Binc IMAP manual (Version 2.0)

Andreas Aardal Hanssen, Erwin Hoffmann

March 1, 2024

Contents

Chapter 1

Introduction

> "As an alternative to existing similar IMAP servers, Binc IMAP strives to be very easy to use, but robust, stable and secure."

Welcome to Binc IMAP, a project started December 2003 by Andreas Aardal Hanssen. Binc IMAP is an open source IMAP project which differs from existing IMAP projects. Here is a list of the project goals:

1. Binc IMAP will always have a helpful, hospitable community.
    - Although it is expected that users of Binc IMAP do their homework before posting to the mailing list, the server author and community of the Binc IMAP project will be friendly and will approach everyone with respect. The same behavior is expected from those who post to the list.
    - There will be no RTFM[1] on the project's mailing list. Flaming and personal insults on the project's mailing list will result in banning of the originator.
    - The community is encouraged to pay back to the project's contributors by sharing their own experience and contributions to Binc IMAP through the GPL license.
2. Binc IMAP will not compete with other IMAP projects
    - Under no circumstance will this project be in market driven competition with other IMAP servers.
    - Binc IMAP will first and foremost be a quality driven project.
    - This project is meant to influence the community of authors of similar network protocols and servers, and hopes to increase the general quality of software that is used all over the globe and beyond.

[1] According to acronymfinder.com, RTFM stands, among many other suggestions, for "Read The Fscking Manual" (fsck is a Unix filesystem checker)

3. Binc IMAP provides security through good design
    - A well designed server is less exposed to bugs than a poorly designed server.
    - The server will strive to use every kind of security enhancing feature, while keeping the implementation details as good and simple as possible.
    - The source is open and downloadable. Potential bugs and/or nasty pieces of code are easily uncovered when the whole community is able to study every line of code in detail. Bugs should always be reported to the project's mailing list.
4. Binc IMAP is modular
    - Where possible and practical from both a usage and design stand point, modules will be seperated and replaceable through *pluggable extension support*. Examples of future replaceables and enhancements include:
        - authentication modules (shadow, PAM, LDAP)
        - search modules
        - protocol extensions modules (namespaces, ACL, shared folders)
        - mailbox formats (Maildir, mbox, MySQL, POP3 proxy)
    - With a modular and good object oriented design, it will be easy to quickly understand what every method and function does. This will increase third party developers' ability to write extensions and modifications fast.
5. Binc IMAP favors quality over quantity
    - Binc IMAP's releases are milestones. We strive for perfection.
    - Work on improving the existing design and extensibility will always go ahead of adding new features.
    - Through extensive module support, the community is encouraged to contribute to the adding and testing of new features.
    - Core design and implementation will always focus on quality.

This document serves as the server's main documentation.

1.1 Scope and requirements

Binc IMAP is a IMAP server suited for *Maildirs*, however not restricted to those. It should work with all MTAs providing Maildir support, but it is design to work mainly with **qmail** or **s/qmail** or other derivatives.

Here, it supports the virtual mail managers:

- `vpopmail`
- `vmailmgr`

Binc IMAP includes no networking stack and thus depends on particular TCP/IP services like

- `ucspi-ssl`'s **sslserver** or perhaps
- `ucspi-tcp6`'s **tcpserver**.

The current versions of `ucspi-tcp6` and `ucspi-ssl` require the installation of the `fehQlibs`.

In the layered communication model of Binc IMAP TLS encryption, cipher and X.509 certificate management needs to be provided on the network stack, which is the domain of **sslserver**. Apart from IMAPS services on port 993, *StartTLS* over port 143 is possible, but not recommended.

Others, like **inetd** or **xinetd** could work in principle, but are not recommended. To use Binc IMAP together with **stunnel** or other programs providing TLS capabilities should be possible, but was not tested.

Chapter 2

Installation

The installation procedure for Binc IMAP is designed to be quick and easy, and in most cases you will get what you want at your first attempt. The source code is designed to be able to compile on most UNIX-like platforms.

If you experience problems with compiling and installing this package on your own platform, don't hesitate to post your problem to the project's mailing list.

