Smap – socket map utilities

version 2.1, 1 July 2021

Sergey Poznyakoff

Short Contents

Table of Contents

1 Introduction

When a Mail Transfer Agent (MTA) receives a message, it undertakes a sequence of steps to decide the fate of that particular message: whether to deliver it locally, to relay it to some other site, to reject or bounce it, or to take some other action. When taking its decision, MTA examines a set of data sources which hold such data as lists of local and relayed domains, tables of system accounts, etc. These data sources may be of various nature. For example, domain tables can be stored on disk as plaintext files or as DBM files; they can also be retrieved from LDAP or from some database management system. To provide a uniform access to such a variety of data sources, MTA usually implement some intermediate layer. Sendmail[1] and MeTA1[2] call this layer a *map*.

Among various types of maps implemented by these MTAs, there is one which merits special attention. It is *socket map*, also called *sockmap*, for short. This map is not associated with any particular data storage. When the MTA looks up for a key in a sockmap, the latter sends the request over TCP/IP to a preconfigured address, waits for a reply from there and hands it back to the MTA. It is supposed, of course, that some server is listening on this address.

Sockmaps provide an incredibly effective way of extending the functionality of MTAs. For example, one may use them to configure one's Sendmail to keep all data in an SQL database or in any other database, not directly supported by the MTA.

So far sockmaps have been given undeservedly little attention. Perhaps, this is due to lack of suitable free software servers that could be queried using them.

Smap aims to fill this gap. Its main component is `smapd` – a modular server which handles sockmap requests. Instead of handling each request itself, `smapd` relies on *loadable modules* to provide the requested functionality. In other words, `smapd` is responsible for handling socket map protocol, and for dispatching queries to appropriate modules. The module itself is responsible for looking up the requested key and returning the result.

Second important part of the package is a set of loadable modules for `smapd`. These modules cover several important database management systems and make it possible to easily configure servers for retrieving data from them.

Furthermore, the package provides a framework for writing new modules for `smapd`. New modules may be written either in C or in Guile.

And finally, Smap contains a client program, `smapc`, which may be used to query arbitrary socket servers from the command line. Among other possible uses, `smapc` is a valuable tool for testing your socket servers.

The main audience of Smap are administrators of Sendmail or MeTA1 mail transport agents, as well as those who use Mailfromd[3], a flexible general-purpose mail filter.

---

[1] See http://www.sendmail.org.

[2] See http://www.meta1.org.

[3] See mailfromd.software.gnu.org.ua.

2 Overview of the Smap Architecture

The Smap server consists of the following conceptual parts: `smapd` daemon, map modules and databases.

`smapd` *daemon*
: The `smapd` daemon is the principal part of the system. It is responsible for handling incoming connections and dispatching socket map requests to appropriate modules.

Map modules
: These are external loadable libraries which contain backend-specific lookup drivers. For example, one module may provide a driver for lookups in plain-text files, another one may handle DBM lookups and yet another – searches in MySQL databases. Notice, that modules provide abstract drivers, in the sense that they are not bound for look ups in particular disk files or databases. This specific information is supplied by Smap *databases*.

Databases
: A *database* is an intermediate logical entity associated with a particular module. The database provides actual configuration for the module. Several different databases may be associated with the same module, thereby creating several instances of the same lookup driver.

  If the underlying module allows for such use, a database may also be used to modify input map name and/or key value, before passing them on to another database.

The relationships between these parts are shown in the figure below:

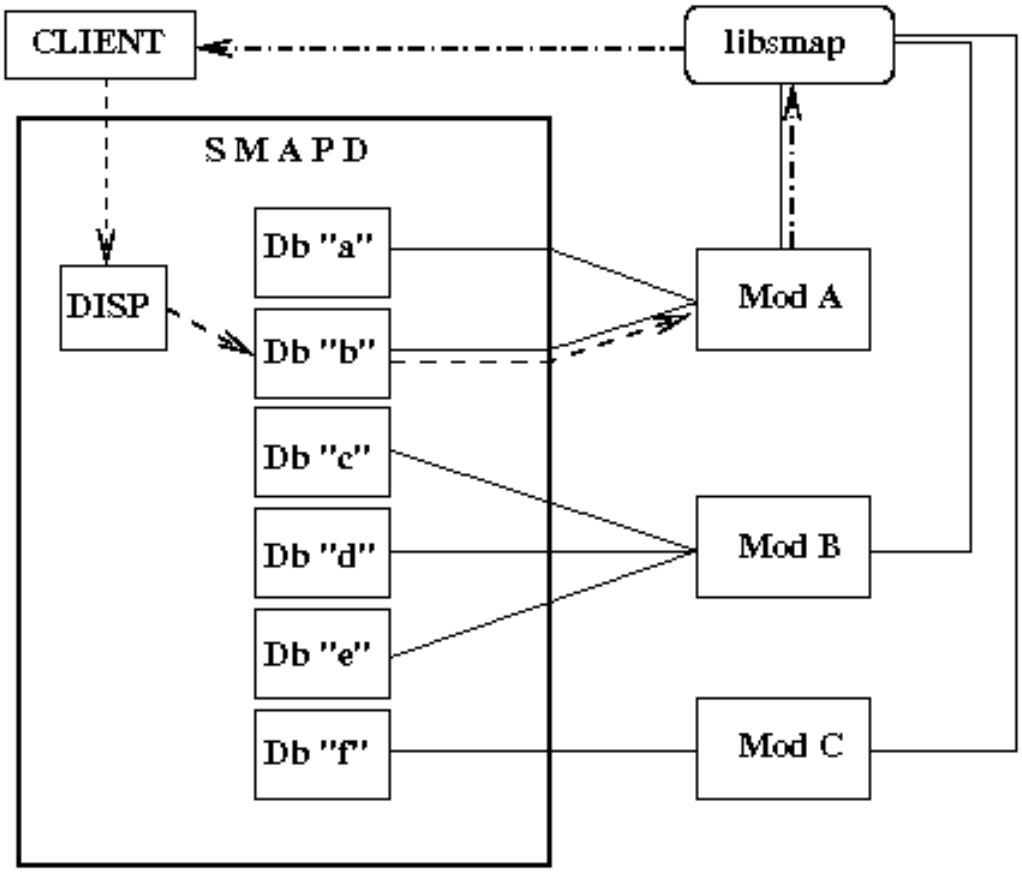

Figure 2.1: Smap Control Flow

Here, the `smapd` daemon is configured with six databases (shown as *Db a* through *Db f*), interfacing to three different modules (boxes *Mod A* through *Mod C*). The databases '`a`' and '`b`' interface to module '`A`', databases '`c`', '`d`' and '`e`' interface to module '`B`', and database '`f`' interfaces to module '`B`'. All three modules are linked with the `libsmap` library.

The box labeled '`CLIENT`' represents a client program. When `smapd` receives a request from client (its path is shown as a dashed line), it uses a set of *dispatch rules* (see the '`DISP`' box on the figure above) to dispatch it to the appropriate database. This database ('`Db b`', on the figure) is used to pass the request to the underlying module ('`Mod A`'). The module, after performing a look-up, sends the response back to the client (the dot-dashed line on the figure), using interface functions from `libsmap`. The latter is responsible for formatting the answer in accordance with the socket map protocol.

If the request matches no database, the server sends a default '`NOTFOUND`' reply back to the client.

Dispatch rules mentioned above are supplied by the user in the `smapd` configuration file. They resemble access control lists: each rule consists of a *condition* and *destination*. The condition may use various data from the connection and request itself, such as client IP address or map name from the request, and compare them with some static data. If the

condition yields true, the destination part of the rule points to the database which will handle this request.

3 The Socket Map Server

Socket map server `smapd` is the main part of the package. When invoked, it reads the *configuration file* and parses its command line to determine its configuration settings. Command line settings override ones from the configuration file. The default configuration file is `/etc/smapd.conf`[1] After that, `smapd` loads the requested modules and starts operation.

In this chapter we will describe the server operation in detail. The discussion below will often refer to *command line options* and *configuration statements*, so we'll first describe shortly what are those. The formal description will be given later.

Command line options have two forms. In *traditional*, or *short* form, an option is a letter prefixed by a dash (e.g. `-f`). In *long* form, an option consists of two dashes and option name (e.g. `--foreground`). Both option forms allow for an argument. For more information on option syntax, see Chapter 4 [smapd-options], page 17.

Configuration file uses the traditional UNIX syntax. Each statement occupies a single line. Very long lines may be split into several physical lines by ending each one with a backslash character. Comments are introduced with the '`#`' character: the character itself and everything after it up to next newline is ignored. For a detailed description, see Chapter 5 [smapd-config], page 19.

You can instruct `smapd` to read an alternative configuration file instead of the default one. It may be necessary, for example, to test a new configuration. To do so, use the `--config=`*`file`* (`-c` *`file`*) command line option. Its argument specifies the file name to read, e.g.:

```
$ smapd -c ./mysmapd.conf
```

To check whether your configuration is error-free, use the `--lint` (`-t`) option. It instructs `smapd` to parse the configuration file and exit after that. Any errors found are reported on the standard error. The exit code is '`0`' if the file parsed without errors and '`78`' otherwise (see Section 3.11 [exit codes], page 15, for a full list of exit codes). For example:

```
$ smapd -t
```

Of course, the two options may be used together:

```
$ smapd -t -c ./mysmapd.conf
```

or, in long form:

```
$ smapd --lint --config ./mysmapd.conf
```

3.1 Smapd Operation Modes

There are two modes of operation. In *standalone* mode, `smapd` detaches itself from the terminal and listens on incoming requests in background. In other words, it becomes a *daemon*. When a connection arrives, the server spawns a copy of itself (called *child process*) to handle it. Thus, a number of incoming connections are handled in parallel. This is the default mode.

[1] To be precise, it is *`sysconfdir`*`/smapd.conf`, where *sysconfdir* is the name of *system configuration directory*, determined when configuring the package. The two most frequently used values for it are `/etc` and `/usr/local/etc`.

In *inetd* mode, `smapd` does not listen on network addresses nor becomes a daemon. Instead, it reads requests from its standard input and sends replies on its standard output. As its name implies, this mode is intended for use from the `inetd.conf` file.

The inetd mode is requested from command line using the `--inetd` (`-i`) option, or from configuration file, using '`inet-mode yes`' statement.

3.2 Logging

The server determines automatically where its diagnostics output should go. By default, it goes to standard error. However, after detaching from the terminal in standalone mode, `smapd` sends diagnostics to syslog, using facility '`daemon`' by default. The same applies also if its standard input is not attached to a terminal.

Two command line options are provided to manually alter these defaults. The `--stderr` (`-e`) option instructs `smapd` to always send its diagnostics to the standard error, In contrast, the `--syslog` (`-l`) option forces it to use syslog.

The log facility can be changed in configuration file, using the '`log-facility`' statement (see [conf-log-facility], page 20), or in the command line, using the `--log-facility` (`-F`) option. In both cases, the argument is the facility number or symbolic name. Valid names are: '`user`', '`daemon`', '`auth`', '`authpriv`', '`mail`', '`cron`', and '`local0`' through '`local7`'.

