request-for-confirmation template of their own: Configure mapSoN to load that request-for-confirmation template first, that is located in the user's home directory. If this file does not exist, then fall back to the system-wide file.

In effect, that means that the user can simply use mapSoN to filter his mail, and if he ever feels like it, he can create a request-for-confirmantion template file of his own and it will be preferred over the system-wide one.

`MTA` *`command`*

This directive sets the command mapSoN will use to send-out a request-for-confirmation mail. The actual mail will be piped into the started process.

`PassIncorrectMails` *`boolean`*

When mapSoN parses the incoming mail's headers for the addresses, it may detect syntax errors in the mail header, that do not cause a fatal error, but that surely hint to the fact that this mail was not created by an RFC822-conformant mail client.

Many spam mails contain incorrect header lines, so you may chose to have mapSoN fail on *any* syntax error -- even non-fatal ones. "Failing" means that mapSoN will abort and return the return code configured below to the MTA. Depending on the setting of the return code, the MTA will then bounce the mail.

The parameter given to this option is a boolean, meaning that you may specify either `yes` or `no`.

`StrictRFCParser` *`boolean`*

If you enable this option by specifying `yes`, mapSoN will perform additional syntax checks on the incoming mail, if you say `no`, it will check only those headers that are needed for mapSoN to operate at all.

Enabling this option makes little sense unless you disable the PassIncorrectMails option.

`RuntimeErrorRC` *`integer`*

This directive sets the return code mapSoN exits with in case it had to abort with a run-time error. Possible run-time errors are failure to open file, lack of available memory, etc. ...

The default choice is "75", which sendmail will interpret as a temporary system error, so it will queue the mail and re-try.

A valid return code is a positive integer up to 128.

`SyntaxErrorRC` *`integer`*

This directive sets the return code mapSoN exits with in case it encountered a fatal syntax error in the e-mail. If PassIncorrectMails is disabled, non-fatal syntax errors will also cause mapSoN to abort with this return code.

The default choice is "65", which sendmail will interpret as a permanent error that causes the mail to bounce.

A valid return code is a positive integer up to 128.

`Debug` *`boolean`*

If you enable debugging messages by saying `yes` here, mapSoN will log additional information about its proccssing of the mail. If you say `no`, mapSoN will log only very few messages at all.

Debugging is available only when the binary has been compiled with the DEBUG symbol defined. Currently, that is the default, though, so unless you exclicitely disabled it, debugging will be available.

`LogFile` *`file`*

This directive sets the complete path of the file mapSoN uses to log its actions.

In order to make the contents of the configuration file as independent from the system's directory structure as possible, mapSoN provides a set of environment variables, which are guaranteed to be defined. You can use them anywhere in the data part of a configuration directive, and you can use the usual manipulations on them.

Environment variables are looked-up case-sensitively, so `$home` is not the same thing as `$HOME`. This behavior is different in the request-for-confirmation template, where you can spell the variables upper- or lower-case as you wish.

That's because the variables there are not coming from the environmnet, but are mapSoN's internal variables. So be sure not to confuse that, because an undefined variable in this file will cause mapSoN to abort with an error.

Here is the complete list:

`$MAILBOXDIR`

> This variable contains the complete path of directory, in which the system's mailboxes are located, usually `/var/spool/mail`. Please note that the value provided here is the one determined at *compile-time*, so if you changed your system's installation and want to rely on this variable, you'll have to re-compile.

`$MTA`

> This variable contains the path to the systems mail transport agent. Please note, that this is only the path of the executable -- for example `/usr/sbin/sendmail` --, the variable does not contain the flags that must be passed to the MTA in order to do something useful.

`$DATADIR`

> This variable contains the complete path of the directory, which has been compiled into mapSoN as the directory where read-only architecture-independent data should be stored. You will, for example, find the system-wide request-for-confirmation template file here.

`$USER`

> This variable contains the name of the user under which mapSoN is running. Depending on your MTA, this must not necessarily be the user who is receiving mail! If you're using sendmail, though, you're on the secure side.

`$HOME`

> This variable contains the complete path of `$USER`'s home directory.