2.1 Downloading the package and requirements

Binc IMAP is available in the slashpackage source format to be installable on all POSIX compliant Unix systems, given a C++ compiler is available such as:

- GCC 10.2
- Clang 13.0

Binc IMAP will work on 32 and 64 bit systems. You can download the package from: https://www.fehcom.de/binc/binc.html.

2.2 Slashpackage installation

Before going to install Binc IMAP you have probably setup

- `ucspi-ssl`,
- `ucspi-tcp6`, and
- `s/qmail` (at `/var/qmail`)

thus you are familiar with the */slashpackage* installation procedure:

1. `cd /package`
2. `tar -xzf <path>/bincimap-2.x.y.tgz`
3. `cd mail/bincimap/bincimap-2.x.y`
4. `package/install`

Unlike previous Binc IMAP versions, *autoconf-tools* are not used here; and there is no need for any adjustments, apart from the following:

- Edit `conf-qmail`, if `s/qmail` (or `qmail`) is not installed at its default location.
- Edit `conf-man`, if you like to install Binc IMAPs man-pages at some self-defined locations.

Individual installation steps are as usual:

- package/compile – just compile.
- package/man – install man-pages.
- package/upgrade – in case new Binc IMAP packages are going to get installed.
- package/deploy – install the executables at their destination according to `conf-home` .

2.3 Configuration

Unlike the previous version of Binc IMAP, no configuration files are required; the entire setting can be done

- via *environment variables* – preferable given by means of **envdir**, or
- by *arguments* handed over to **bincimap-up** or **bincimapd**.

Note: Environment variables overwrite the command line arguments!

Assuming Binc IMAP is supervised by DJB's *daemontools* (or a variant, like *runit*), pre-configured run-scripts are available in the installation's `./service` directory:

- `bincimap`/**run** – Binc IMAP running un-encrypted under **tcpserver** on port 143.
- `bincimaps`/**run** – `IMAPS` services provided by **sslserver** over port 993.

- `bincimapstls`/**run** – Binc IMAP using StartTLS on port 143 while using **sslserver**.

Thus, we have a sharing of responsibilities:

1. **sslserver** is responsible to provide TLS encryption services and X.509 certificate management.
2. **bincimap-up** takes care of the IMAP configuration and the user log-in (authentication), and
3. **bincimapd** is the actual work-horse providing mail depot services and following the IMAP requests from you client.

2.3.1 Network Service

In Binc IMAP network services are available through

- **tcpserver** (from `ucspi-tcp6`; and other would do as well). This means an unencrypted IMAP service on port 143. Apart from the commands and the messages content, user names and passwords are readable in cleartext from the network, though Binc IMAP support CRAM-MD5 as authentication method.
- **sslserver** is invokes providing StartTLS support. Binc IMAP is still invoked on port 143 but using the environment variable '`UCSPITLS="+"`' announcing optional StartTLS service or with '`UCSPITLS="!"`' requiring the client to start a StartTLS session.
- **sslserver** is bound to the IMAPS port 993. From the beginning, a TLS session is setup. This is the preferred solution.

Other comparable solutions will probably work (`ucspi-tls`, `stunnel` in conjunction with `xinetd`). However, in any case using TLS by any other service, X.509 certificate and cipher management is now part of those programs and not of Binc IMAP, which was the case in the previous versions.

2.3.2 TLS

Though Binc IMAP may announce `STARTTLS`, there are not network capabilities included here. Binc IMAP depends on some network daemons, typically.

- **sslserver** or
- **tcpserver**

Other servers (like **xinetd** or **stunnel**) can be used with caution. In any case, environment variables providing information about the network peers need to be given as detailed in `tcp-env`.

StartTLS within BincIMAP only can achieved with **sslserver**, or other servers providing the `UCSPI-TLS` interface thus allowing a 'delayed' TLS negotiation.

sslserver facilitates TLS 1.3 connectivity (TLS 1.0, 1.1 and 1.2 are currently supported as well) and need to be fed with the following minimal crypto material:

- `dhparam` file (at least with 2048 bit size) for the legacy DLog Diffie-Hellman key exchange.
- X.509 `certificate` (in PEM format) identifying the Binc IMAP server and presented to the client.
- X.509 `key` file (in PEM format) used to digitally sign the TLS handshake.