Similarly, the log tag can also be changed, either from the configuration file, using the '`log-tag`' statement, or from the command line, using the `--log-tag` (`-L`) option,

3.3 Tracing and Debugging

The amount of information logged by the server is configurable. By default, it is quite silent and outputs only diagnostics that call to special attention, such as errors and the like. You may request more information, however. For further discussion, it is convenient to introduce two main information groups: query traces and debugging information. *Query traces* are log messages that show received queries and corresponding replies. They look like:

```
user bar => NOTFOUND
access connect:111.67.206.187 => OK REJECT
```

The part to the left of the '`=>`' sign shows the query exactly as received from the client, i.e. the first word is the map name, and the rest of words constitute the key. The part to the right of '`=>`' is the reply to this query.

To enable query traces, use the `--trace` (`-T`) command line option or '`trace yes`' statement in the configuration file.

When using syslog, query traces are reported using the '`info`' priority.

Some requests may be of particular interest to you, whereas others may not be relevant at all. There is a way to abridge the traces to show those relevant requests only. If you give the `--trace-pattern=`*`pattern`* option, only those requests that begin with *pattern*[2] will be shown. For example, to show only positive responses, use

[2] Actually, the argument would better be named *prefix*, but I plan to implement globbing patterns (or maybe even regular expressions) in future versions, so I refer to it as *pattern* in anticipation.

```
--trace --trace-pattern=OK
```

The same can be requested in the configuration file as well:

```
trace yes
trace-pattern OK
```

Any number of `--trace-pattern` options (or configuration statements) may be given. The server will log only those queries that match one of the patterns specified by them.

Debugging information is auxiliary diagnostics reflecting various details of internal functionality of `smapd`. Although aimed primarily to help in debugging the server, it may occasionally be of use for server administrators as well.

Debugging information is requested using the `--debug` (`-d`) command line option or '`debug`' configuration statement. In both cases, the argument is a *debug specification*, consisting of two parts, separated by a dot: '`cat.lev`'. The *cat* part is a *debug category*. It is either an integer number identifying the category, or its symbolic names. For a list of categories and their meaning, see Appendix C [Debug Categories], page 57.

The *lev* part is the category *level*, an integer specifying how much verbosity is requested from that category. The '`0`' value means no verbosity (i.e. to disable that category), the value of '`100`' means maximum verbosity. The convention is that levels below '`10`' may be of occasional use for sysadmins, whereas higher values are useful only for debugging.

To enable several debug categories, use several `--debug` option (or '`debug`' configuration statements).

3.4 Runtime Privileges

By default `smapd` runs with the privileges of the user that started it. Normally, this user is root. If you wish it to switch to some unprivileged user after startup, use the `user` configuration statement:

```
user daemon
```

The above example instructs `smapd` to switch to the UID of the user '`daemon`' and to the GID of its principal group. The rest of groups the user might be a member of is dropped. To retain all supplementary user groups, use the `allgroup` statement. Its argument is a *boolean value*, i.e. '`yes`', '`on`', '`true`', or '`t`' to indicate the *true value*, and '`no`', '`off`', '`false`' or '`nil`' to indicate *false*. So, to switch to the user '`daemon`' and also retain all its supplementary groups, one would write:

```
user daemon
allgroups yes
```

You may also retain only some of the user's group, or even some groups the user is not member of. This is done using the `group` statement:

```
user daemon
group mail mysql
```

Arguments to `group` are any number of valid group names.

Notice, that while running `smapd` with non-root privileges might be a good idea, it may render some modules useless. For example, the `mailutils` module in '`mbq`' mode (see Section 6.2.5 [mbq], page 28) requires root privileges for normal operation. To allow for such uses, instead of setting global user privileges, set them on a per-server basis. See Section 3.5 [servers], page 10, for a detailed discussion of this technique.

3.5 Server Configuration

Servers are internal `smapd` objects, responsible for listening on sockets and handling socket I/O operations. Each server has a *server id*, which is a unique name associated with it, and *socket address*, which describes the socket this server handles.

Socket addresses are represented as *URL*s. Smap version 2.1 recognizes the following URL forms:

inet://*ip*:*port*
: Listen on the IPv4 address *ip*, on the given *port*. IP address may be given either in "dotted-quad" notation or as a hostname. Port may be specified either as a port number, or as a name of a service from `/etc/services`.

unix://*pathname*
: Listen on the UNIX socket *pathname*. Notice that the name of the socket must be absolute, so you would usually have three slashes running together, e.g. the notation

  ```
  unix:///var/run/smap.sock
  ```

  means UNIX socket `/var/run/smap.sock`.

The `server` statement configures servers. It takes two mandatory arguments: the socked ID and URL, e.g.:

```
server main inet://10.10.1.11:3056
server local unix:///var/run/smap.sock
```

These statements configure two servers. The one called '`main`' is listening on IP 10.10.1.11, port 3056. The one called '`local`' listens on UNIX socket `/var/run/smap.sock`.

If a server is assigned an '`inet`' address, access to it will be controlled by TCP wrappers. The server ID is used as *daemon name*. See the next section (see Section 3.6 [TCP wrappers], page 12) for a detailed description.

The `server` statement has also another form, called *block form*, which allows to configure more details. In this form, the statement is given third argument – the word '`begin`'. This statement is followed by one or more statements supplying additional configuration for this server, followed by the word '`end`' on a line by itself, which closes the construct. This is illustrated in the following example:

```
server local unix:///var/run/smap.sock begin
  backlog 10
  user mail
end
```

Statements which may be used between '`begin`' and '`end`' fall into two categories: privilege control statements, and socket configuration statements.

The former are `user`, `allgroups` and `group`, described in the previous section (see Section 3.4 [privileges], page 9). Syntactically they are exactly the same as their public scope counterparts. The only difference is that they apply only to child processes spawned to handle connections to that particular URL. For example, consider the following statement:

```
server local unix:///var/run/smap.sock begin
  user daemon
  group mail mysql
```

```
end
```

This configuration works as follows. The master `smapd` process runs with root privileges. When a connection is requested to socket `/var/run/smap.sock`, the master spawns a subprocess to handle that connection. This subprocess switches to the UID and GID of user '`daemon`' and retains GIDs of the groups '`mail`' and '`mysql`' and then enters the mail read-and-reply loop. The ownership of the socket `/var/run/smap.sock` is set to UID of user '`daemon`' and GID of its primary group (see also the description of `owner`, below).

Of course, the per-server privilege control statements work only if the master daemon runs with the root privileges.

The second group of server statements are socket configuration statements. Similarly to privilege control statements, these too may appear inside a server block statement as well as outside of it, in the global scope (with the exception of the `owner` statement, which is allowed only in `server` scope). When used in global scope, they affect all `server` statements. When used in per-server context, they apply to that particular server only. These statements are:

`backlog` *number* [Config]

Sets the maximum size of pending connections queue for sockets. If a connection request arrives when the queue is full, the client receives an error with an indication of '`ECONNREFUSED`'.

Default backlog is 8.

`reuseaddr` *bool* [Config]

If *bool* is '`yes`' reuse existing socket addresses (both INET and UNIX). This is the default.

`max-children` *number* [Config]

Maximum number of children processes allowed to run simultaneously. When the actual number of children reaches *number*, the server stops refusing further connections until any of them terminates. The value of '`0`' means '`unlimited`'.

The default limit is '`128`'.

`single-process` *bool* [Config]

Operate in single-process mode. This options may become necessary only when debugging the `smapd` daemon. *Never use it in production environment!*

`socket-mode` *mode* [Config]

Set file mode for UNIX socket. Specify the *mode* argument either int octal notation (e.g. '`600`'), or in `chmod`-style notation (e.g. '`rw-------`').

`socket-owner` *user:group* [Config]

Set socket ownership to the given user and group. This applies only to UNIX sockets. User and group may be specified either by their symbolic names or numeric IDs. Either *user* or *group* may be omitted. There are following cases:

owner *user*:*group*

Set both owner UID and GID.

owner *user*

Set UID of the user *user* and GID of his primary group.

owner *user*:
: Set UID of the user *user*, but do not change the GID.

owner :*group*
: Set only owner GID, do not change the UID.

Note, that this statement cannot be used outside of `server` scope.

3.6 TCP Wrappers

Access to servers having addresses in '`INET`' family is controlled using *TCP wrappers*[3].

This system is based on two files, called *tables*, containing access rules. There are two tables: the *allow table*, stored in file `/etc/hosts.allow`, and the *deny table*, kept in file `/etc/hosts.deny`. The rules in each table begin with an identifier called *daemon name*. Access to a Smap server is controlled by two entries: a *global one*, with the daemon name '`smapd`', and per-server one, with server ID (see Section 3.5 [servers], page 10 as its daemon name. The latter takes precedence over the former. For example, if you have the following in your `smapd.conf`:

```
server main inet://192.168.10.1
```

and wish this server to be accessible only to machines 192.168.10.2 and 192.168.10.3, then you would add the following line to your `/etc/hosts.allow`:

```
main: 192.168.10.2 192.168.10.3
```

and the following line to your `/etc/hosts.deny`:

```
main: ALL
```

The former allows access from these two IPs, and the latter blocks it from any other IPs.

A detailed description of TCP wrapper table format lies outside the scope of this document. Please, see Section "ACCESS CONTROL FILES" in *hosts_access(5) man page*, for details.

3.7 Loadable Modules

Mapper modules are external pieces of software designed to handle a particular subset of map queries. They are built as shared libraries and loaded into `smapd` at startup.

Modules are loaded using the `module` command:

`module` *module-id module-name* [*args*] [Config]
: Load module `module-name`. Additional arguments (*args*), if specified, are passed to the module initialization function.

  The *module-id* is a unique identifier, which will subsequently be used to refer to that module.

A *module load path* is an internal list of directories which `smapd` scans in order to find a loadable file name specified in `module` statement. By default the scan order is as follows:

1. Additional search directories specified by `prepend-load-path` (see below);

[3] This feature requires `smapd` to be compiled with the TCP wrappers library `libwrap`. It is always enabled at configure time, unless `libwrap` is absent, or you explicitly disable it.

2. Smap module directory: `$prefix/lib/smap`;
3. Additional search directories specified by `append-load-path` (see below);
4. Directories specified in the environment variable `LTDL_LIBRARY_PATH`.
5. The system dependent library search path (e.g. on GNU/Linux it is set by the file `/etc/ld.so.conf` and the environment variable `LD_LIBRARY_PATH`).

Values of `LTDL_LIBRARY_PATH` and `LD_LIBRARY_PATH` must be colon-separated lists of absolute directory names, for example: '`/usr/lib/mypkg:/lib/foo`'.

In any of these directories, `smapd` first attempts to find and the given *module-name* verbatim and to load it. If this fails, it tries to append the following suffixes to it:

1. the libtool archive suffix '`.la`'
2. the suffix used for native dynamic libraries on the host platform, e.g.: '`.so`', '`.sl`', etc.