7. The Request-for-Confirmation File

When mapSoN issues a request for confirmation, it will try to load the template file containing the text to be used for this purpose. Unless configured otherwise in the configuration file (see Section 6), the first path to look is `${HOME}/.mapson/reqmail.template`; `${HOME}` meaning the home directory of the user under which's id mapSoN is running under.

If that file does not exist, mapSoN will fall back to the system-wide file at `/usr/local/share/mapson/reqmail.template`. If this file doesn't exist either, mapSoN will abort with an error.

The request-for-configuration template file is supposed to contain a complete RFC822 message, including headers and everything. The actual request-for-confirmation mail is created by loading the template and expanding the variables contained in it. The result is then piped into the command, you have configured to use to access the Mail Transport Agent.

Here is an example of a request-for-confirmation template you might use:

```
From: username@example.com (Real Name's Anti-Spam-Tool)
To: ${ENVELOPE:-${RETURN_PATH:-${SENDER}}}
Subject: please confirm [${MD5HASH}]
```

```
Precedence: junk
Auto-Submitted: auto-generated
References: $MESSAGEID
In-Reply-To: $MESSAGEID

This is an automated request for confirmation in order to make
sure that the message quote below was actually sent by you. You
don't wanna know the details, trust me. Just press <reply>
and send me a mail back without changing that cookie in the subject
line, that's it. You will never have to do that again -- sorry for
the inconvenience!

Your mail was:

[ | ${HEADER[#]}] |
[${BODY[#]:+ | }${BODY[#]}]{0,5} | \[...\]
```

mapSoN will replace the variables you see in this example by the actual values from the incoming mail and deliver the confirmation request. Don't panic, there's a pretty good template included in the distribution (see Appendix B) that you can use, you don't have to worry about the variable stuff too much if you don't want to. For those who want to ... Here is the complete list of variables provided by mapSoN for this file:

`$MD5HASH`

mapSoN will calculate an MD5 checksum of the received mail and make that result available in this variable. This string will also be used as the filename of the mail in the spool directory, by the way. Your request-for-confirmation template *must* contain this string somewhere, or mapSoN won't be able to process the confirmation when it arrives.

A good idea is to place the cookie in the Subject of the mail, because users are less likely to erase it there by accident. (Friendly euphemism for "stupidity".)

`$ENVELOPE`

This variable contains the envelope of the incoming mail. The "envelope" is the address that was given as the sender during the SMTP dialog when the mail is transported. It's usually the only address that's not entirely trivial to fake or mess up, so you should use this one whenever possible to send the request for confirmation to.

Unfortunately, the envelope is not available in the standard RFC822 message format, but under Unix, it is customary to include it in the very first "From␣" line. At least sendmail does that.

`$SENDER`

This variable will expand to the address stated in the message's "Sender:" header.

`$RETURN_PATH`

This variable will expand to the address stated in the message's "Return-Path:" header.

`$HEADER`

This variable contains the complete headers of the incoming mail.

`$BODY`

This variable contains the complete body of the incoming mail. Be careful, this may be long!

`$MESSAGEID`

This variable contains the contents of the incoming mail's "Message-Id:" header.

In addition to those, the following arrays are provided:

`$HEADERLINES[]`

This array contains one text line of the message's header per entry.

`$HEADER[]`

This array contains one of the message's header lines per entry. A "header line" in this context means actually several text lines, because RFC822 headers may span over multiple lines if the next line starts with whitespace.

`$BODY[]`

This array contains one text line of the message's body per entry.

In addition to those, you can access any environment variable available at run-time. The template file included in this distribution, for example, will use `${USER}` in order to make the template file independent of the user who's actually running mapSoN. Don't be to daring, though, not every environment variable you can see in your shell will be set when sendmail, procmail, or whoever calls mapSoN! The only environment variables that are guaranteed to be available are those list in Section 6.

One more thing: The variables listed explicitely in this section can be access case-insensitively. `${bOdY}` is the same as `${boDY}`, because these are variables provided by mapSoN internally. But environment variables like `${USER}` must be accessed in upper-case!