Other optional TLS material can also been provided:

- Location of the X.509 CA file (trust store) containing certificate authorities with their public keys.
- CA path: The path to a set of CA files.
- Cipher list mainly to exclude unwanted TLS ciphers.

sslserver supports client certificate exchange and verification. However, a client certificate does not substitute the authentication of the IMAP users.

2.3.3 Authentication Service

The authentication details in Binc IMAP follow this path:

1. **bincimap-up** is responsible to announce the authentication capabilities. Currently, the following methods are known to Binc IMAP:
   - `LOGIN` – *userid* and *password* in plain text,
   - `AUTHENTICATE PLAIN` – concatenated *userid* and *password* base64 encoded,
   - `AUTHENTICATE LOGIN` – *userid* and *password* each base64 encode, and
   - `AUTHENTICATE CRAM-MD5` – generating the *challenge* and deploying it to the **authenticator**.

Others are subject of the forthcoming releases, in particular `SCRAM` and `ECRAM`.

2. The **authenticator** is a *Pluggable Authentication Module* (PAM) – and not a library – setting up its own Unix context while communicating with the *Identity Provider* (IdP). Given Binc IMAP works in conjunction with *(s/)qmail*, we have as IdP:

   - The **Unix** system with regular Unix accounts (users), there home directory with a *Maildir* (the <`maildepot`>).
   - **VPopMail** as system-wide (virtual) mail user for different domains, each having virtual users here, and only providing <`maildepot`> services.
   - **VMailMgr** acting a (virtual) mail user for a specific domain, again each having virtual users here.

     ↪ It is the duty of the authentication PAM to **chdir** to the user's home directory and to **setuid** to the specific responsible user. In the context of a virtual mail manager, this is one of the user ids given in `/var/qmail/virtualdomains`.

3. The *authentication methods*: Given this scenario, the following choices for an **authenticator** are possible:

   (a) Unix account: Here you can use **checkpassword.pl**.
   (b) *VPopMail*: **vchkpw**.
   (c) *VMailMgr*: **checkvpw**.
   (d) **qmail-authuser** is *s/qmail*'s PAM taking simultaneously care of all these methods.

4. The IdP's *backend* is where the authentication information is stored. Here we have:

   - The Unix password and/or shawdow password file (*/etc/passwd*) with a salted password.
   - A local database of **VPopMail** or **VmailMgr** including the virtual user's and their passwords.
   - A SQL-based database.
   - A LDAP-based database.
   - A plain textfile (typically `/var/qmail/users/authuser`) used for authentication purpose only without providing support for the user's home directory or <`maildepot`> as required for IMAP/POP3.

5. It is important to understand, that Challenge/Response type authentication (`CRAM-MD5`, `APOP`) requires the access to the *plaintext* password to be fetched. With `ECRAM` (Enhanced CRAM) this situation will be relaxed.

6. Binc IMAP announces `PLAIN` and `LOGIN` as authentication method automatically, while `CRAM-MD5` is subject of the settings in the environment variable `BINCIMAP_LOGIN` including `+CRAM-MD5`.

The following table depicts the current authentication possibilities:

Method / Backend	PLAIN	LOGIN	CRAM-MD5	PAM Module
Unix *crypt*	yes	yes	no	**checkpasword.pl**
VMailMgr	yes	yes	no	**checkvpw**
VPopMail	yes	yes	yes	**vchkpw**

Table 2.1: Supported authentication methods of Binc IMAP given different PAMs and backends. Methods LOGIN and PLAIN expect the plain text *userid* and *password* over FD 3; in case of a C/R method, additionally the challenge has to be transferred to the authenticator [see: http://cr.yp.to/checkpwd/interface.html].

↪ *s/qmail*'s **qmail-authuser** can be used as wrapper for all these cases. It enables a site to provide services for local Unix users, **VPopMail**, and **VMailMgr** settings concurrently.

2.3.4 Logging

The logging details in Binc IMAP are set in a section called `Log`. Binc IMAP recognized the following log options:

- **multilog**: Binc IMAP will log to *stderr*, which is the default input for `multilog` (default).
  Alternatively, use the environment variable `LOG_TYPE=multilog` to enable it.

- **syslog**: Binc IMAP will log using *multilog*.
  The environment variable `LOG_TYPE=syslog` can be used to enable this.