Additional search directories may be configured with `prepend-load-path` and `append-load-path` statements:

`prepend-load-path` *path* [Config]
: Prepends the directories listed in its argument to the module load path. The *path* argument must be a colon-separated list of absolute directory names.

`append-load-path` *path* [Config]
`load-path` *path* [Config]
: Appends the directories listed in its argument to the module load path. The *path* argument must be a colon-separated list of absolute directory names.

3.8 Databases

A *database* is a logical entity associated with a particular module, that provides a specific configuration for it. In other words, database is a configured instance of the module.

Databases are declared using the following statement:

`database` *dbname modname* [*args*] [Config]
: Declare database *dbname* as an instance of module *modname*. This module should have been declared previously using the `module` statement (see Section 3.7 [loadable modules], page 12). Optional *args* provide configuration information for the module initialization function. They are module-specific.

To illustrate this, let's consider the '`echo`' module, which replies to any request with a constant string supplied to it as arguments (see Section 6.1 [echo], page 25). The following example configures two instances of this module:

```
database nomap echo NOTFOUND No such map
database tempfail echo TEMP Try again later
```

The '`nomap`' database always sends the string '`NOTFOUND No such map`' in reply. The '`tempfail`' database replies with the string '`TEMP Try again later`'.

3.9 Query Dispatch Rules

When a query arrives, `smapd` uses *query dispatch rules* to decide to what database to dispatch this query. Dispatch rules are somewhat similar to ACLs: each rule consists of a set of conditions and a target part. The rules are joined in a list. When applied to a particular query, this list is scanned from top down. The conditions of each rule are evaluated using the query as their argument. If all conditions return '`True`', then the target part of this rule is applied. The target part may either *transform* the map name and/or key value (a *transformation rule*), or indicate a database to dispatch this query to (a *destination rule*). After applying a transformation rule, the scanning resumes at the next rule. Destination rules end the processing.

If the list is exhausted without having found a matching destination rule, `smapd` sends back the default '`NOTFOUND`' reply.

Consider for example the following rule:

```
dispatch map eq alias database maildb
```

It says that if the map part of a query is the word '`alias`', then this query must be handled by the database '`maildb`'.

The `map` condition allows for more sophisticated comparisons. If you use '`like`', instead of '`eq`', than shell-style globbing patterns are used. For example, this rule

```
dispatch map like us* database user
```

matches queries whose map part begins with '`us`'.

Finally, you may also use regular expressions:

```
dispatch map regexp /(alias)|(virtusers)/ database maildb
```

See [cond-map], page 23, for a detailed description of this condition.

Another important condition is `from`. It returns '`True`' if its argument, which is an IP address or a CIDR, matches the IP of the machine that sent the query. For example, the following rule directs all queries coming from IP addresses 192.168.0.1 through 192.168.0.31 to the database '`local`':

```
dispatch from 192.168.0.0/27 database local
```

Several conditions may be used together. The result is '`True`' if all conditions yield '`True`'. For example:

```
dispatch from 192.168.0.0/27 \
         map regexp /^(alias)|(virtuser)$/ \
         database local-maildb
```

This rule dispatches to the database '`local-maildb`' all queries coming from the network 192.168.0.0/27 and having '`alias`' or '`virtuser`' as their map part.

The `server` condition is often used together with `from`. Its argument is the id of a server (see Section 3.5 [servers], page 10) declared in the configuration. The condition returns '`True`' if the query was sent to that particular server. For example:

```
dispatch from 192.168.0.0/27 \
         server privileged
         database secret
dispatch from 192.168.0.0/27 database public
```

These rules dispatch to the database '`secret`' any queries coming from IP address in network 192.168.0.0/27 and received by the server '`privileged`'. Queries from that network accepted by another servers are dispatched to the database '`public`'. It is, of course, supposed that somewhere in the configuration file there is a declaration, that looks like

```
server privileged inet://192.168.0.1:3145
```

The result of any condition may be reverted using the '`not`' prefix before it, e.g.:

```
dispatch from 192.168.0.0/27 \
         not map regexp /^(alias)|(virtuser)$/ \
         database user
```

There is a special condition which is convenient for the last rule in the list. The '`default`' condition always returns '`True`', so this rule:

```
dispatch default database nomap
```

will match any rule and dispatch it to a database named '`nomap`'. The '`default`' condition cannot be combined with other conditions.

3.10 Transformations

Transformations are special rules that modify the key or map value. Syntax of transformation rules is:

```
dispatch cond-list transform key-or-map dbname
```

where *cond-list* is a condition list, as described in the previous section, *key-or-map* is '`key`' if the transformation is applied to the key value and '`map`' if it is applied to the map name, and *dbname* is the name of a database that handles the transformation. For example:

```
dispatch key like <*> transform key dequote
```

This rule applies the '`dequote`' database to any key that is enclosed in angle brackets. It is supposed that the '`dequote`' database removes the brackets. It may be implemented using the the '`sed`' module (see Section 6.7 [sed], page 39), as follows.

```
module sed sed
database dequote sed extended 's/<(.*)>/\1/g'
```

The transform rules can be chained, as in the example below:

```
# This database removes domain part from its argument.
database localpart sed 's/@.*$//'

# Dispatch rules:
dispatch key like <*> transform key dequote
dispatch key like *@* transform key localpart
dispatch default database getpwnam
```

As a result, the '`getpwnam`' database will get the local part of the original key (which may be supplied in the form of an email address).

3.11 Smapd Exit Codes

The following table summarizes exit codes used by `smapd`. For each code it lists its decimal number, symbolic name from the `sysexits.h` header file, and its meaning.

Code	Name	Meaning
0	EX_OK	Normal termination.
64	EX_USAGE	Command line usage error.
69	EX_UNAVAILABLE	Some other error occurred.
78	EX_CONFIG	Errors in configuration file detected.

4 Command Line Syntax

Most command line options have two forms, called short and long forms. Both forms are absolutely identical in function; they are interchangeable.

The *short* form is a traditional form for UNIX utilities. In this form, the option consists of a single dash, followed by a single letter, e.g. `-c`.

Short options which require arguments take their arguments immediately following the option letter, optionally separated by white space. For example, you might write `-f name`, or `-fname`. Here, `-f` is the option, and `name` is its argument.

Short options' letters may be clumped together, but you are not required to do this. When short options are clumped as a set, use one (single) dash for them all, e.g. `-cvl` is equivalent to `-c -v -l`. However, only options that do not take arguments may be clustered this way. If an option takes an argument, it can only be the last option in such a cluster, otherwise it would be impossible to specify the argument for it. Anyway, it is much more readable to specify such options separated.

The *long* option names are probably easier to memorize than their short counterparts. They consist of two dashes, followed by a multi-letter option name, which is usually selected to be a mnemonics for the operation it requests. For example, `--verbose` is a long option that increases the verbosity of a utility. In addition, long option names can abbreviated, provided that such an abbreviation is unique among the options understood by a given utility. For example, if a utility takes options `--foreground` and `--forward`, then the shortest possible abbreviations for these options are `--fore` and `--forw`, correspondingly. If you try to use `--for`, the utility will abort and inform you that the abbreviation you use is ambiguous, so it is not clear which of the options you intended to use.

Long options which require arguments take those arguments following the option name. There are two ways of specifying a mandatory argument. It can be separated from the option name either by an equal sign, or by any amount of white space characters. For example, if the `--file` option requires an argument, and you wish to supply `name` as its argument, then you can do so using any of the following notations: `--file=name` or `--file name`.

The following table summarizes the options available for `smapd`. For each option a brief description is given and a cross reference is provided to more in-depth explanation in the body of the manual.

`-c` *file*
`--config=`*file*
: Read configuration from *file*, instead of the default `/etc/smapd.conf`. See Chapter 3 [–config], page 7.

`-t`
`--lint`
: Test configuration and exit with code '`0`' if the file parsed without errors and '`78`' otherwise. Any errors found are reported on the standard error. See Chapter 3 [–lint], page 7.

`-f`
`--foreground`
: Do not detach from the controlling terminal, operate in foreground.

`-e`
`--stderr` Output diagnostic to stderr. See Section 3.2 [logging], page 8.

`-l`
`--syslog` Output diagnostic to syslog (default). See Section 3.2 [logging], page 8.

`-s`
`--single-process`
Operate in single-process mode. This option is intended to help in debugging `smapd`. *Do not use it in production environment!*

`-i`
`--inetd` Operate in inetd mode (see [inetd-mode], page 7).

`-T`
`--trace` Trace queries and replies. See Section 3.3 [Query traces], page 8.

`-p` *`pattern`*
`--trace-pattern=`*`pattern`*
Trace only queries that begin with the given *pattern*. See [trace-pattern], page 8.

`-d` *`level`*

`-x` *`level`*
`--debug=`*`level`*
Set debug verbosity level. See Section 3.3 [Debugging information], page 8. The `-x` alias is for compatibility with version 1.0 and will be removed in subsequent releases.

`-L`
`--log-tag=`*`tag`*
Set syslog tag. See Section 3.2 [logging], page 8.

`-F` *`facility`*
`--log-facility=`*`facility`*
Set syslog facility. See Section 3.2 [log-facility], page 8.

`-h`
`--help` Give a concise summary of the command line options.

`--usage` Give a short usage reminder.

`-V`
`--version`
Print program version.

5 Smapd Configuration File

The `smapd` configuration file consists, on a lexical level, of logical lines. A *logical line* is any sequence of characters between two unescaped newline characters. The word '`unescaped`' means a newline character not preceded by a single backslash. Thus, escaped newlines allow to combine several physical lines into a single logical one.

Within a logical line, unescaped '`#`' character introduces a comment. The character itself and the rest of characters after it up to the end of line are ignored.

Empty lines are ignored as well.

Each not empty line constitutes a *configuration statement*. Before further processing the statement is subject to the following *expansions*:

variable substitution
: Variable substitution consists in replacing each sequence '`$name`' or '`${name}`' with the value of the *variable name*. Valid variable names begin with a letter of the Latin alphabet or underscore and consist of alphanumeric symbols and underscores. Variable names are case-sensitive. Variables are expanded in unquoted and doubly-quoted arguments. Variable expansion is suppressed within single-quoted strings (see below).

field splitting
: A *word* is defined as any contiguous sequence of non-whitespace characters or any sequence of characters enclosed in double or single quotes. Standalone words and doubly-quoted strings are subject to variable substitution and escape expansion.

escape expansion
: A backslash character introduces an *escape sequence*. The following escape sequences are expanded:

Sequence	Replaced with
\a	Audible bell character (ASCII 7)
\b	Backspace character (ASCII 8)
\f	Form-feed character (ASCII 12)
\n	Newline character (ASCII 10)
\r	Carriage return character (ASCII 13)
\t	Horizontal tabulation character (ASCII 9)
\v	Vertical tabulation character (ASCII 11)

Table 5.1: Escape sequences

A '\' followed by any character not listed in the table above is replaced with that character alone. This allows, for example, to include double-quote characters in a doubly-quoted string.

quote removal
: This stage consists in removing unescaped single and double quotes, which where not inserted due to variable expansion.