8. Variable-Expression Magic

Throughout mapSoN, the user may specify variables in the text files in order to have their actual contents inserted at the apropriate location. This is a functionality provided by libvarexp. Hence, this section has been inserted verbatim from libvarexp's documentation. Please don't worry if the documentation says things like "implementation defined", etc. Just read about the expressions the library provides you with and how you can use them. Anything you need to know is included in *this* document.

If you're interested in incorporating libvarexp into programs of your own, though, check out the copy available in the `libvarexp` directory in the mapSoN distribution or take a look at libvarexp's homepage (http://cryp.to/libvarexp/) for further details.

8.1. Variable Expressions

libvarexp distinguishes variables into simple and complex expressions. A simple expression has the form "$NAME" and will basically only replace the variable in the text buffer with its contents. Complex expressions have the form "${NAME:operation1:operation2:...}" and may perform various operations on the variable's contents before inserting it into the text buffer.

Please note that due to the way simple expressions are parsed, it may not always be possible to use the simple-expression form even though you do not want to perform any operations. If your input text was “This is a $FOObar”, but the last “bar” part is meant to be a literal string, you’d have use “This is a ${FOO}bar”, because the parser will interpret any valid variable-name character following the dollar as part of the variable name; it will not recognize that “$FOO” would exist while “$FOObar” would not.

Also, libvarexp does not distinguish case in any way. For the library, “$FoObAr” and “$fOoBaR” are just strings -- whether they refer to the same variable or not is entirely up to the application that provides the callback used to resolve variables to their contents.

If you want to enter a text like “$foo” literally, you’ll have to escape the “$” sign by prefacing it with a backslash: “\$foo”. Then libvarexp won’t interpret this expression as a variable.

8.2. Operations on Variables

In addition to just inserting the variable’s contents into the buffer, you can use various operations to modify its contents before the expression is expanded. Such operations are used by appending a colon plus the apropriate command character to the variable name in complex expression, for example: “${FOOBAR:l}”. Furthermore, you can chain any number of operations simply by appending another command to the last one: “${FOOBAR:l:u:l:u:... }”.

The supported operations are:

`${NAME:#}`

> This operation will expand the expression to the length of the contents of `$NAME`. If, for example, `$FOO` is “foobar”, then `${FOO:#}` will result in “6”.

`${NAME:l}`

> This operation will turn the contents of `$NAME` to all lower-case, using the system routine tolower(3).

`${NAME:u}`

> This operation will turn the contents of `$NAME` to all upper-case, using the system routine toupper(3).

`${NAME:*`*`word`*`}`

> This operation will expand to *`word`* if `$NAME` is empty. If `$NAME` is not empty, it will expand to an empty string.
>
> *`word`* can be an arbitrary text. In particular, it may contain other variables or even complex variable expressions, for example: “${FOO:*${BAR:u}}”.

`${NAME:-`*`word`*`}`

> This operation will expand to *`word`* if `$NAME` is empty. If `$NAME` is not empty, it will evaluate to the `$NAME`’s contents.
>
> *`word`* can be an arbitrary text. In particular, it may contain other variables or even complex variable expressions, for example: “${FOO:-${BAR:u}}”.

`${NAME:+word}`

This operation will expand to *word* if `$NAME` is not empty. If `$NAME` is empty, it will expand to an empty string.

word can be an arbitrary text. In particular, it may contain other variables or even complex variable expressions, for example: "${FOO:+${BAR:u}}".

`${NAME:o`*start*`,`*end*`}`

This operation will expand to a part of `$NAME`'s contents, which starts at *start* and ends at *end*. Both parameters *start* and *end* are unsigned numbers.

Please note that the character at position *end* is *included* in the result; "${FOOBAR:o3,4}", for instance, will return a two-character string. Also, please note that start positions begin at zero (0)!

If the *end* parameter is not specified, as in "${FOOBAR:o3,}", the operation will return the string starting from position 3 to the end of the string.

`${NAME:o`*start*`-`*length*`}`

This operation will expand to a part of `$NAME`'s contents, which starts at *start* and ends at "*start*+*length*". Both parameters *start* and *end* are unsigned numbers.

"${FOOBAR:o3-4}", for example, means to return the next 4 charaters starting at position 3 in the string. Please note that start positions begin at zero (0)!

