Af

An advanced. emacs-like mail reader and composer
Edition 1.4, for Af version 2.4
August 2002

by Malc Arnold

Preface

This manual documents the use and simple customisation of the af mail reader and composer. The reader is not expected to be a programmer; simple customisations do not require programming skills. If you are not interested in customising then you can ignore the customisation hints. This manual is primarily intended as a reference manual, but can also be used as a primer.

To find the documentation on a particular command, look in the index. Keys (character commands) and command names have separate indexes. There is also a glossary, with a cross reference for each term.

This manual is available as a printed book and also as an Info file, which can be browsed on-line using either GNU Emacs or the info program from the GNU project. The Info file and the printed book contain substantially the same text and are generated from the same source files, which are distributed along with af.

Distribution

Af is *free software*; this means that everyone is free to use it and free to redistribute it on certain conditions. Af is not in the public domain; it is copyrighted and there are restrictions on its distribution, but these restrictions are designed to permit everything that a good cooperating citizen would want to do. What is not allowed is to try to prevent others from further sharing any version of af that they might get from you. The precise conditions are found in the GNU General Public License that comes with af and also appears following this section.

The simplest way to get a copy of af is from someone else who has it. You need not ask for permission to do so, or tell anyone you have done so; just copy it. If you have access to the Internet, you can get the latest version, or a patch to upgrade to the latest version, from the Af Home Page on the World Wide Web '`http://www.thing.demon.co.uk/af`', or by anonymous FTP from '`ftp://ftp.csv.warwick.ac.uk/pub/mail/af`'.

GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software— to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   a. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   b. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   c. If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

   a. Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   b. Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   c. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

   The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

   If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program 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.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.
7. If, as a consequence of a court judgement or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

   It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

   This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.
8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.
9. The Free Software Foundation may publish revised and/or new versions of the General Public 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.

   Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.
10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software

which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.
12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
one line to give the program's name and an idea of what it does.
Copyright (C) 19yy  name of author

This program is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License
as published by the Free Software Foundation; either version 2
of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
```

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
Gnomovision version 69, Copyright (C) 19yy name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details
type `show w'.  This is free software, and you are welcome
to redistribute it under certain conditions; type `show c'
for details.
```

The hypothetical commands '`show w`' and '`show c`' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than '`show w`' and '`show c`'; they could even be mouse-clicks or menu items—whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright
interest in the program `Gnomovision'
(which makes passes at compilers) written
by James Hacker.