If, after expansion, the statement consists of a single word that begins with a valid variable name immediately followed by an equals sign, such statement is treated as a *variable*

assignment. The string to the right of the equals sign is assigned to the variable named to the left of it.

Otherwise, if the statement has two or more words, the first word is treated as a *keyword*, which identifies a configuration statement, and the rest of words as its arguments.

The following configuration statements are understood.

`inetd-mode` *bool* [Config]
: If *bool* is '`yes`', enable inet mode (see [inetd-mode], page 7).

`pidfile` *filename* [Config]
: Write pidfile to the file *filename*.

`foreground` *bool* [Config]
: If *bool* is '`yes`', run in foreground. This also means that log output goes to standard error, unless requested otherwise by '`log-to-syslog`' statement or `--syslog` command line option.

`idle-timeout` *number* [Config]
: Sets *idle timeout* to *number* seconds. A child process terminates if it has not received any request within this amount of time.

`log-to-stderr` *bool* [Config]
: If *bool* is '`yes`' send log output to standard error.

`log-to-syslog` *bool* [Config]
: If *bool* is '`yes`' send log output to syslog.

`log-tag` *string* [Config]
: Tag each log line in syslog with *string*. By default, the name of the program ('`smapd`') is used.

`log-facility` *fac* [Config]
: Write logs to the syslog facility *fac*. Valid values for *fac* are: '`user`', '`daemon`', '`auth`', '`authpriv`', '`mail`', '`cron`', and '`local0`' through '`local7`'.

  Default is '`daemon`'.

`debug` *dspec1* [*dspec2...*] [Config]
: Enable debugging output according to the given specifications. See Section 3.3 [debugging], page 8, for a description of specifications.

`trace` *bool* [Config]
: If *bool* is '`yes`' enable query traces. See Section 3.3 [debugging], page 8.

`trace-pattern` *pat1* [*pat2...*] [Config]
: Abridge query trace output to queries beginning with the given patterns. See [trace-pattern], page 8.

`user` *name* [Config]
: After startup, switch to UID and GID of the user *name*.

`group` *name1* [*name2 ...*] [Config]
When switching to user privileges (see above), retain also these supplementary groups.

`allgroups` *bool* [Config]
When switching to user privileges (see above), retain all supplementary groups the user is a member of.

`socket-mode` *mode* [Config]
Set default file mode for creating UNIX sockets. The *mode* argument must be either in octal notation (e.g. '`600`'), or in `chmod`-style notation (e.g. '`rw-------`').

Default mode is '`600`'.

`shutdown-timeout` *seconds* [Config]
Sets the number of seconds to wait for all children to terminate before shutdown, after sending them the '`SIGTERM`' signal. Any children remaining active after this timeout are terminated forcefully using '`SIGKILL`'.

Default value is 5 seconds.

`backlog` *number* [Config]
Sets the maximum size of pending connections queue for sockets. If a connection request arrives when the queue is full, the client receives an error with an indication of '`ECONNREFUSED`'.

Default backlog is 8.

`reuseaddr` *bool* [Config]
If *bool* is '`yes`' reuse existing socket addresses (both INET and UNIX). This is the default.

`max-children` *number* [Config]
Maximum number of children processes allowed to run simultaneously. When the actual number of children reaches *number*, the server stops refusing further connections until any of them terminates. The value of '`0`' means '`unlimited`'.

The default limit is '`128`'.

`single-process` *bool* [Config]
Operate in single-process mode. This option may be necessary only when debugging `smapd`. *Never use it in production environment!*

`server` *name address* [*block*] [Config]
Configure a server. The *name* argument gives its symbolic name, which will be used in logs to identify it. The *address* argument specifies network address to listen on. As of version 2.1 two kind of addresses are recognized:

inet://*ip*:*port*
Listen on the IPv4 address *ip*, on the given *port*. IP address may be given either in "dotted-quad" form or as a hostname. Port may be specified either as a port number, or as a name of a service from `/etc/services`.

unix://*pathname*

Listen on the UNIX socket *pathname*. Notice that the name of the socket must be absolute, so you would usually have three slashes together. For example, the following statement will listen on a socket named `/var/run/smap.sock`:

```
server main unix:///var/run/smap.sock
```

Optional *block* is a *block statement* consisting of the word '`begin`' followed by a newline, one or more configuration statements and the word '`end`' alone on a line. For example:

```
server main unix:///var/run/smap.sock begin
  user smap
  allgroups yes
end
```

The statements within block apply only to that particular server. That is, in the example above, the connections requested on the server *main* will be handled by a subprocess with privileges of the user *smap*, retaining all the supplementary groups of this user. The following statements are allowed for use in the block statement:

- allgroups
- backlog
- group
- max-children
- reuseaddr
- single-process
- user
- socket-mode
- socket-owner

Their meaning is the same as of the corresponding statements in global scope (see above), but applies to that particular server only.

`load-path` *path* [Config]

Add *path* to the current set of directories searched for module files. *Path* is a list of directory names separated by colons.

`module` *modname libname* [*args...*] [Config]

Declare new module. Arguments are:

modname A name which uniquely identifies this module in the configuration. It will be used to associate databases with this module.

libname Name of the shared library file (without suffix) to load.

args... Arguments to the module initialization function.

`database` *dbname modname* [*args...*] [Config]

Define a database *dbname* and associate it with the module *modname*, which must be loaded by a prior `module` statement. Optional *args* are passed to the database initialization function verbatim.

`dispatch` *cond target* [Config]

Dispatch incoming queries.

Cond is a list of conditions that must be satisfied in order to dispatch this query to *target*. Conditions are separated by any amount of whitespace. They are evaluated from left to right and are joined using boolean '`AND`' so that *cond* yields '`True`' only if all conditions evaluate to '`True`'. Supported conditions are:

`from` *ipaddr* [Condition]

Returns '`True`' if the IP address of the client equals *ipaddr*. The latter may be given either as an IP address or as a host name, in which case it will be resolved and the first of its IP addresses will be used.

`from` *ipaddr/netmask* [Condition]

Returns '`True`' if the result of logical '`AND`' between the client IP address and *netmask* equals to *ipaddr*. The network mask must be specified in "dotted quad" form, e.g.:

```
from 10.1.10.1/255.255.255.224
```

`from` *ipaddr/netlen* [Condition]

Returns '`True`' if first *netlen* bits from the client IP address equal to *ipaddr*. The network mask length, *netlen* must be an integer number in the range from 0 to 32. The address part, *ipaddr*, is as described above. For example:

```
from 10.1.10.1/27
```

`server` *name* [Condition]

'`True`' if this query is being served by server *name* (see [config-server], page 21).

`map` *op string* [Condition]

'`True`' if the map name part of the query (see Appendix B [Protocol], page 55) matches *string*. The *op* part specifies the comparison algorithm:

eq
is
: Literal equality. Map name must be exactly the same as *string*.

like
fnmatch
: Match using shell wildcard patterns (see Section "glob" in *Glob(7) man page*).

regexp
: Match using regular expressions. *String* must have the following form:

```
/expr/flags
```

The slashes may be uniformly replaced with any other punctuation character. The *expr* must constitute a valid regular expression. The *flags* are optional. When given, they allow to control the type of the regular expression:

Flag	**Meaning**
i	Use case-insensitive matching
x	*expr* is an *extended regular expression*. This is the default setting.

b *expr* is a *basic regular expression*.

See Section "Extended regular expressions" in *GNU sed*, for a description of Extended regular expressions.

`key` *op string* [Condition]

'`True`' if the key value (see Appendix B [Protocol], page 55) matches *string*. The *op* argument has the same meaning as for `map` above.

`not` *cond* [Condition]

Reverts the value returned by *cond*, which is one of the conditions described above. For example:

```
not map like "local*"
```

`default` [Condition]

Always '`True`'. This must be the only condition in *cond*. It is useful to declare default query destination.

The *target* instructs the server to direct this query to a particular database. The syntax is:

`database` *dbname* [Target]

Pass this query to the database *dbname* (see [config-database], page 22).

6 Modules Shipped with Smap

Smap is shipped with a set of loadable modules, which are installed in its default module directory, *$prefix*`/lib/smap`. The modules are configurable on a per-module (see Section 3.7 [loadable modules], page 12), and per-database (see Section 3.8 [databases], page 13) levels.

Smap version 2.1 is shipped with several modules, which are described in detail in the following sections.

6.1 Echo

The echo module is the simplest of all modules. It sends back a static reply string, no matter what the query was. This module is useful for default databases, which catch erroneous or not handled queries.

Loading

The module needs no additional arguments for initialization. Normal loading statement is:

```
module echo echo
```

Database

Database initialization function treats its arguments as a string to be sent in reply to all queries. An example database definition:

```
database default echo NOTFOUND [no such map]
```

Such a definition is normally used as a target of the '`default`' dispatch rule:

```
dispatch default database default
```

6.2 Mailutils

This module uses GNU Mailutils (`http://www.gnu.org/software/mailutils`) and provides two main modes:

'`auth`' This mode uses GNU Mailutils authorization mechanism to obtain user data (similar to the system '`getpwnam`' routine) and returns positive reply if the data were retrieved and negative reply otherwise. See Appendix A [MeTA1], page 51, for an example on how to use it as a local user and alias database.

'`mbq`' This mode allows to check whether the user's mailbox exceeded the allotted quota, and if not, whether it is able to accept a message of the given size without exceeding it. The mode name is an abbreviation of *Mailbox Quota*.

6.2.1 Variable Expansion

In the discussion below we often refer to *meta-variable expansion* in strings. This is a process, whereby any sequence '`${`*variable*`}`' is replaced with the value of *variable*. The defined variables are:

db The database name.

map The map name.

key The lookup key.

diag — If the key was not found or some error occurred, this variable expands to a short diagnostics string, suitable for return message. Otherwise, expands to empty string.

name — The '`name`' field from the retrieved record. Empty string if the user not found.

passwd — The '`passwd`' field from the retrieved record. Empty string if the user not found.

uid — The '`uid`' field from the retrieved record. If the user was not found, expands to '`-1`'.

gid — The '`gid`' field from the retrieved record. If the user was not found, expands to '`-1`'.

gecos — The '`gecos`' field from the retrieved record. Empty string if the user not found.

dir — The '`dir`' field from the retrieved record. Empty string if the user not found.

shell — The '`shell`' field from the retrieved record. Empty string if the user not found.

mailbox — The '`mailbox`' field from the retrieved record. Empty string if the user not found.

quota — The '`quota`' field from the retrieved record. If the user was not found, expands to '`NONE`'.

mbsize — Mailbox size, in bytes. Defined only in '`mbq`' mode.

msgsize — Expected message size, in bytes. Defined only in '`mbq`' mode.

6.2.2 Mailutils Loading Sequence

```
module mailutils mailutils [args]
```

Arguments are:

`config-verbose`
: Verbosely trace the processing of the main Mailutils configuration files.