If the *end* parameter is left out, as in "${FOOBAR:o3-}", the operation will return the string from position 3 to the end.

`${NAME:s/`*pattern*`/`*string*`/gti}`

This operation will perform a search-and-replace operation on the contents of `$NAME` and return the result. The behavior of the search-and-replace may be modified by the following flags: If a `t` flag has been provided, a plain text search-and-replace is performed, otherwise, the default is to do a regular expression search-and-replace as in the system utility sed(1). If the `g` flag has been provided, the search-and-replace will replace *all* instances of *pattern* by *replace*, instead of replacing only the first instance (the default). If the `i` flag has been provided, the search-and-replace will take place case-insensitively, otherwise, the default is to search case-sensitively.

The parameters *pattern* and *replace* can be an arbitrary text. In particular, they may contain other variables or even complex variable expressions, for example: "${FOO:s/${BAR:u}/$FOO/ti}".

`${NAME:y/`*ochars*`/`*nchars*`/}`

This operation will translate all characters in the contents of `$NAME` that are found in the *ochars* class to the corresponding character in the *nchars* class -- just like the system utility tr(1) does. Both *ochars* and *nchars* may contain character range specifications, for example "a-z0-9". A hyphon as the first or last character of the class specification is interpreted literally. Both the *ochars* and the *nchars* class must contain the same number of characters after all ranges are expanded, or an error is returned.

If, for example, "$FOO" contains "foobar", then "${FOO:y/a-z/A-Z/}" would yield "FOOBAR". Another goodie is to use that operation to ROT13-encrypt or decrypt a string with the expression "${FOO:y/a-z/n-za-m/}".

The parameters *ochars* and *nchars* can be an arbitrary text. In particular, they may contain other variables or even complex variable expressions, for example: "${FOO:y/${BAR:u}/$TEST/}".

`${NAME:p/`*width*`/`*string*`/`*align*`}`

This operation will pad the contents of `$NAME` with *string* according to the *align* parameter, so that the result is at least *width* characters long. Valid parameters for align are `l` (left), `r` (right), or `c` (center). The *string* parameter may contain multiple characters, if you see any use for that.

If, for example, "$FOO" is "foobar", then "${FOO:p/20/./c}" would yield ".....foobar....."; "${FOO:p/20/./l}" would yield "foobar....."; and "${FOO:p/20/./r}" would yield ".....foobar";

The parameter *string* can be an arbitrary text. In particular, it may contain other variables or even complex variable expressions, for example: "${FOO:p/20/${BAR}/r/}".

8.3. Quoted Pairs

In addition to the variable expressions discussed in the previous sections, libvarexp can also be used to expand so called "quoted pairs" in the text. Quoted pairs are well-known from programming languages like C, for example. A quoted pair consists of the backslash followed by another character, for example: "\n".

Any character can be quoted by a backslash; the terms "\=" or "\@", for instance, are valid quoted pairs. But these quoted pairs don't have any special meaning to the library and will be expanded to the quoted character itself. There is a number of quoted pairs, though, that does have a special meaning and expands to some other value. The complete list is shown below. Please note that the name "quoted pair" is actually a bit inaccurate, because libvarexp supports some expressions that are no "pairs" in the sense that they consist of more than one quoted character. But the name "quoted pair" is very common for them anyway, so I stuck with it.

The quoted pairs supported by libvarexp are:

\t
\r
\n

These expressions are replaced by a tab, a carrige return and a newline respectively.

abb

This expression is replaced by the value of the octal number *abb*. Valid digits for *a* are in the range from 0 to 3; either position *b* may be in the range from 0 to 7. Please note that an octal expression is recognized only if the backslash is followed by *three* valid digits! The expression "\1a7", for example, is interpreted as the quoted pair "\1" followed by the verbatim text "a7", because "a" is not valid for octal numbers.

\x*aa*

This expression is replaced by the value of the hexadecimal number $*aa*. Both positions *a* must be in the range from 0 to 9 or from "a" to "f". For the letters, either case is recognized, so "\xBB" and "\xbb" will yield the same result.