If Binc IMAP is configured to log using *syslog*, it will log using the *syslog* facility described via the environment variable `LOG_USER`. Here you can tailor the logging by the following environment variables:

- The syslog facility: `SYSLOG_FACILITY=LOG_DAEMON` or perhaps `SYSLOG_FACILITY=LOG_MAIL`.

- Up to eight tags embedded in the environment variables `LOG_LOCAL0` to `LOG_LOCAL7` can be applied for a customized logging.

2.3.5 Security

The current Binc IMAP does not provide a 'jailing' service as previous versions did. Thus it is depending on the invoking application like **sslserver** or **tcpserver** to provide a 'secure' (say setuid) environment.

Both, tcpserver and sslserver provide these means to run under a dedicated and low privileged user. In case other network servers are considered, **envuidgid** from the *Daemontools* package would provide a comparable solution.

Lets consider an example given a run script:

```
#!/bin/sh
exec 2>&1
HOSTNAME=‘hostname‘
BINCU=‘id -u qmaild‘
BINCG=‘id -g qmaild‘
. /var/qmail/ssl/ssl.env
  exec envdir ./env sslserver -seV -Rp -l $HOSTNAME -u $BINCU -g $BINCG :0 imaps\
       bincimap-up -- /var/qmail/bin/qmail-authuser bincimapd Maildir
```

1. **sslserver** is started with *root* privileges binding to port 993 (imaps).

2. **sslserver** drops its privileges and **bincimap-up** is running with the user/group credentials given as *$BINCU* and *$BINCG*.

3. Upon successful invocation of **qmail-authuser** and subsequent authentication, the effective user is the (Unix) *owner* of the `Maildir`.

↪ In order to achieve the last step, the authentication module needs to be 'sticky'. In case of a Unix authentication is has to belong to root; in case of a virtual domain manager it has (at least) to belong to the respective user.

2.3.6 Maildepot settings

Binc IMAP uses the term '`maildepot`' to reference one of the following:

1. A `Maildir++` directory.

2. A `IMAPDdir` directory.

For those, the following settings can be performed:

- `DEPOT = <depottype>`
  Sets the type of depot that Binc IMAP should use and how the contents of an IMAP user's mail path is interpreted. Currently, the two supported depot types are
  - `Maildir++` (default),
  - `IMAPdir`.
- `DELIMITER = <char>`
  allows to instruct Binc IMAP to obey the structure of an existing `maildepot` created by other IMAP servers like Courier. Nested mail directories are organized as concatenated sub directory names. The 'binding' character is called 'delimiter'.
  - `DELIMITER=/` (default)
  - `DELIMITER=.`
- `<path>`
  The <path> is given as first argument to **bincimapd**. It should be given as `Maildir` in most cases without leading character like '`./`' resulting in `./Maildir`. Also a trailing '`/`' shall be omitted.
  Some authentication modules like **qmail-authuser** only perform a **setuid** to the user, if `Maildir` is used.

Chapter 3

The Binc IMAP Depot

The main role of an IMAP server is to give email clients an interface through which they can authenticate and then access email and mailboxes located on a (remote) mail server. The mails are stored on the mail servers in a special structure, and this structure varies from system to system.

Most often we distinguish between the mailbox *structure* (or *hierarchy structure*) and its *format*. The structure determines how the different mailboxes and submailboxes are stored, and the format decides how each and every email is stored within one mailbox.

Binc IMAP uses the generic container object *Depot* to describe the map between the hierarchy structure and its translation to selectable mailboxes in IMAP. The Depot has two specializations: one for IMAPdir and one for Maildir++.

3.1 Maildir & mbox

Maildir

A `Maildir` is by Dan J. Bernstein's definition identified by a directory that contains the subdirectories

- `cur`,
- `new`,
- `tmp`

and nothing else.

mbox

When using the `mbox` storage format, the user's `INBOX` is typically stored at `/var/spool/mail/<username>`. Using this format, the mail depository client must both have the path to the user's INBOX and the path to the user's local mailbox depository, typically `mail/`.

3.2 Maildir++

The `Maildir++` definition follows naturally from `Maildir++` being an extension to Dan J. Bernstein's `Maildir` format. However, although the directory `/Maildir/` itself is a standard representation of `INBOX` for Maildir clients, it is not standard for other mailbox formats. With `Maildir++`, your mailboxes and INBOX in particular must be a `Maildir`.