`config-dump`
: Dump the parse tree from the Mailutils configuration.

`positive-reply=`*`str`*
: Declare default positive reply string. This string is returned when the underlying database was able to found the requested key. Prior to returning, *str* is subject to meta-variable expansion, as described above.

  Default positive reply string is '`OK`'.

`negative-reply=`*`str`*
: Declare default negative reply string. This string is returned when the underlying database failed to found the requested key. Prior to returning, *str* is subject to meta-variable expansion.

  Default negative reply string is '`NOTFOUND`'.

`onerror-reply=`*str*
: Declare a reply to be returned on error. Prior to returning, *str* is subject to meta-variable expansion. Default string is '`NOTFOUND`'.

The module reads most of its configuration settings from the main Mailutils configuration file. See Section "configuration" in *GNU Mailutils Manual*, for a description of GNU Mailutils configuration system. It looks for `smap`-specific settings in the section '`program smap-mailutils`'.

Statement	**Reference**
server	See Section "Server Settings" in *GNU Mailutils Manual*.
auth	See Section "Auth Statement" in *GNU Mailutils Manual*.
pam	See Section "PAM Statement" in *GNU Mailutils Manual*.
virtdomain	See Section "Virtdomain Statement" in *GNU Mailutils Manual*.
radius	See Section "Radius Statement" in *GNU Mailutils Manual*.
sql	See Section "SQL Statement" in *GNU Mailutils Manual*.
ldap	See Section "LDAP Statement" in *GNU Mailutils Manual*.
debug	See Section "Debug Statement" in *GNU Mailutils Manual*.
logging	See Section "Logging Statement" in *GNU Mailutils Manual*.
include	See Section "Include" in *GNU Mailutils Manual*.

The module uses GNU Mailutils authorization databases to obtain the requested data. This concept is described in detail in Section "Auth Statement" in *GNU Mailutils Manual*.

6.2.3 Mailutils Databases

Mailutils databases are normally declared as follows:

```
database name mailutils mode=mode [args]
```

Here, *name* is the database name, *mode* is '`auth`' if the database should work in auth mode, and '`mbq`' if it should run in mbq mode. If the '`mode`' argument is omitted, '`auth`' is assumed. Optional *args* may be used to supply additional database configuration. These are:

`positive-reply=`*str*
: Declare positive reply string. This string is returned when the underlying database was able to found the requested key. Prior to returning, *str* is subject to meta-variable expansion, as described above.

  Default positive reply string is '`OK`', unless overridden by the module-level `positive-reply` option (see Section 6.2.3 [db-mailutils], page 27.

`negative-reply=`*str*
: Declare negative reply string. This string is returned when the underlying database failed to found the requested key. Prior to returning, *str* is subject to meta-variable expansion.

  Default negative reply string is '`NOTFOUND`', unless overridden by the module-level `positive-reply` option (see Section 6.2.3 [db-mailutils], page 27.

`onerror-reply=`*str*
: Declare a reply to be returned on error. Prior to returning, *str* is subject to meta-variable expansion. Default string is '`NOTFOUND`', unless overridden by the module-level `positive-reply` option (see Section 6.2.3 [db-mailutils], page 27.

6.2.4 Mailutils Auth Mode

Mailutils module in '`auth`' mode uses GNU Mailutils authorization mechanism to obtain user data. It returns '`positive-reply`' if the data were retrieved and '`negative-reply`' otherwise. This mode is often used for databases of local users and aliases. The key is normally a user name (either local part or fully qualified).

6.2.5 Mailutils MBQ Mode

MBQ, or *Mailbox Quota* mode, uses key as the name of a local user. It obtains the user parameters via Mailutils authorization mechanism and then switches to this user privileges and opens his mailbox for a brief period of time. After opening it determines the mailbox size and closes it. The mode returns '`positive-reply`' if the mailbox size is less than the quota, and '`netagive-reply`' otherwise.

If the key value consists of two words, separated by whitespace, then the first word is used as a user name, and the second one as a size of a message which is about to be delivered to that user's mailbox (the size may be optionally prefixed by '`SIZE=`'). In this case, '`positive-reply`' is returned if the actual mailbox size plus the message size is less than quota.

Two additional meta-variables may be used in reply templates to return quota-related information:

mbsize
: Mailbox size, in bytes. Defined only in '`mbq`' mode.

msgsize
: Expected message size, in bytes. Defined only in '`mbq`' mode.

The following example shows a definition of '`mbq`' database which the author uses on his servers:

```
database mbq mailutils mode=mbq \
  positive-reply="OK [${diag}] ${mailbox} ${mbsize} ${quota}"\
  negative-reply="NOTFOUND [${diag}] ${mailbox} ${mbsize} ${quota}"\
  onerror-reply="NOTFOUND [${diag}]"
```

The '`diag`' meta-variable contains a diagnostic string suitable for passing it back to the MTA. For example, in the case of '`negative-reply`', '`${diag}`' expands to:

```
mailbox quota exceeded for this recipient
```

if the mailbox has grown beyond the allowed quota, and

```
message would exceed maximum mailbox size for this recipient
```

if message of the given size cannot be delivered to mailbox without violating its quota.

Notice, that this mode requires superuser privileges.

6.3 Guile

Guile is an acronym for *GNU's Ubiquitous Intelligent Language for Extensions*. It provides a Scheme interpreter conforming to the R5RS language specification and a number of convenience functions. For information about the language, refer to *Revised(5) Report on the Algorithmic Language Scheme*. For a detailed description of Guile and its features, see Section "Overview" in *The Guile Reference Manual*.

The `guile` module provides an interface to Guile which allows for writing Smap modules in Scheme. The module is loaded using the following configuration file statement:

```
module name guile [args]
```

Optional *args* are:

`debug` Enable Guile debugging and stack traces.

`nodebug` Disable Guile debugging and stack traces (default).

`load-path=`*path*
: Append directories from *path* to the list of directories which should be searched for Scheme modules and libraries. The *path* must be a list of directory names, separated by colons.

  This option modifies the value of Guile's `%load-path` variable. See Section "Configuration Build and Installation" in *The Guile Reference Manual*.

`init-script=`*script*
: Specifies the name of a Scheme source file that must be loaded in order to initialize the module. The file is looked up using '`%load-path`' variable.

`init-args`
: The `init-args` parameter supplies additional arguments to the module. They will be accessible to the *`script`* via the `command-line` function.

`init-fun` This parameter specifies the name of a function that will be invoked to perform the initialization of the module and of particular databases. Default name is '`init`'. See Section 6.3.3 [Guile Initialization], page 31, for a description of initialization sequence.

Guile databases are declared using the following syntax:

```
database dbname modname [args] [cmdline]
```

where: *dbname* gives the name for this database and *modname* is the name given to Guile module in `module` statement (see above).

Optional *args* override global settings given in the `module` statement. The following options are understood: `init-script`, `init-args`, and `init-fun`. Their meaning is the same as for `module` statement (see above), except that they affect only this particular database.

Any additional arguments, referenced as *cmdline* above, are be passed to the Guile `open-db` callback function (see [open-db], page 31).

6.3.1 Virtual Functions

Any database handled by `guile` module is associated with a *virtual function table*. This table is an association list which supplies to the module the Scheme *call-back functions* implemented to perform particular tasks on that database. In this list, the `car` of each element contains the name of a function, and its `cdr` gives the corresponding function. The defined function names and their semantics are described in the following table:

init	Initialize the module.
done	Close the module, releasing any resources held by it.
open	Open the database.
close	Close the database.
query	Handle a socket map query
xform	Handle a transformation request (see Section 3.10 [transformations], page 15).

For example, the following is a valid virtual function table:

```
(list (cons "open" open-module)
      (cons "close" close-module)
      (cons "query" run-query))
```

Apart from per-database virtual tables, there is also a global virtual function table, which is used to supply the entries missing in the former. Both tables are created during the module initialization, as described in the next subsection.

Particular virtual functions are described in Section 6.3.4 [Guile API], page 31.

6.3.2 Guile Output Ports

Guile modules are executed in a specially prepared environment. Current error port is redirected so that everything written to it ends up in the `smapd` error stream. So, if `smapd` is writing its log to syslog, everything you write to '`(current-error-port)`' will be written to syslog as well. The port is line-buffered. For example, the following code:

```
(with-output-to-port
  (current-error-port)
 (lambda ()
   (display "The diagnostics follows:")
   (newline)
   (display "Module opened")
   (newline)))
```

will result in two lines in your syslog file, which will look like

```
Jun 19 12:49:05 netbox smapd[7503]: The diagnostics follows
Jun 19 12:49:05 netbox smapd[7503]: Module opened
```

For any debugging output, use `smap-debug-port`. This port is configured so that everything written to it is explicitly marked as being debug output. If `smapd` logs to stderr, it will be prefixed with '`DEBUG:`', and if it logs to syslog, the output will be logged with '`LOG_DEBUG`' priority.

Finally, current output port is closed for any functions, excepting '`query`' (see [query-db], page 32). For '`query`' function, it is redirected so that anything written to it is reformatted

according to the socket map protocol (see Appendix B [Protocol], page 55) and sent back as a reply to the client.

6.3.3 Guile Initialization

The `module` configuration statement causes loading and initialization of the `guile` module:

```
module modname guile [init-script=script] \
                            [init-fun=function"]
```

Upon module initialization stage, the module attempts to load the file named *`script`*. The file is loaded using `primitive-load-path` call (see Section "Loading" in *The Guile Reference Manual*), i.e. it is searched in the Guile load path. The `init-fun` parameter supplies the name of the *initialization function*. This Scheme function returns virtual function tables for the module itself and for each database that uses this module. It must be declared as follows:

```
(define (function arg)
  ...)
```

This function is called several times. First of all, it is called after *script* is loaded. This time it is given `#f` as its argument, and its return value is saved as a global function table. Then, it is called for each `database` statement that uses module *modname* (defined in the `module` statement above), e.g.:

```
database dbname modname ...
```

This time, it is given *dbname* as its argument and its return is stored as the virtual function table for this particular database.

The following example function returns a complete virtual function table:

```
(define (my-smap-init arg)
  (list (cons "init" db-init)
        (cons "done" db-done)
        (cons "open" db-open)
        (cons "close" db-close)
        (cons "query" db-query)
        (cons "xform" db-xform)))
```

6.3.4 Guile API

This subsection describes callback functions that a Guile database module must provide. The description of each function begins with the function prototype and its entry in the virtual function table.

`open-db` *name . args* [Guile Callback]

Virtual table: `(cons "open" open-db)`

Open the database. The argument *name* contains database name as given in `dbname` of the `database` declaration (see Section 3.8 [databases], page 13). Optional argument *args* is a list of command line parameters obtained from *cmdline* in `database` statement (see [guile-cmdline], page 29). For example, if the configuration file contained:

```
database foo guile db=file 1 no
```

then the `open-db` callback will be called as:

```
(open-db "foo" '("db=file" "1" "no"))
```

The `open-db` callback returns a *database handle*, i.e. an opaque Scheme object which identifies this database, and keeps its internal state. This value, hereinafter named *dbh*, will be passed to another callback functions that need to access the database.