signature of Ty Coon, 1 April 1989
Ty Coon, President of Vice
```

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

Introduction

You are reading about af, an advanced, self-documenting, customisable, real-time display-oriented mail reader and composer. If you think that sounds rather like the Emacs editor, you're correct; af is designed to follow the Emacs paradigm as closely as possible.

We say that af is *display-oriented* because normally the messages being read are visible on the screen and are updated automatically as you type your commands. See Chapter 2 [Screen], page 20.

We call it a *real-time* mail reader because the display is updated very frequently, usually after each character or pair of characters you type. This minimises the amount of information you must keep in your head as you process your mail. See Chapter 5 [Basics], page 31.

We call af advanced because it provides facilities that go beyond simple reading of messages: viewing multiple mail folders at once, searching and sorting mailboxes, and dealing with groups of messages in one operation.

Similarly, we have tried to take care that af complies with any relevant standards: it conforms closely to RFC822, the Internet Standard for Mail Messages, and will work with such standard Mail Transfer Agents as UUCP, sendmail, and MMDF; and supports the POP3 protocol for reading mail from a remote server. Af also has good support for reading MIME mail, and limited support for composingl MIME mail, although this is still being improved.[1].

Self-documenting means that help about af is available from within af itself. You can find out what any command does, or find all the commands that are relevant to a topic. See Chapter 10 [Help], page 51.

Customisable means that you can change the definitions of af commands in little ways. For example, if you prefer to include the text of the original message into the text of a reply to that message, then you can tell af to do so. Another sort of customisation is rearrangement of the command set. For example, if you prefer the four basic cursor motion commands (up, down, left and right) on keys in a diamond pattern on the keyboard, you can rebind the keys that way. See Chapter 21 [Customisation], page 82.

[1] Don't worry if you don't understand this; you don't need to know about these things to use af

Acknowledgements

Several people have contributed to af's development. Here's my chance to thank them for their efforts, and for an attempt at a moment of glory.

Malc Arnold (malc@thing.demon.co.uk) conceived af, did the initial design work, wrote almost all of the code, and argued passionately in defence of the interface.

Kay Dekker (kay@vide.coventry.ac.uk) did a vast amount of design work, discussing and arguing about features I should support and how the interface should work. More af features than I can readily recall were originally Kay's idea. He also wrote the manual pages and the first draft of the user manual, and showed great patience when struggling with some of the less stable versions of af.

Ian "Vato" Dickinson (idickins@fore.com) actually suggested turning af into a full-featured mailer, and had a wealth of suggestions for features, some of which I've actually implemented. He also wrote the first draught of the POP3 and SMTP support code.

Andrew "Vic" Fry (vic@pootug.demon.co.uk) has contributed insights about the af interface, as well as enough encouragement and coffee to keep me going. He has also ported and tested several versions of af.

Justin Murdock (justin@vide.coventry.ac.uk), Dave Berry (daveb@harlequin.co.uk), Iain Bowen (alaric@harlech.demon.co.uk), and Onno van der Linden (onno@simplex.nl) have reported many bugs and suggestions for improving af.

Emma Kemm (emma@thing.demon.co.uk) and Iain Bowen (alaric@harlech.demon.co.uk) proofread the manuals and supplied more much-needed moral support.

Richard Stallman (rms@gnu.mit.edu) wrote the eleventh edition of the GNU Emacs manual, from which many parts of this manual have been derived. Thanks to the Free Software Foundation for permission to distribute this manual as a "derived work".

Note that this does not imply any endorsement of af by the Free Software Foundation.

Bugs

Sometimes you will run into a bug in af. Although I can't promise that I can or will fix the bug, and I may not even agree that it is a bug, I want to hear about problems you encounter. Often I will agree that they are bugs and want to fix them.