Courier-IMAP defines the hierarchy structure `Maildir++`, which provides a way for existing Maildir users (Maildir is a mailbox format) to have multiple mailboxes and submailboxes inside the directory that contains the default mailbox "INBOX" (which often resides in `~/Maildir` for each UNIX user).

Read more about the Maildir++ format at the following location:

- http://www.inter7.com/courierimap/README.maildirquota.html

Binc IMAP supports this mailbox structure with a few limitations and a few enhancements:

- The *maildirfolder* file is not created inside each `Maildir` submailbox. The reason for this is that this only works with mailbox formats that store a mailbox inside a directory, such as Maildir.
- `Maildir++` quotas are not supported.
- No `Maildir++` restrictions to mailbox names apply (such as `.Trash`).
- Mailboxes inside a `Maildir++` structure can be of any format, not just `Maildir`.

3.3 IMAPdir

`IMAPdir` is Binc IMAP's native mailbox structure. It is open and usable for most existing local mail clients. Some architectural designs decisions:

- `IMAPdir` does not extend/change any mailbox formats; it merely defines a way to describe mailboxes and submailboxes in a way that is suitable for an IMAP server, with as few restrictions as necessary.
- The goal of the work behind this specification is to provide the community with an unambiguous representation of a mailbox hierarchy where a mailbox name has a one-to-one match against a file system representation. The hierarchy and naming style is inspired by the IMAP4 protocol.
- Rather than being a completely new mailbox format, this specification sets conventions on how to represent a mailbox hierarchy on a file system, using existing mailbox formats. `IMAPdir` is not bound to any protocol.

Note that although `IMAPdir` has no restrictions with regards to mailbox names, the protocol that uses `IMAPdir` might. For example, IMAP servers will require the mailbox `INBOX` to be present.

`IMAPdir` works with any mailbox format where one mailbox can be identified by a file, a directory or a symbolic link. One entry in an `IMAPdir` folder is a candidate for a mailbox.

If the `IMAPdir` client can not identify a directory entry as a selectable mailbox, then the client must either skip the entry or mark it as invalid (in IMAP, marked as \NoSelect).

- There is no limitation to the type of file system or the number of file systems represented inside an `IMAPdir`.
- There are no reserved ordinary folder names such as `"Sent"`, `"Draft"` or `"Trash"`.
- Clients of the mailboxes inside an `IMAPdir` folder must follow the respective format and protocol conventions strictly.

The format of a mailbox representation in `IMAPdir` is a sequence of one or more US-ASCII characters (32-126), encoded using the following rules:

- A dot '.' character represents a soft hierarchy delimiter with two exceptions:
  - A leading dot represents the dot itself.
  - A dot '.' preceded by a backslash '\' represents the dot '.' itself.
- A backslash '\' preceded by a backslash '\' represents the backslash '\' itself. For all other cases than before a dot '.' or a backslash '\', a stray backslash '\' character is considered an error.
- A backslash '\' as the first character of an entry is considered an error.
- All other characters represent themselves.

Note that the protocol used to fetch the mailbox using the structuring `IMAPdir` convention may restrict the character set allowed. The clients must in those cases translate the mailbox names to a selectable format.

As with `Maildir++`, submailboxes can not be represented in a recursive fashion in the file system. The mailbox' representation name will contain the soft hierarchy delimiter character dot '.', and all mailboxes must reside in the same root level directory.

IMAPdir mapping to Maildir and mbox

The following example shows the typical content of an `IMAPdir` stored under the directory `mail/`. The file system column displays the contents as viewed by the UNIX command '`ls -alF`'.

File System	IMAP	Description
mail/INBOX -> /var/mail/paul	"INBOX"	Symbolic link to mbox
mail/INBOX.old/ -> ../Maildir/	"INBOX/old"	Symbolic link to Maildir
mail/INBOX.outbox/	"INBOX/outbox"	Maildir
mail/work	"work"	mbox
mail/3rd. of July	"3rd. of July"	mbox
mail/Sent.2003.Jan/	"Sent/2003/Jan"	Maildir
mail/Sent.2003.Feb/	"Sent/2003/Feb"	Maildir
mail/Sent.2003.Mar/	"Sent/2003/Mar"	Maildir
mail/.foo	".foo"	mbox

In other multi level mailbox formats, INBOX is treated as a special case.