The unspecified return value indicates an error.

`close-db` *dbh* [Guile Callback]

Virtual Table: `(cons "close" close-db)`

Close the database. This function is called during the cleanup procedure, before termination of `smapd` child process. The argument `dbh` is a database handle returned by `open-db`.

The return value from `close-db` is ignored. To communicate errors to the daemon, throw an exception.

`query-db` *dbh map key . rest* [Guile Callback]

Virtual Table: `(cons "close" close-db)`
Perform the query. Arguments are:

dbh A database handle returned by `open-db`.

map The map name.

key The lookup key.

rest If this query came over a UNIX socket, this argument is '`()`'. Otherwise, if the query came over an INET socket, *rest* is a list of two network socket addresses (see Section "Network Socket Address" in *The Guile Reference Manual*): first element is the address of the remote party (client), second element is the address of the server that is handling the query.

This function must write the reply, terminated with a newline, to the current output port, e.g.:

```
(define-public (smap-query handle map arg . rest)
  (display "NOTFOUND")
  (newline))
```

`xform-db` *dbh arg . rest* [Guile Callback]

Virtual Table: `(cons "xform" xform-db)`
Transform the argument *arg*. Arguments *dbh* and *rest* have the same meaning as in [query-db], page 32.

Returns transformed value or '`#f`' if no transformation applies. This callback may be used to alter map or key values using '`guile`' module (see Section 3.10 [transformations], page 15). The following example function removes optional domain part from its argument:

```
(define (smap-xform handle arg . rest)
  (let ((arg-parts (string-split arg #\@)))
    (if (null? (cdr arg-parts))
#f
(car arg-parts))))
```

The following snippet from the `smapd.conf` file shows how to apply it:

```
database localpart guile init-script=local.scm

dispatch key like *@* transform key localpart
```

6.4 Mysql

The `mysql` module provides interface to MySQL database management system. It may be used to build smap databases over SQL ones.

The SQL database to use may be configured either globally, when loading the module, or locally, when defining a smap database. If a database definition lacks SQL configuration statements, then it attempts to use a globally defined connection.

Each database is configured with a *SQL query template*, and a set of *smap reply templates* to use. When dispatched a sockmap query, the database expands the SQL query template using the actual values of '`${map}`' (the map name) and '`${key}`' (the key value) and sends the expanded query to the MySQL server. If the server responds with a non-empty set of tuples, the *positive reply template* is expanded and the result is used as a response. Otherwise, if the query produced an empty set, the smap database uses the *negative reply template* to create the response.

6.4.1 MySQL Configuration

The SQL database is configured using the following options:

`config-file=`*file*
: Set the name of the MySQL configuration file to read. By default `/etc/my.cnf` is used.

`config-group=`*name*
: Set the name of the group in the MySQL configuration file, from where to read the configuration options.

The statements above allow to keep all security-sensitive information, such as MySQL username and password, in an external configuration file and thus to relax permission requirements for `smapd.conf`. For a detailed description of the format of such external configuration file (or *option file* in '`MySQL`' parlance), see Section "option-files" in *MySQL Manual*.

In case the use of option files is not feasible for some reason, you may specify MySQL connection and database parameters in `smapd.conf` when loading the `mysql` module or defining a smap database. The following options are used to define MySQL connection parameters:

`host=`*hostname*
: Sets the hostname or IP address of the host running the MySQL server.

`port=n` Sets port number the MySQL server is listening on. Default is 3306.

`socket=file`
: Sets the socket name, if the server is listening on a UNIX socket.

`ssl-ca=file`
: Sets the pathname to the certificate authority file, if you wish to use a secure connection to the server via SSL.

Notice, that either `host` and, optionally, `port` or `socket` must be used. Specifying both is senseless.

MySQL database and user credentials are defined using the following options:

`database=name`
: Sets the name of the MySQL database to use.

`user=name`
: Sets MySQL user name.

`password=string`
: Sets the password for accessing the MySQL database.

When using these options, it is reasonable to tighten the permissions on `smapd.conf` so that no third person could see the MySQL password. The recommended permissions are '`0600`'.

6.4.2 MySQL Query and SMAP Replies

MySQL query is defined using the following option:

`query=template`
: Define MySQL query template.

The *template* may reference the following variables:

Variable	**Meaning**
map	Name of the map being queried
key	Lookup key

Table 6.1: MySQL query template variables

For example:

```
database alias mysql \
  query="SELECT alias FROM aliases WHERE email='$key'"
```

If the database definition lacks the `query` option, it will attempt to use one from the module statement. If the module statement lacked it as well, an error is reported.

Reply templates define the responses to be given. They are given by the following options:

`positive-reply=template`
: Defines a reply to be sent if the query returned a non-empty set of tuples. In addition to the variables described above (see Table 6.1), the *template* may also refer to the MySQL result columns, by using their names from the '`SELECT`' part of the query. For example:

```
database alias mysql \
  query="SELECT alias FROM aliases WHERE email='$key'" \
  positive-reply="OK $alias"
```

The default `positive-reply` is '`OK`'.

`negative-reply=`*template*
: Defines a reply to be sent if the query returned an empty set of tuples. The *template* may refer to the variables described in Table 6.1.

  Default value is '`NOTFOUND`'.

`onerror-reply=`*template*
: Defines a reply to be sent if an error occurred when executing the query. The *template* may refer to the variables described in Table 6.1.

  Default value is '`NOTFOUND`'.

6.5 Postgres

The `postgres` module provides interface to PostgreSQL database management system. It may be used to build smap databases over SQL ones.

The module is in many regards similar to `mysql` module, described above. In particular, its overall functionality is exactly the same as described in Section 6.4 [mysql], page 33, except, of course, that it uses PostgreSQL databases.

6.5.1 Postgres Configuration

A Postgres database is configured using a set of options understood by the Postgres `PQconnectdb` function. See `http://www.postgresql.org/docs/8.4/static/libpq-connect.html`, for a detailed description. The following is a short summary of the most useful options:

`host=`*name*
: Name of host to connect to. If this begins with a slash, it specifies Unix-domain communication rather than TCP/IP communication; the value is the name of the directory in which the socket file is stored.

`hostaddr=`*ip*
: Numeric IP address of host to connect to.

`port=`*number*
: Port number to connect to at the server host, or socket file name extension for Unix-domain connections.

`dbname=`*name*
: The database name.

`user=`*name*
: PostgreSQL user name to connect as. Defaults to be the same as the operating system name of the user running the `smapd`.

`password=`*string*
: Password to be used if the server demands password authentication.

`connect_timeout=`*`number`*
: Maximum wait for connection, in seconds. Zero or not specified means wait indefinitely.

`options=`*`string`*
: Any additional command-line options to send to the server at run-time. For example, setting this to '`-c geqo=off`' sets the session's value of the '`geqo`' parameter to '`off`'. For a detailed discussion of the available options, see Postgres documentation[1].

`sslmode=`*`mode`*
: This option determines whether or with what priority an SSL TCP/IP connection will be negotiated with the server. There are six modes: '`disable`', '`allow`', '`prefer`', '`require`', '`verify-ca`' and '`verify-full`'[2].

`sslcert=`*`file`*
: This parameter specifies the file name of the client SSL certificate.

`sslkey==`*`file-or-engine-name`*
: This parameter specifies the location for the secret key used for the client certificate.

`sslrootcert=`*`file`*
: This parameter specifies the file name of the root SSL certificate.

`sslcrl=`*`name`*
: This parameter specifies the file name of the SSL certificate revocation list (CRL).

`krbsrvname=`*`name`*
: Kerberos service name to use when authenticating with Kerberos 5 or GSSAPI.

`service=`*`name`*
: Service name to use for additional parameters.

6.5.2 Postgres Query and SMAP Replies

Postgres SQL query and the smap replies are configured the same way as for `mysql` module (see Section 6.4.2 [MySQL Query and SMAP Replies], page 34). The following is a short summary:

`query=`*`template`*
: Define the Postgres query template. The *template* may reference the following variables:

Variable	Meaning
map	Name of the map being queried
key	Lookup key

Table 6.2: Postgres query template variables

[1] For PostgreSQL version 8.4, see Chapter 18 in PostgreSQL Manual.

[2] For PostgreSQL version 8.4, see Section 30.17 in PostgreSQL Manual.

If the database definition lacks the `query` option, it will attempt to use one from the module statement. If the module statement lacked it as well, an error is reported.

`positive-reply=`*template*
: Defines a reply to be sent if the query returned a non-empty set of tuples. In addition to the variables described above (see Table 6.2), the *template* may also refer to the column names from the SQL result set.

  The default `positive-reply` is '`OK`'.

`negative-reply=`*template*
: Defines a reply to be sent if the query returned an empty set of tuples. The *template* may refer to the variables described in Table 6.2.

  Default value is '`NOTFOUND`'.

`onerror-reply=`*template*
: Defines a reply to be sent if an error occurred when executing the query. The *template* may refer to the variables described in Table 6.2.

  Default value is '`NOTFOUND`'.

6.6 ldap

The `ldap` module provides interface to the Lightweight Directory Access Protocol. The configuration is similar to that of SQL modules:

LDAP parameters may be configured either globally, when loading the module, or locally, when defining a smap database. If the database definition lacks some configuration statements, it looks them up in a global definition.

Each database has a *filter template* and up to three *smap reply templates*. When dispatched a sockmap query, the database expands the filter template using the actual values of '`${map}`' (the map name) and '`${key}`' (the key value) and uses the obtained filter to query the LDAP server. If the server responds with a non-empty set of tuples, the *positive reply template* is expanded and the result is used as a response. Otherwise, if the query produced an empty set, the smap database uses the *negative reply template* to create the response.

The module gets its configuration from the file `/etc/ldap.conf` and from module and database command line. The settings from the command line override those from `/etc/ldap.conf`. Alternative configuration file can be specified using the `config-file` option. The subsections that follow discuss the keywords meaningful for the `ldap` module. Unless explicitly stated otherwise, these can be used in the command line as well as in the configuration file. For compatibility with other LDAP software, keywords in the configuration file are case-insensitive. Unrecognized keywords appearing in the configuration file are silently ignored. You can use the '`ldap.2`' debug level to get a listing of those. This can be useful to trace possible typos.

Unrecognized keywords appearing in the command line are treated as errors, as usual.

The only keyword that can be used only in the command line is `config-file`:

`config-file=`*file*
: Read configuration from file *file* instead of `/etc/ldap.conf`.

6.6.1 LDAP Configuration

The following keywords configure access to the LDAP database:

`base=`*`string`*
: Sets the default base DN for ldap operations. The base must be specified as a Distinguished Name in LDAP format.

`binddn=`*`dn`*
: The DN to bind as.

`bindpw=`*`password`*
: Password for `binddn`.

`bindpwfile=`*`file`*
: Read password from *file*. This is a safer alternative to `bindpw`.