If af should ever crash (ie fall over with an operating system error message), or exit back to the shell without warning, then it is certainly a bug. Commands doing the "wrong thing" are also bugs, but you should check the manual and help entry for the command carefully to be sure that the command isn't doing what it is supposed to.

If you think that you've found a bug, it is important to report it, and to provide enough information with the report to be useful. The most useful kind of bug report (except the fix for it) is an exact description of what commands you type, from entering af until the bug manifests. It may also help if you could include a small folder which triggers the bug, since many af bugs can be related to the data in the current folders. It is also important to tell me which version of af you are using, and what machine it is running on. You can find out which version of af you are using by typing *`M-x af-version`* ⟨RET⟩.

At present, I am also interested in any feedback on af. If you have comments, praise, criticisms or complaints, then I'd like to hear it.

The best way to send bug reports or comments is to mail them electronically to the af maintainer at '`af-bug@csv.warwick.ac.uk`'. I don't promise to fix the problem; but if I agree that its a bug then I'll most likely want to fix it. And remember that the clearer your bug report is, the more likely it is that the bug will get fixed quickly, or indeed at all.

Future Developments

Af has now been released, but this doesn't mean that it is complete. There are several major enhancements that af could benefit from, some of which are loosely planned to be done at some time in the future.

- Internal support for composing non-textual MIME mail.
- Commands to allow you to use PGP conveniently with af. PGP stands for Pretty Good Privacy; it is a public-key encryption package written by Phil Zimmerman.
- Support for internationalising af, so that messages are displayed in the user's native language. I would need volunteers to actually do the translations.
- Major enhancements to the Afl language, allowing people to write their own extensions to af.
- An X interface with pull-down menus, scroll bars and mouse support.
- An additional major mode for reading news, similar to the mail mode but carefully tailored for convenient news-reading.
- Ports to other operating systems, such as MS-DOS or windows.

If you have any strong preferences about the direction af should take in the future, or suggestions of other enhancements you think I might like to consider, then please mail them to me at '`af-bug@csv.warwick.ac.uk`'.

1 An Introduction to Electronic Mail

Electronic mail (often known as *e-mail*, or simply *mail*), is the exchange of messages between people using computer networks. In this manual, when we say mail, we mean electronic mail. A *mail message* is a message which has been or will be sent via electronic mail. Again, when we say message in this manual, we are referring to a mail message.

When a mail message is sent, it is transmitted from one computer to another until it reaches the correct one. A message can be addressed to more than one person, in which case a separate copy of the message is sent to each person. Once a message has reached the machine to which it was addressed, then it is usually stored in the incoming mailbox of the person it is addressed to.

An *incoming mailbox*, or *mailbox* is simply a file containing mail messages, with a special sequence of characters separating each message from the others. Each user of a system will normally have their own mailbox. Users can also store messages in files of their own, which are called *folders*. In this manual, the term folder will refer to any file which contains mail messages.

To read your mail, you will usually use a *mail reader*, a program which can understand the format of a folder and present the contents to you in a convenient form. Af is an example of a mail reader. Similarly, you would normally send mail by using a *mail composer*, a program which knows how to construct and send a mail message, and gives the user a convenient environment in which to compose and send their message. Most mail readers, including af, are also mail composers.

To allow a mail message to be sent to the correct person, it needs an *envelope*. To supply this, a mail message is divided up into two parts, the *headers* and the *body*. The headers supply the information required to send the message to the right people, while the body is the actual message you wanted to send.

1.1 Mail Headers

The headers of a mail message are held at the beginning of the message, and are separated from the message body by a blank line. Here's an example of message headers:

```
From malc@thing.demon.co.uk Tue Jan 23 16:20:39 1996
Date: Tue, 23 Jan 1996 16:20:36 GMT
Message-Id: <4780.199201201539@thing.demon.co.uk>
From: Malc Arnold <malc@thing.demon.co.uk>
Organization: Team Limpid
Sender: The Management <root@thing.demon.co.uk>
To: Kay Dekker <kay@thing.demon.co.uk>,
    Andrew Fry <vic@pootug.demon.co.uk>
Subject: Af manual in progress
MIME-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit
```

Note that apart from the first line, each header consists of a header name, followed by a colon and then the header text. The first line is not really a header at all; it is the marker that most Unix systems insert to mark the start of each new message. You can also see that unlike the envelope for a postal letter, the envelope information contained in the header can also be of interest to the recipient. There are actually several other headers contained in most mail messages, but by default af will filter them out when it displays a message for you to read (see Section 5.1 [Reading Messages], page 31).

Another interesting feature is the '`To:`' header, which has been extended over two lines by starting the second line with a tab. You can also continue headers in the same way by starting the following line with a space. Probably it's best to try to keep headers to a single line though; if a long header line needs to be *folded* like this, then af will normally do so for you.

Here is a brief summary of some of the headers you are likely to see in a mail message.

'`From:`' The e-mail address of the person who the mail is from. Replies to the message will be sent to this address (see Chapter 8 [Sending Mail], page 41). It is possible for a message to be from more than one person.

'`Organization:`'
: The organisation the sender belongs to.

'`Subject:`'
: A brief description of what the message is about.

'`To:`' The e-mail addresses of the message's primary recipients.

'`Cc:`' A "carbon copy" of the message was sent to the addresses listed. People listed in the '`Cc:`' header should regard the mail as being a copy for informational purposes.

'`MIME-Version:`'
: Indicates that the message is a MIME message. See Section 1.3 [Message Body], page 16.

'`Content-Type:`'
: What type of data the message contains. See Section 1.3.2 [Content Type], page 16.

'`Content-Transfer-Encoding:`'
: How the data has been encoded for transport via mail. See Section 1.3.4 [Transfer Encoding], page 18.

1.2 Mail Addresses

Mail addresses usually take the form '`user@full.domain.name`'. They may also be written as '`Full Name <user@full.domain.name>`', or '`user@full.domain.name (Full Name)`'. In all of these cases, the address of the user is the same; the addition of the full name is simply a convenience to allow mail readers to display the person's full name rather then their e-mail address.

Usually, the part of the address to the left of the "`@`" character ('`user`') identifies the user to which the mail is to be sent, while the part to the right ('`full.domain.name`') identifies the specific computer or organisation where that user can be found. Note that while mail addresses must be unique, a user name may be duplicated many times by different organisations; each with a different full domain name.

Lists of addresses (such as in a '`To:`' header) should be separated either by spaces or commas.

We shall not go into the intimate details of mail address syntax here; af will check any addresses you give it, and either correct minor problems or report any serious errors in an address.

1.3 The Message Body

Normally, the body of a mail message is just text, which is not handled in any special way. Recently, electronic mail has been extended to allow for a message body which is something other than text. These extensions are called MIME (Multipurpose Internet Mail Enhancements), and mail which makes use of them is called MIME *mail*.

A MIME message uses the '`Content-Type:`' header to define the type of data the message contains. The '`Content-Transfer-Encoding:`' header describes how the data was encoded in order to be sent via mail. The '`Content-Disposition:`' header describes how a message should be handled when read. Finally the '`Content-Description:`' header can be used to describe the contents of a message.

1.3.1 Body Parts

A MIME message body may consist of more than one part. Each of these *body parts* is like a miniature MIME message in its own right, and has its own headers specifying the MIME details.

For example, a message might contain two body parts. The first is a description of the work you've been doing; the second is an image displaying the results of that work. You can see that the two parts of the message are related, but they must go into different body parts since they contain a different type of data.

Since we can think of a normal message as having one body part, when describing MIME mail it is conventional to talk about body parts rather than the message body. We will follow this convention in this manual.

1.3.2 Content Type

The body of a MIME mail message may be textual, or it may be an image or some other non-textual material. The type of the message body is specified by the '`Content-Type:`' header.

The form of a '`Content-Type:`' header is something like:

`Content-Type:` *type*`/`*subtype* `[;` *parameter*`=`*value* `... ]`

The main *type* is separated from a *subtype* by a slash. The content type may be further refined by *parameters*, which are separated from the type and subtype, and from each other, by semicolons.

The content types you are likely to see in mail messages are:

'`multipart`'
: The message body consists of a set of body parts. The '`mixed`' subtype indicates that the body parts may contain any content type. The '`digest`' subtype

indicates that each body part is an entire mail message, which may itself be in MIME format. The 'alternative' subtype indicates that each body part contains a different form of the same data, and only the "best" should be displayed. The 'parallel' subtype indicates that the body parts should be displayed simultaneously if possible. Af can handle multipart messages internally.