IMAPdir $\longleftrightarrow$ Maildir conversion

Binc IMAP includes the following Perls convenient scripts suited to realize a bi-directional conversion between the `IMAPdir` and `Maildir++` format:

- **IMAPdir2Maildir++.pl** (by Henry Baragar)
- **Maildir++2IMAPdir.pl** (by Henry Baragar)
- **tomaildir++.pl**: Adding all Maildir entries to the `.subscribed` file
- **toimapdir.pl**: Adding all IMAPdir entries to the `.subscribed` file

Those scripts are shipped without verification checks.

Chapter 4

Binc IMAP - Technical Documentation

Binc IMAP uses either `Maildir++` or a structure called `IMAPdir` to store its set of mailboxes. `IMAPdir` is more or less similar to `Maildir++`, but it provides more flexibility with regards to mailbox names and hierarchy structure.

In a sense, `IMAPdir` takes all the goods from `Maildir` and adds root level mailboxes, submailboxes both of regular root level mailboxes and of the special mailbox `INBOX`, mail in mailboxes of any level, and with no restrictions.

In the root of the `IMAPdir` structure, Binc IMAP stores the list of a user's subscribed folders in a file called `.subscribed`. This file should only be edited manually if you are confident with `Binc::Storage`. Normally the administrator and the IMAP user will leave this to Binc IMAP.

Binc IMAP's `Maildir` backend (default) will temporarily create a lock file called `bincimap-scan-lock` inside a `Maildir` when it is scanning for mailbox changes and delegating unique message identifiers. This is to ensure that `UIDs` are delegated exactly once to every message that has been detected by any one *Binc IMAP* server instance.

Inside each `Maildir`, Binc IMAP stores two files that allow multiple instances of the server to communicate the state and changes of the mailbox:

- `bincimap-uidvalidity` and
- `bincimap-cache`.

These files are safe to delete, although that will trigger `UIDVALIDITY` to bounce and clients may have to resynchronize their local state.

4.1 Object Oriented Design: Brokers, Depots, Operators, IO

I's design is simple and modular. This makes it easy to maintain and extend.

Although the IMAP protocol is relatively complex, you will find that *Binc IMAP*'s solution is surprisingly easy to grasp.

At the heart of *Binc IMAP*'s implementation lies the basic functionality for Object Oriented Design provided by the ISO C++ standard and general knowledge in the area of standard Design Patterns.

The main design components are:

- Architecture:
  - The Broker
  - The BrokerFactory
- Behavior:
  - The Operator
  - The Command
- Repository:
  - The Depot
  - The DepotFactory
  - The Mailbox
- The IO
- The Session

4.2 Broker

One *Broker* holds a set of Operators. For each state *Binc IMAP* is in, the *BrokerFactory* delegates exactly one *Broker* to hold the relevant `Operator` objects.

Typically, an `Operator` can be assigned to more than one *Broker*. For example, the `Operator` that serves the IMAP command `"NOOP"` (a command that is available in all three IMAP states), `NoopOperator`, is available in all Broker objects.

The *Broker* is responsible for first passing the Depot and the IO *singleton* to the appropriate IMAP command, generating a `Operator` object.

The *Broker* is also responsible for passing the resulting `Operator` object to the Operator together with the Depot, generating the untagged responses that come as a result of the processing.

```
Operator *Broker::get(const string &name) const
{
  if (operators.find(name) == operators.end()) return 0;

  return operators.find(name)->second;
}
```

4.2.1 BrokerFactory

The *BrokerFactory* knows three *states*:

- `NONEAUTHENTICATED` (realized by **bincimap-up**), where only generic IMAP commands like "`CAPABILITY`" and "`ID`" are honored (and or course the potentially following "`STARTTLS`" and "`AUTHENTICATE`" commands) are honored.
- `AUTHENTICATED` allowing the access to the Depot of the authenticated userid.
- `SELECTED` allowing to perform operations on mails like "`STORE`".