\x{...}

This expression denotes a set of grouped hexadecimal numbers. The `...` part may consist of an arbitrary number of hexadecimal pairs, such as in "\x{}", "\x{ff}", or "\x{55ffab04}". The empty expression "\x{}" is a no-op; it will not produce any output.

This construct may be useful to specify multi-byte characters (as in Unicode). "\x{0102}" is effectively equivalent to "\x01\x02", but the grouping of values may be useful in other contexts, even though for libvarexp it makes no difference.

8.4. Arrays of Variables

In addition to normal variables, libvarexp also supports arrays of variables. An array may only be accessed in a complex expression -- "$NAME[1]" is not correct syntax. Use "${NAME[1]}" instead. The reason for this limitation is that the brackets used to specify the index ("[" and "]") have a different meaning in ordinary text; see Section 8.5 for further discussion.

Which variables are arrays -- and which are not -- is entirely up to the application developer. In some applications, every variable may be accessed as both a normal variable and an array. In other applications, normal variables and arrays are different things. libvarexp does not dictate this. There exists the convention that accessing an array with a negative index, such as "${ARRAY[-1]}" should return the number of elements the array contains. But again, this is not a behavior required by libvarexp; different applications may behave differently here.

When specifying the index of the array's element you wish to access, you can use complete arithmetic expressions to calculate the entry. libvarexp supports the operands "+" (addition), "-" (subtractin), "*" (multiplication), "/" (division), and "+" (modulo).

These operations may be used on any signed integer. A valid expression is, for example: "${ARRAY[-12/4+5]}". Please note that libvarexp follows the usual operator precedence. To group expressions explicitely, put brackets around them: "${ARRAY[-12/(2+4)]}".

In any place you can write a number in such an expression, you can also use a simple or complex variable expression. If "$TWO" is "2", the following expression would access the 5th entry in the "$FOO" array: "${FOO[10/$TWO}".

8.5. Looping

Obviously, arithmetic in array indices would be quite pointless without a looping construct. libvarexp offers such a costruct, which can model both a "for" and a "while" loop. Let's start with the second version, which is slightly simpler.

If the index delimiters "[" and "[" are found in the text, the start a looping construct. An example would be "This is a test: [$FOO]". What happens now is that all text between the loop-delimiters is repeated again and again until all variables found in the body of the loop say they're undefined for the current index. The current index starts counting at zero (0) and is increased with every interation of the loop. In the index-specifier of the variable, it is available as "#".

Hence, if we assume that the variable "ARRAY[]" had three entries: "entry1", "entry2", and "entry3", then the loop "[${ARRAY[i]}]" would expand to "entry1entry2entry3". Once the conter reached index 4, "all" arrays in the loop's body are undefined.

That raises the question what the first example we presented, “This is a test: [$FOO]”, would expand to? The answer is: To the empty string! The loop would start expanding the body with index 0 and right at the very first iteration, all arrays in the body were empty -- that is, no array would have been expanded, because there weren’t any arrays.

Thus, this form of looping only makes sense if you *do* specify arrays in the loop’s body. If you do, though, you can do some weird things, like “[${ARRAY[#%2]}]”, which expands to “${ARRAY[0]}” for even numbers and to “${ARRAY[1]}” for odd numbers. But the expression has another property: It will never terminate, because the array-loopup will never fail, assuming that indices 0 and 1 are defined!

That is unfortunate but can’t be helped, I’m afraid. Users of libvarexp may choose to disable looping for the users of their application to prevent the end-user from shooting himself in the foot with infinite loops, though. But if you want to use loops, you must know what you’re doing. There ain’t no such thing as a free lunch, right?

There is another form of the looping construct available, that resembles a “for” loop more closely. In this form, the start value, the step value and the stop value of the loop can be specified explicitely like this: “[$FOO]{`start`,`step`,`stop`}”. This loop will start to expand the body using index `start`, it will increase the current index in each iteration by `step`, and it will terminate when the current index is greater than `stop`. (Please note that “greater than” is concept that needs much thought if you use negative values here! There may be some infinite loops coming. You have been warned.)

If any of the first two values are omitted, the following defaults will be assumed: `start` = 0 and `step` = 1. If `stop` is omitted, the loop will terminate if none of the arrays in the loop’s body is defined for the current index. Consequently, using the loop-limits “{,,}” is equivalent to not specifying any limits at all.