'message' The body part is itself a MIME message. The 'rfc822' subtype indicates that the body part is an entire MIME message. Af is capable of handling such *encapsulated* messages internally.

The 'partial' subtype indicates that the original message has been split into a number of smaller messages, presumably to avoid limits on the size of mail messages. Af can rebuild the original message from such *partial messages*, if all the parts of the original message have arrived intact.

'text' The body part is textual. Although it may contain formatting information, it should be readable as it stands. The 'plain' subtype means there is no formatting information; the text should be displayed as it stands.

The 'charset' parameter indicates the character set the text is written in. Common values are 'us-ascii' for plain ASCII text, and 'iso-8859-1', which is suitable for Western European languages. See Section 1.3.3 [Character Sets], page 17.

'image' The body part is an image, in a format specified by the subtype, such as 'image/gif' or 'image/jpeg'. Af will call an external program to display 'image' body parts.

'audio' The body part is audio data, in a format specified by the subtype. Af will call an external program to display 'audio' body parts.

'video' The body part is video data, in a format specified by the subtype. Af will call an external program to display 'video' body parts.

'application'
The body part is raw data for some application, often specified by the subtype. One common value is 'application/octet-stream', which indicates that the body part contains raw data with no meaning attached. Hopefully the context will make it clear what you are to do with the data. Af will call an external program to display 'application' body parts.

1.3.3 Character Sets

One of the commonest uses of MIME (in Europe at least) is to send mail which contains international characters (such as accents, or Cyrillic characters) which aren't in the us-ascii character set.

To allow these characters in a message, textual MIME messages can specify which character set they are written in with a 'charset' parameter. The character set may be either 'us-ascii' (the default), or one of the 'iso-8859' character sets.

If your display is capable of displaying these character sets properly, Af can be set up so that it display's such messages properly by setting the variable viewable-charsets (see Section 21.4 [Variables], page 85).

The person who installed af at your site may have already configured it to handle messages in the available character sets. In this case you don't need to do anything to make af handle internationalised messages correctly.

Sadly, none of these character sets is capable of displaying all the characters used in all the world's languages. So you may sometimes receive textual messages which you cannot read. The ISO-8859-1 character set, which is suitable for most Western European languages, is by far the most common character set used in MIME mail.

1.3.4 Transfer Encoding

Traditionally, the body of a mail message had to contain only plain text consisting of only ASCII characters, and much of the infrastructure which supports mail is only capable of transmitting such messages. For this reason, it may have been necessary to encode the contents of the message before they were send via mail. This encoding is specified by the '`Content-Transfer-Encoding:`' header.

The encodings that you may encounter in a MIME body part are:

'`7bit`'
: The body part contains only short lines of 7 bit ASCII characters. No encoding has been applied.

'`8bit`'
: The body part contains only short lines of 8 bit characters, not including the null character ASCII NUL. No encoding has been applied.

'`binary`'
: The body part contains binary data, but has not been encoded. The electronic mail infrastructure is such that this encoding is unlikely to work.

'`quoted-printable`'
: The body part has been encoded with the '`quoted-printable`' encoding. This encoding leaves ASCII characters untouched, but replaces international characters with sequences beginning with '`=`'. It is intended for encoding text, since many characters will still be readable without decoding the message.

'`base64`'
: The body part has been encoded with the '`base64`' encoding. This renders the body part completely unreadable, but is more compact than '`quoted-printable`' if the message contains a high proportion of non-ASCII characters. It is suitable for encoding binary data, such as images.

'`x-uuencode`'
'`x-uue`'
: These encodings are not part of MIME at all. They represent the Unix '`uuencode`' encoding, which predates MIME. Some people send messages marked in this format, and af is able to handle them. These encodings are similar to '`base64`', but is not as robust. It is a bad idea to send messages encoded in this way except by prior arrangement with the recipient.

1.3.5 Content Disposition

Sometimes a body part isn't primarily intended to be read. It might be a program source file, or a configuration file. The '`Content-Disposition:`' header specifies whether a body part should be displayed *inline* when the message is read, or that it is an *attachment*, which requires confirmation before it is displayed.

You can also specify a file name in the '`Content-Disposition:`' header. Again, this is a hint to the recipient's mail reader that the body part should be stored in the named file if it is saved.

The form of a '`Content-Disposition:`' header is something like:

`Content-Disposition:` *disposition* `[; filename=`*file* `... ]`

1.3.6 Content Description

Sometimes it can be useful to the recipient to have a textual description of a body part, especially if the body part contains something other than text. The '`Content-Description:`' header contains just that; a free text description of the contents of a body part.

1.4 Signatures

It is common for people to want to include a small amount of text at the end of each message, to give information such as their telephone number, paper-mail ("snail-mail") address, other e-mail addresses. and so on.

To aid people in doing this, af supports a feature known as *signature files*. These files contain text which will automagically be included at the end of each mail message you send. By default, af's uses the file `.signature` in your home directory as your signature file.