`tls-cacert=`*`file`*
`tls_cacert=`*`file`*
: Read TLS Certificate Authority from *file*.

`uri=`*`string`*
: Specifies the URI of LDAP server to connect to. Multiple URIs are allowed. Each URI is '*`scheme://[name[:port]]`*'. The *scheme* part is one of: '`ldap`', meaning LDAP over TCP (default port 389), '`ldaps`', meaning LDAP over SSL (TLS) (default port 636), or '`ldapi`', meaning LDAP over UNIX socket. For '`ldap`' and '`ldaps`', *name* is the host name or IP address of the remote server. Optional *port* specifies the TCP port to use instead of the default one. For '`ldapi`', *name* is the pathname of the UNIX socket and *port* is not used. Note, that directory separators must be URL-encoded (using '`%2F`' instead of '`/`').

6.6.2 LDAP Filter and SMAP Replies

The following keywords configure LDAP lookups and replies.

`join-delim=`*`string`*
: When constructing a reply, join multiple occurrences of LDAP attribute with *string*. If this parameter is not defined, only first attribute will be returned.

`filter=`*`pattern`*
: Specifies LDAP filter. The *pattern* can use the usual variables (see Section 6.2.1 [expansion], page 25). For example:

```
database user ldap filter=(&(objectClass=posixAccount)(uid=$key))
```

: There is no default for this option, so it is mandatory.

Replies are configured via the following three keywords:

`positive-reply=`*`reply`*
: Defines a positive reply string. It is used when the LDAP lookup using the defined filter returned one or more objects. Only the first returned object is used. The *reply* string can contain the basic `smap` variables '`$db`', '`$map`', and '`$key`'. It can also refer to values of any attribute from the returned object using the variable notation. For example:

```
positive-reply="OK $uid"
```

returns the string 'OK' followed by the value of the `uid` attribute.

The default positive reply string is 'OK'.

`negative-reply=`*reply*
: Defines the negative reply string, which is used when the LDAP lookup returns empy set of objects. The *reply* string can contain the basic `smap` variables '`$db`', '`$map`', and '`$key`'.

The default negative reply string is '`NOTFOUND`'.

`onerror-reply=`*reply*
: Defines the string to be returned if the LDAP lookup fails. The *reply* argument can contain the basic `smap` variables '`$db`', '`$map`', and '`$key`'.

The default value is '`NOTFOUND`'.

6.7 Sed

The '`sed`' module applies sed-like *s-expressions* to strings in order to modify them. It is designed mainly for use in transformation rules (see Section 3.10 [transformations], page 15).

6.7.1 Loading sed module

```
module sed sed [args]
```

The following arguments may be given in the statement above:

`icase` Use case-insensitive expressions.

`noicase` Use case-sensitive expressions. This is the default.

`extended` Use extended regular expressions. This is the default.

`noextended`
: Use basic regular expressions.

Although `sed` module is designed for transformations in the first place, it may also be used as a conventional lookup database (see Section 6.7.4 [sed lookups], page 40). The following options modify its behavior in this mode:

`positive-reply=`*str*
: Use *str* for positive replies.

`negative-reply=`*str*
: Use *str* for negative replies.

`onerror-reply=`*str*
: Reply with *str* if an error occurred.

6.7.2 Defining Sed Databases

The definition of a sed databases requires a single argument: the *s-expression* to be applied. For example:

```
database dequote sed 's/<(.*)>/\1/g'
```

Be sure to properly quote the expression, especially if it contains backreferences. It is preferable to use single quotes, to avoid duplicating each backslash in the expression, as

shown in the example below. If the expression itself contains single quote, you may either use double-quotes to quote the entire expression:

```
database foo sed "s/'utf8'(.*)/u8_\\1/"
```

or use escaped single quotes outside of quoted expression (a technique familiar for shell programmers):

```
database foo sed 's/'\''utf8'\''(.*)/u8_\1/'
```

All options valid for module initialization (see Section 6.7.1 [sed module], page 39) may also be used in database declarations. When used so, they take precedence over module initialization options. For example, the following database definition uses basic case-insensitive regular expressions:

```
database bar sed noextended noicase 's/test(\([^)]\))/\1/g'
```

6.7.3 S-expressions

The transformation expression is a `sed`-like replace expression of the form:

```
s/regexp/replace/[flags]
```

where *regexp* is a *regular expression*, *replace* is a replacement for each part of the input that matches *regexp*. Both *regexp* and *replace* are described in detail in Section "The 's' Command" in *GNU sed*.

As in `sed`, you can give several replace expressions, separated by a semicolon.

Supported *flags* are:

'`g`' Apply the replacement to *all* matches to the *regexp*, not just the first.

'`i`' Use case-insensitive matching

'`x`' *regexp* is an *extended regular expression* (see Section "Extended regular expressions" in *GNU sed*).

'*`number`*' Only replace the *number*th match of the *regexp*.

Note: the POSIX standard does not specify what should happen when you mix the '`g`' and *number* modifiers. The `sed` module follows the GNU `sed` implementation in this regard, so the interaction is defined to be: ignore matches before the *number*th, and then match and replace all matches from the *number*th on.

Any delimiter can be used in lieue of '`/`', the only requirement being that it be used consistently throughout the expression. For example, the following two expressions are equivalent:

```
s/one/two/
s,one,two,
```

Changing delimiters is often useful when the *regex* contains slashes. For instance, it is more convenient to write `s,/,-,` than `s/\//-/`.

6.7.4 Using Sed for Lookups

The `sed` module is designed primarily for argument transformation. Nevertheless, it may also be used to define simple look-up databases. When used in a `database` clause of a dispatch rule, the module behaves as follows. The s-expression is applied to the key. If the result differs from the input key, the '`positive-reply`' is returned. It the result is the same

as the input key, '`negative-reply`' is returned. If some error occurred, '`onerror-reply`' is returned. The reply strings may be supplied as arguments to the database definition or to the module loading statement. The following variables are expanded within these strings:

'`map`' The map name.

'`key`' The key value.

'`xform`' Transformed key value. This variable is not defined for '`onerror-reply`'.

Default replies are:

Reply	**Value**
positive-reply	'`OK ${xform}`'
negative-reply	'`NOTFOUND`'
onerror-reply	'`NOTFOUND`'

7 Socket map client

The `smapc` program is a console-based utility for querying socket map servers. It has two operation modes. In *single query mode*, the utility performs a query, displays its result and exits immediately. In *interactive mode*, the utility enters a read-and-eval loop, in which it reads queries from the keyboard, runs them, and displays obtained results on the screen.

7.1 Single Query Mode

The simplest way to use `smapc` utility is to invoke it as follows:

```
smapc -S url map key
```

The `-S` option introduces the URL of the server to query (see [smap url], page 10). The *map* argument gives the name of the map to use, and the *key* argument supplies the search key.

For example:

```
$ smapc -S unix:///var/run/smap/sockmap aliases root@example.com
OK smith dmk <rev@mail.example.org>
```

You can give as many map-key pairs in the command line as is necessary, the only requirement being that the number of arguments be even:

```
$ smapc -S unix:///var/run/smap/sockmap aliases root@example.com users root
OK smith dmk <rev@mail.example.org>
OK root uid 0
```

If multiple map-key pairs are given in the command line, `smapc` can *annotate* each response with the corresponding query. Such annotations are enabled by the `-a` (`--annotate`) option, e.g.:

```
$ smapc -S unix:///var/run/smap/sockmap -a aliases root@example.com users root
aliases root@example.com: OK smith dmk <rev@mail.example.org>
users root: OK root uid 0
```

You may simplify the invocation if you add the URL to your *initialization file*, i.e. to the file `smapc` reads at startup for its defaults. This file resides in your home directory and is named `.smapc`. Open this file with your favorite editor, and add the following line to it:

```
open unix:///var/run/smap/sockmap
```

Now, when invoked without the `-S` option, `smapc` will use this URL by default:

```
$ smapc aliases root@example.com
OK smith dmk <rev@mail.example.org>
```

See Section 7.3 [Initialization File], page 46, for a detailed description of this file.

7.2 Interactive Mode

If insufficient number of arguments is given in the command line, `smapc` enters interactive mode. In this mode it reads commands from the standard input, executes them and displays the results on the standard output. If the standard input is connected to a terminal, the readline and history facilities are enabled (see Section "Command Line Editing" in *GNU Readline Library*).

When in interactive mode, `smapc` displays its prompt and waits for you to enter a command. The default prompt is the name of the program, enclosed in parentheses:

```
(smapc) _
```

Depending on the first character, your input is recognized either as a `smapc` command, or as a query. All `smapc` commands begin with a single punctuation character, called *command prefix*. The default command prefix is a dot, but it can be changed using the `prefix` command (see Section 7.2.1 [Command Summary], page 45). The prefix is not a part of the command, it is merely a means by which `smapc` recognizes that it has been given a command. So, when explaining commands below, we will refer to them by their name, without the prefix.

The most important command is '`open`'. It takes a server URL as its argument and opens a connection to that server:

```
(smapc) .open unix:///var/run/smap/sockmap
```

Now, if you type two or more words (the first of them not starting with the command prefix), `smapc` builds a query using the first word as of them is used as a map name and the rest of them as a key. It then sends the request to the server using the socket opened with the `open` command and displays the result on the standard output:

```
(smapc) aliases root@domain.com
OK smith dmk <rev@another.domain>
```

If you wish to change to another URL, give another '`open`' command. Do not bother to close the previously opened socket: it will be done automatically.

If you are going to send a series of queries using the same map, you will save yourself some typing by declaring the *default map*, e.g.:

```
(smapc) .map aliases
```

From now on, every non-command input you give will be treated as lookup keys for that map name, e.g.:

```
(smapc) root@domain.com
OK smith dmk <rev@another.domain>
(smapc) postmaster
OK root
(smapc) daemon
NOTFOUND
```

If you forget what map you are currently using, type the *map* command without arguments. It will display the map name:

```
(smapc) .map
current map is aliases
```

Finally, to forget the default map and return to typing map name before the key, use '`nomap`':

```
(smapc) .nomap
```

To quit the program, type '`.quit`'. Typing end-of-file character (`C-d`) has the same effect.

To obtain a listing of available commands with a short description for each of them, type '`help`' or '`?`'.

7.2.1 Smapc Command Summary

This subsection lists all available `smapc` commands along with their short description and a reference to the part of this manual where they are described in detail. The command names are given without prefix.

`annotate` [*bool*] [smapc]
: Without arguments, displays current status of annotations (see [annotation], page 43). If *bool* is `true`, enables annotations. If it is `false`, disables it.

  Allowed values for `true` are: `true`, `t` , `yes`, `on`. Allowed values for `false` are: `false`, `nil`, `no`, `off`.

`close` [smapc]
: Close previously opened connection.

`debug` *spec* [smapc]
: Sets the debug level. See Section 3.3 [debugging], page 8, for a description of *spec*.

`help` [smapc]
: Display short command usage summary.

`history` [smapc]
: Prints the history of recently issued commands.