The *BrokerFactory* manages the *Broker* objects:

- Given a state, the *BrokerFactory* provides a *Broker* that holds all the Operator objects available to the client.
- This provides a modular and safe separation of the privileges available at the different states in the IMAP session.
- The *preauthenticate* stub has a *BrokerFactory* that can only generate Broker objects for the non-authenticated state.

```
Broker *BrokerFactory::getBroker(int state)
{
  if (brokers.find(state) == brokers.end()) {
    setLastError("No appropriate broker for state.");
    return 0;
  }
  return brokers[state];
}
```

4.3 Operators

An Operator is associated with an IMAP command such as `"SEARCH"` or `"AUTHENTICATE"`. In short, the Operator is used to perform an arbitrary operation on a Mailbox.

- Typically, an Operator can be assigned to one or more Broker objects.
- Operators contain, among others, the two public methods: *parse()* and *process()*.
- When given the IO singleton as input, the parse() method generates a Command object. This object can then be fed to process() together with a Depot.
- When processing its command, an Operator is allowed to generate untagged responses and it can also update the state of a Mailbox, the Depot or the Session *singleton*.

Operator objects are assigned dynamically to each Broker, making it very easy to write extensions that add or replace existing Operator objects using Binc IMAP's loadable module support.

4.3.1 Command

A `Command` object holds all information that was passed to the `Operator` that served a specific IMAP command.

- `Command` objects are named. Examples of such names are `"CHECK"`, `"SUBSCRIBE"` and `"LOGOUT"`.
- For the name `"FETCH"`, the Command object is decorated with sequence set, optionally a section and so on. The parse() method in each Operator is responsible for decorating the Command object.

The Command object is short-lived. It is created, decorated, passed on to the Operator, then discarded.

4.4 Depot

A *Depot* is responsible for handling the different `Mailbox` objects, and it is the mailbox structure authority.

- Given an IMAP mailbox path as input, a Depot can give the caller a corresponding `Mailbox` object if it finds one that successfully identifies the type of Mailbox.
- The Depot is also aware of what the default `Mailbox` type object is. This `Mailbox` object is used when creating new IMAP mailboxes.

- Finally, the `Depot` is used to translate mailbox names to a representation on the file system and back. There are currently two specializations of the `Depot` object available: one for `Maildir++` and one for `IMAPdir`. Each has its own characteristics in how do translate the mailbox hierarchy to the file system.

```
Mailbox *mailbox = depot.getSelected();
if (mailbox != 0) {
  mailbox->closeMailbox();
  depot.resetSelected();
  mailbox = 0;
}
```

4.4.1 DepotFactory

The `DepotFactory` manages the `Depot` objects.

- New Depot objects are assigned to the `DepotFactory` in runtime. This makes it easy to add new Depot objects using loadable modules.
- The Depot objects are registered and accessed via their names, such as `"Maildir++"` or `"IMAPdir"`.
- The `DepotFactory` gives individual users of Binc IMAP the option to choose the Depot object that suits their needs the best.

```
if ((depot = depotfactory.get(depottype)) == 0) {
  bincLog << "bincimapd: pid " << pid
          << " Found no Depot for: " << depottype
          << ". Please check your configurations file under the Mailbox section\n";
  bincLog.flush();
  return false;
}
```

4.4.2 Mailbox

The `Mailbox` is an abstraction for Binc IMAP's different backends.

- Bundled with Binc is a backend for `Maildir`. The class `Maildir` inherits `Mailbox`.
- In short, a Mailbox contains all methods needed for Binc IMAP to serve a specific backend. It also holds a method to identify a Mailbox of its own kind.
- All registered Mailbox objects are held by the `Depot`.

4.5 IOFactory

The IOFactory makes use of thee different streaming devices:

- `StdIODevice` – responsible for reading and writing to the network (`FD 0/1`).
- `MultilogDevice` – used for the logging to `FD2`. Here, you heed to define the `LOG_TYPE=multilog` upon start.
- `SyslogDevice` – in charge for logging to the *Syslog Facility*. This is the default device for logging. Alternatively, you can set `LOG_TYPE=syslog`.