Since most users will not need the `step` parameter frequently, a shorter form “{`start`,`stop`}” is allowed, too.

By the way: Loops may be nested. :-)

To confuse the valued reader completely, let’s look at this final example. Assume that the arrays “${FOO[]}” and “${BAR[]}” have the following values:

```
FOO[0] = "foo0"
FOO[1] = "foo1"
FOO[2] = "foo2"
FOO[3] = "foo3"
```

and

```
BAR[0] = "bar0"
BAR[1] = "bar1"
```

Then the expression:

```
[${BAR[#]}: [${FOO[#]}${FOO[#+1]:+, }]${BAR[#+1]:+; }]
```

would expand to:

```
bar0: foo0, foo1, foo2, foo3; bar1: foo0, foo1, foo2, foo3
```

Have fun!

9. Expiring the Mail Spool

to be written

10. Importing Addresses From a Mail Archive

Of course it would be unpolite to have mapSoN send out requests for confirmation to people who you have been communicating with you for months or years, just because you installed a new tool. If you were wise enough to archive your old e-mails, there's a simple way to avoid that happening: Import their addresses into mapSoN's database.

Unfortunately, most mail readers archive old mails in one single file: Each new mail is just appended at the end, just like the mailbox format itself. Currently, mapSoN can not deal with those files. The current version can import addresses only from an archive where a each mail is stored in a separate file, like the archives maintained by the Gnus software, that is part of Emacs, for example.

In this case, though, it's simple enough: Just start mapSoN and give it the file names as parameters on the command line. You might want to enable debugging by giving it the `-d` flag, so that you can see what's going on:

```
simons@peti:~/mail-archive$ mapson -d *
1:
12:
    simons@peti.gmd.de................................ new
16:
    simons@peti.gmd.de................................ known
17:
    th@example.com..................................... new
    th@example.com..................................... known
    th@example.com..................................... known
19:
    bscw@cscwmail.example.org.......................... new
    manfred.bogen@gmd.example.org...................... new
53:
    pakhomenko@example.com............................. new
    pakhomenko@example.com............................. known
```

Depending on the size of your mail archive, this may take a while, but usually mapSoN is pretty quick.

Once that's finished, you'll have a pretty good database to start with, and it's highly unlikely that someone, who has been in contact with you before, will be bothered with an request-for-confirmation mail.

11. What To Do If Something Does Not Work

There's a chance that mapSoN isn't working the way you expect it -- especially in the current unfinished state of the program. Here's a short description of how you can probably figure what's going wrong. The magic word is "log file". mapSoN logs pretty much everything it does to a file, which is per default located at `$HOME/.mapson/log`.

A typical set of messages found there may look like this:

```
debug: mapSoN verion 2.0-beta-2 starting up
debug: My configuration:
```

```
debug:     Mailbox            = '/var/spool/mail/'
debug:     ConfigFile         = '/home/user/.mapson/config'
debug:     SpoolDir           = '/home/user/.mapson/spool'
debug:     AddressDB          = '/home/user/.mapson/address-db'
debug:     ReqConfirmTemplate = '/home/user/.mapson/reqmail.template: \
    /usr/local/share/mapson/reqmail.template'
debug:     MTA                = '/usr/sbin/sendmail '-f<>' -i -t'
debug:     StrictRFCParser    = 'false'
debug:     PassIncorrectMails = 'true'
debug:     RuntimeErrorRC     = '75'
debug:     SyntaxErrorRC      = '65'
debug:     Debug              = 'true'
error: Runtime error while processing mail 'no-message-id': \
    Can't open address db '/home/user/.mapson/address-db' \
    for reading: No such file or directory
```

Please note that the backslashes in this example are not actually there, they just denote added line breaks for the layout. In the real file, these split lines are just one one long line.

If you find that your copy of mapSoN does not log the proceedings in this amount of detail, set the `Debug` directive in the configuration file to *`yes`* or add the *`-d`* parameter to the command line when calling mapSoN.

By looking at the log file, you can see what exactly mapSoN is doing and why it's doing it. In the example shown above, it fails because of a file permission error.