`nomap` [smapc]
: Clear the default map name. After this command, map names must be given explicitly with each query. See [smapc-defmap], page 44.

`map` [*name*] [smapc]
: Set the default map name. Without arguments, print the name of the map currently in use. See [smapc-defmap], page 44.

`open` *url* [smapc]
: Open connection to socket map server at the given *url*. See [smapc-open], page 44.

`server` [smapc]
: Show URL of the currently opened connection.

`source` [*ip*] [smapc]
: With argument, sets the source address for outgoing queries. Without argument, displays currently used source address.

`prefix` [*char*] [smapc]
: If *char* is given, sets it as the command prefix. If called without arguments, displays the currently selected command prefix.

`prompt` [*string*] [smapc]
: Redefines the command prompt. Without arguments, prints the current prompt.

`quit` [smapc]
: Quits interactive mode.

`quiet` *bool* [smapc]

This command command toggles the display of `smapc` startup banner. When started, `smapc` prints a short list of information useful for beginning users: the program version and warranty conditions and a command to get help, e.g.:

```
smapc (smap) 2.1
Copyright (C) 2010 Sergey Poznyakoff
License GPLv3+: GNU GPL version 3 or later
<http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Type ? for help summary

(smapc) _
```

If you find this output superfluous and useless, you can suppress it by setting

```
quiet yes
```

in your initialization file.

`version` [smapc]

Displays the package name and version number.

`warranty` [smapc]

Displays the copyright statement.

7.3 Initialization File

When you start `smapc`, it automatically executes commands from its *initialization file*, if such file exists. This file is located in your home directory and called `.smapc`.

Initialization file contains a series of `smapc` commands, as described in Section 7.2 [Interactive Mode], page 43, with the only difference that no command prefix is used by default. The '`#`' character introduces a comment: any characters from (and including) '`#`' up to the newline character are ignored[1].

Init files are useful to change the defaults for your `smapc` invocation. Consider, for example, this init file:

```
# Turn welcome banner off
quiet yes
# Open the default connection
open inet://127.0.0.1:3145
# Use 'aliases' as a default map
map aliases
# Finally, set the custom command prefix
prefix :
```

Notice, that if you wish to change your command prefix, it is preferable to do it as a last command in your init file, as shown in this example.

[1] The same holds true for interactive mode as well, but you will hardly need comments on a terminal.

7.4 Smap Invocation

The following table summarizes the options available for `smapc`. For each option a description is given and a cross reference is provided to more in-depth explanation in the body of the manual.

`-a`
`--annotate`
: Annotate responses with the corresponding queries. see [annotation], page 43.

`-B`
`--batch`
: Enable batch mode. This mode is optimized for reading input from files. The startup banner is suppressed, editing capabilities and input history are disabled, and input prompt is not shown. This mode is enabled automatically if `smapd` detects that its standard input is not connected to a terminal.

`-d` *spec*
`-x` *spec*
`--debug=`*spec*
: Set debug verbosity level. See Section 3.3 [debugging], page 8, for a detailed description. The `-x` alias is for compatibility with version 1.0 and will be removed in subsequent releases.

`-h`
`--help`
: Give a short summary of available command line options.

`-p` *string*
`--prompt=`*string*
: Change command prompt. See [smapc-prompt], page 45.

`-Q`
`--quiet`
: Do not print the normal welcome banner. See [smapc-quiet], page 46.

`-q`
`--norc`
: Do not read initialization file. See Section 7.3 [Initialization File], page 46.

`-S` *url*
`--server=`*url*
: Connect to server at the given *url*. See [smapc-open], page 44.

`-s` *addr*
`--source=`*addr*
: Set source address. See [smapc-source], page 45.

`-T`
`--trace`
: Enable query traces. See Section 3.3 [debugging], page 8.

`--usage`
: Display a list of available command line options.

`-V`
`--version`
: Print program version and exit.

8 How to Report a Bug

Email bug reports to gray+smap@gnu.org.ua. Please include a detailed description of the bug and information about the conditions under which it occurs, so we can reproduce it. The minimal information needed is:

- Version of the package you are using.
- Compilation options used when configuring it.
- Run-time configuration (the `smapd.conf` file and command line options used).
- Conditions under which the bug appears.

Appendix A Example: Using smapd with MeTA1

In this appendix we will show how to use the 'mysql' module (see Section 6.4 [mysql], page 33) to configure local user and alias maps for MeTA1. For this purpose, we will assume that the actual data is stored in two tables in a MySQL database. The two maps will be served by two separate databases, each of which uses a separate configuration file.

To reduce the number of connections to the MySQL server, the MySQL database will be opened at the module level and shared between the two smap databases. Thus, the module initialization in `smapd.conf` looks like:

```
module mysql mysql config-group=smap
```

The 'config-group' parameter refers to a group name in the default `/etc/my.cnf` file that contains information about the MySQL database and credentials for accessing it. The following is a sample snippet from `/etc/my.cnf`:

```
[smap]
database   = Mail
user       = smap
password   = guessme
socket     = /tmp/mysql.sock
```

A.1 Configure local_user_map.

Let's configure 'local_user_map' first. User data will be stored in the table 'userdb', which has the following structure:

```
CREATE TABLE userdb (
  user varchar(32) NOT NULL default '',
  mailbox text
  PRIMARY KEY (user)
);
```

The smap database is defined as follows:

```
database userdb mysql \
    query="SELECT user FROM userdb WHERE user='$key'"
    positive-reply=OK
```

The 'defaultdb' parameter tells it to use the default SQL database opened in the module initialization instruction. The 'query' parameter supplies the SQL query to run (the '${key}' variable will be expanded to the value of the actual lookup key, prior to executing the query). Finally, 'positive-reply' defines the reply to give if the query returns some tuples. The database only verifies whether the user is present or not, so no additional result is supplied in the reply.

A.2 Configure aliases

We are going to store aliases in the table 'aliases' which has the following structure:

```
CREATE TABLE userdb (
  user varchar(32) NOT NULL default '',
  alias text
  PRIMARY KEY (user)
);
```

It will be served by ‘alias’ database, defined as follows:

```
database alias mysql \
    defaultdb \
    query="SELECT alias FROM aliases WHERE user='$key'" \
    positive-reply="OK $alias"
```

It differs from the ‘userdb’ database only in that it returns a *result section* with its positive reply.

A.3 Dispatch Rules

The following rules dispatch queries based on their map names to the two databases:

```
dispatch map alias database aliasdb
dispatch map userdb database userdb
```

A.4 MeTA1 configuration

Finally we need to inform MeTA1 about new maps. This is done in the file /etc/meta1/meta1.conf, section ‘smar’.

First, the ‘userdb’ map:

```
map password { type = passwd; }
map userdb {
      type = socket;
      path = "/var/spool/meta1/smap/userdb";
      mapname = userdb;
}
map locusr {
      type = sequence;
      maps = { password, userdb };
}

local_user_map {
     name = "locusr";
     flags = { localpart, local_domains };
}
```

As a result, MeTA1 will look up users in the system database first, and, if that fails, in the SQL database.

Next, the ‘aliasdb’ map:

```
map lum {
      type = socket;
      path = "/var/spool/meta1/smap/userdb";
      mapname = aliases;
}
map stdal { file = "aliases.db"; type = hash; }
map aliasmap { type = sequence; maps = { lum, stdal }; }
aliases {
      name = aliasmap;
```

```
        flags = { localpart, local_domains };
}
```

As for '`userdb`', this map declaration also uses two different databases. First, it asks `smapd` to find the alias. If it returns a negative reply, the map falls back to the default `aliases.db` database.

Appendix B The Sockmap Protocol

Sockmap is a simple request/reply protocol over TCP or UNIX domain sockets. Both requests and replies are encoded in the following manner:

```
len:text,
```

where *text* is the actual payload, and *len* is its length in bytes, as a decimal number in ASCII representation. The colon and comma are transmitted verbatim. For example, if *text* is the string '`hello there`', then the socket map packet for transmitting it is:

```
11:hello there,
```

Sockmap requests consist of the *map name* and the actual lookup key, separated by a single space character.

Replies consist of the *status code* and optional data, separated by a single space character.

Below we describe status codes implemented by various programs. The bracketed parts in the '`code`' field of the tables below indicate optional values. The brackets themselves are not required by the protocol.

B.1 Sendmail Status Codes

Status codes understood by Sendmail are:

Code	**Meaning**
OK [*result*]	the key was found; *result* contains the looked up value.
NOTFOUND	the key was not found
TEMP [*reason*]	a temporary failure occurred; optional *reason* field contains an explanatory message.
TIMEOUT [*reason*]	same as '`TEMP`'.
PERM	a permanent failure occurred

Table B.1: Sendmail Status Codes

B.2 MeTA1 Status Codes

MeTA1 further extends the protocol. The result codes it understands are:

Code	**Meaning**
OK [*result*]	the key was found; *result* contains the looked up value.
NOTFOUND	the key was not found
NOMORE	the key was not found, stop further search
TEMP [*reason*]	a temporary failure occurred; optional *reason* field contains an explanatory message.
TIMEOUT [*reason*]	same as '`TEMP`'.
PERM [*reason*]	a permanent failure occurred; optional *reason* field contains an explanatory message.

Table B.2: MeTA1 Status Codes

The '`NOMORE`' status indicates that the key has not been found and also instructs MTA[1] to stop any further searches using this key and its derivatives.

B.3 Mailfromd Status Codes

Mailfromd does not itself require any particular status codes. The allowed status codes depend entirely on your filter program.

[1] To be precise, the `smar`, a component responsible for resolving various things for MeTA1.

Appendix C Debug Categories

The following table describes the debug categories available in the `smapd` server (see Section 3.3 [debugging], page 8). For each category, the table gives its symbolic name, ordinal number (in parentheses), and a short description.

Particular modules may define their own debug categories.

smap (0) Man smap functionality. Level '`1`' includes some mild warnings, like, e.g. '`ignoring master privilege settings`'.

Level '`10`' enables detailed protocol traces, which look like:

```
C: 22:mailertable foobar.net,
S: 19:OK local:foobar.net,
```

srvman (1)
Server manager, i.e. routines responsible for spawning children processes, controlling their number and lifetime, etc.

Level '`1`' gives additional information about allowed connections and children exit codes.

Level '`2`' gives insight to the server manager life cycle.

module (2)
Module subsystem: shows what modules and with what arguments are loaded, etc.

database (3)
Databases and their functionality.

query (4) Query dispatcher.

conf (5) Configuration file parser.

Level '`1`' enables warnings about undefined variables.

Level '`2`' displays each logical line and the result of expanding and splitting it.

Level '`100`' enables wordsplitter debugging. This means a *lot* of cryptic output useful only to those who have a good knowledge of how the wordsplitter works.

Appendix D GNU Free Documentation License

Version 1.2, November 2002

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released

under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

   You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and

that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

   If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

   If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

   If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

   It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

   You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

   A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

   You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

   The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

   In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called

an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

D.1 ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.2
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts.  A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Concept Index

This is a general index of all issues discussed in this manual.

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X