The buffer sizes (in bytes) for `StdIODevice` are defined in `globals.h`:

- `TRANSFER_BUFFER_SIZE` = 1024;
- `INPUT_BUFFER_LIMIT` = 8192;

4.5.1 Logging information

bincimap-up and **bincimapd** are using different prefixes for logging but common is the display of the `<PID>`:

- **bincimap-up** records IP addresses and user names. Both information are fetched from the environment via `TCPREMOTEIP` and `USER`.
- **bincimap** logs user name, number of IMAP statements, and number of read in writes (in byte).

4.5.2 Syslog Device

In order to customize the *syslog* feeding, the following environment variables can be defined:

- `LOG_USER=<loguser>`, the default is `LOG_DAEMON`.
- `LOG_LOCAL0=<...>` to `LOG_LOCAL7=<...>`.

It should be noted, that in case of **syslog** logging, it is possible to define in the context of the virtual user (for **bincimapd**) separate different settings using using specific `LOG_USER` and `LOCAL` variables for each domain.

For a qualified *syslog* logging, the environment variable `TCPLOCALHOST` is required and used.

4.5.3 Protocol Dumping

The analysis of the IMAP protocol flow can be analyzed be means of a protocol dump: Use then environment variable `PROTOCOLDUMP=yes` and Binc IMAP will record the entire IMAP session to a file:

- `/tmp/bincimap-dump-<number>-<client-ip>-<random>`

Beware, that this for debugging and investigation only! Passwords are also recorded! The generated files could be quite large.

4.6 Session

The Session is a *singleton* object that holds information that is relevant to the current IMAP session. Currently, the Session contains information about:

- Global configuration (administrator settings)
- Local configuration (user settings)
- Command line arguments
- Folder option list

4.6.1 Session Parameters

The session details in Binc IMAP are now hard-coded and set up in the header file `globals.h`.

Session timeouts (in seconds):

- `IDLE_TIMEOUT` = 30*60;
- `AUTH_TIMEOUT` = 60;
- `AUTH_PENALTY` = 5; (factor)
- `TRANSFER_TIMEOUT` = 20*60;

4.6.2 Checkpassword Interface

The checkpassword interface

```
checkpassword prog
```

`checkpassword` reads descriptor `3` through end of file and then closes descriptor `3`. There must be at most 512 bytes of data before end of file.

Call Interface

The information supplied on descriptor 3 is

- a *login name* terminated by \0,
- a *password* terminated by \0
- a *timestamp* terminated by \0,

and possibly more data. There are no other restrictions on the form of the *login name*, *password*, and *timestamp*.

Exit Codes

- If the *password* is unacceptable, `checkpassword` exits `1`.
- If `checkpassword` is misused, it may instead exit `2`.
- If there is a temporary problem checking the password, `checkpassword` exits `111`.

Child called

If the *password* is acceptable, `checkpassword` runs `prog`. `prog` consists of one or more arguments.

Compatible tools

There are other tools that offer the same interface as `checkpassword`. Applications that use checkpassword are encouraged to take the `checkpassword` name as an argument, so that they can be used with different tools.

Note that these tools do not follow the `getopt` interface. Optional features are controlled through (1) the tool name and (2) environment variables.

The password database

`checkpassword` checks the login name and password against `/etc/passwd`, using the operating system's `getpwnam` and *crypt* functions, supplemented by `getuserpw` and `getspnam` if necessary. It rejects accounts with empty *passwords*. It ignores the *timestamp*.

Other `checkpassword`-compatible tools have different interpretations of *login names*, *passwords*, and *timestamps*. Both the *login name* and the *password* should be treated as secrets by the application calling `checkpassword`; the only distinction is for administrative convenience. The *timestamp* should include any other information that the password is based on; for example, the *challenge* in a *challenge-response* system such as APOP.

WARNING: **getpwnam** is inherently unreliable. It fails to distinguish between temporary errors and nonexistent users. Future versions of *getpwnam* should return `ETXTBSY` to indicate temporary errors and `ESRCH` to indicate nonexistent users.

Process-state changes

Before invoking prog, checkpassword sets up `$USER`, `$HOME`, `$SHELL`, its *supplementary groups*, its *gid*, its *uid*, and its *working directory*.

Other `checkpassword`-compatible tools may make different changes to the process state. It is crucial for these effects to be documented; different applications have different requirements.

Taken from: https://cr.yp.to/checkpwd/interface.html