Of course there are some reasons that may cause mapSoN to behave in a way different from what you execpted that are not directly connected to the mapSoN program itself. Here's a list of popular mistakes:

- Check whether the mailbox file mapSoN uses to deliver passed mails is correct! If it is not, you obviously won't see anything.
- Check whether mapSoN actually *sees* the incoming mails it is supposed to. Especially when you are using procmail to filter incoming e-mail, make sure that the confirmation mails are passed to mapSoN. You can debug what procmail is doing by adding the lines

  ```
  VERBOSE=on
  LOGFILE=$HOME/procmail.log
  ```

  to your `.procmailrc` file. Then look at procmail's log file.

A. The Default Configuration

The following configuration file shows the defaults compiled into mapSoN, so if your config file looks like this, you can as well erase it completely. The install process will install a sample configuration like this on your machine as `${sysconfdir}/config-sample`. If you didn't mess with the paths during the build, `${sysconfdir}` will be `/usr/local/share/mapson`.

```
Mailbox            ${MAILBOXDIR}/${USER}
SpoolDir           ${HOME}/.mapson/spool
AddressDB          ${HOME}/.mapson/address-db
```

```
AddressDBAutoAdd   true
ReqConfirmTemplate ${HOME}/.mapson/reqmail.template:${DATADIR}/reqmail.template
MTA                ${MTA} '-f<>' -i -t
PassIncorrectMails true
StrictRFCParser    false
RuntimeErrorRC     75
SyntaxErrorRC      65
LogFile            ${HOME}/.mapson/log
Debug              false
```

B. A Request-for-Confirmation Template Example

The following file will be installed on your machine as `${datadir}/reqmail.template-sample`. If you didn't mess with the paths during the build, `${datadir}` will be `/usr/local/share/mapson`.

```
From: ${USER} (${USER}'s Anti-Spam Tool)
To: ${ENVELOPE:-${RETURN_PATH:-${SENDER}}}
Subject: please confirm \[${MD5HASH}\]
Precedence: junk
Auto-Submitted: auto-generated
References: $MESSAGEID
In-Reply-To: $MESSAGEID

                        ----- English -----

Because I receive several dozen spam messages each day, I installed a
small tool that will defer incoming mail message if it comes from an
address it sees for the first time. This is the case with the message
you sent me, I'm afraid.

Before your message will be delivered to my mailbox, I need a
confirmation that your sender address is a valid e-mail address. You
can confirm this simply by replying to this message. Be sure _not_ to
erase the cookie found in the Subject: line whon doing so! Once my
tool sees your reply, it will deliver the original message to my
mailbox, and I'll answer it as soon as possible.

I am sorry that this is somewhat annoying for you, but the amount of
unsolicited commercial advertisement sent through the Internet these
days is to much for me to bear.

                        ----- Deutsch -----

Da ich taeglich dutzende Spam-Nachrichten erhalte, habe ich ein
Programm installiert, das alle eingehende Mail-Nachrichten abweist,
wenn diese von einer mir bisher unbekannten Absendeadresse kommt.
Leider ist dies bei Ihrer Nachricht der Fall.

Bevor mir die Nachricht tatsaechlich zugestellt wird, brauche ich eine
Bestaetigung, das die Absendeadresse "echt" ist. Sie koennen dies
```

```
einfach dadurch bestaetigen, dass Sie auf diese Nachricht antworten.
Bitte achten Sie dabei darauf, dass Sie den Zahlen- und Buchstabencode
in der Betreff-Zeile _nicht_ loeschen! Der Inhalt der Antwort spielt
dabei keine Rolle, es geht nur darum, dass das Programm sieht, dass
Ihre Adresse nicht gefaelscht war.

Ich bedauere, dass ich Ihnen damit Umstaende bereiten muss, aber die
Menge an unerwuenschter Werbung, die derzeit durch das Internet
geistert, war einfach zuviel.

     ----- Your message was: ---------- Ihre Nachricht war: -----

[ | ${HEADER[#]}] |
[${BODY[#]:+ | }${BODY[#]}]{0,5} | \[...\]
```

C. License