Some people also like to include lengthy quotations, large ASCII graphics or other amusements in signature files. This annoys many people on the net, since it costs money to transfer large signatures, and conveys no useful information. Because of this, the version of af which you use may be configured to truncate signature files that are larger than a certain size; typically 4 lines of 79 columns each. You have been warned.

2 The Organization of the Screen

Normally, the af display occupies the whole screen. When you start af, the entire screen except for the last line is devoted to the folder you are reading. This area is called the *window*. The last line is a special *echo area* or *minibuffer window* where prompts appear and where you can enter responses. You can subdivide the window into different windows, each of which can be used to display a different folder (see Chapter 20 [Windows], page 80). In this manual, the word "window" always refers to the subdivisions of the screen within af.

The window that the arrow cursor is in is the *selected window*, in which most operations take place. Most af commands implicitly apply to the messages in the selected window. The other windows display messages for reference only, unless/until you select them.

Most windows display the details of folders, showing a *header line* for each message in the folder. Each header line shows a brief summary of the message's details, and is treated as a pointer to the message it describes.

Each window's last line is a *mode line* which describes what is going on in that window. It contains text that starts like '`==== Af:` *something*'. Its purpose is to indicate what buffer is being displayed above it in the window above it, how many messages are in the buffer, whether the buffer contains unsaved changes, and so on.

2.1 Point

Within a buffer displaying a folder, an arrow pointer shows the location at which commands will take effect. This location is called *point*. Many af commands move point through the buffer, so that you can execute commands at different places in it.

While the arrow appears to point *at* a message, you should think of point as *between* two messages; it lies *before* the message that it is pointing at. Sometimes people speak of "the cursor" when they mean "point", or speak of commands that move point as "cursor motion" commands.

If you are reading several folders in af, each in its own buffer, each buffer has its own point location. A buffer that is not currently displayed remembers where point is in case you display it again later.

When there are multiple windows on the screen, each window has its own point location. The arrow shows the location of point in the selected window. This also is how you can tell which window is selected. If the same buffer appears in more than one window, each window has its own position for point in that buffer.

It is possible to move point past the last message in a buffer. This is deliberate, to allow several other commands to work properly in an "Emacs-like" way. While it may seem strange at first if you are used to other mail readers, it soon becomes familiar; and it allows you to use many af features much more easily. Most commands that deal with messages will report an error if you have moved point past the last message in the buffer, just as they would if there were no messages at all.

2.2 The Echo Area

The line at the bottom of the screen (below the mode line) is the *echo area*. It is used to display small amounts of text for several purposes.

Echoing means displaying the characters that you type. Outside af, the operating system normally echoes all your input. Inside af things work a little differently.

Single-character commands do not echo in af, and multi-character commands echo only if you pause while typing them. As soon as you pause for more than a second in the middle of a command, af echoes all the characters of the command so far. This is to *prompt* you for the rest of the command. Once echoing has started, the rest of the command echoes immediately as you type it. This behaviour is designed to give confident users fast response, while giving hesitant users maximum feedback. You can change this behaviour by setting the variable `echo-keystrokes` (see Section 21.4 [Variables], page 85).

If a command cannot be executed, it may print an *error message* in the echo area. Error messages are accompanied by a beep.

Some commands print informative messages in the echo area. These messages look much like error messages, but they are not announced with a beep. Sometimes the message tells you what the command has done, when this is not obvious from looking at the screen. Sometimes the sole purpose of a command is to print a message giving you specific information—for example, *C-x =* prints a message describing the position of point in the buffer. Commands that take a long time often display messages ending in '`...`' while they are working, and add '`done`' at the end when they are finished.

The echo area is also used to display the *minibuffer*, a window that is used for reading arguments to commands, such as the name of a file to be read. When the minibuffer is in use, the echo area begins with a prompt string that usually ends with a colon; also, the cursor appears in that line because it is the selected window. You can always get out of the minibuffer by typing *C-g* (see Chapter 7 [Minibuffer], page 36).

2.3 The Header Lines

Most of the lines in a window which displays a folder will be *header lines*, which present a one-line summary of a mail message. By default a header line will look something like this:

ptr tags originator subject

ptr is the pointer which indicates the position of point in the window; the ptr will be represented as '`=>`' only on the line that point lies before, and as blank on all other lines. The line on which the pointer is present is sometimes referred to as the *current line*, and the message it represents as the *current message*, or the *message at point*.

tags are the *tags* of the message. Normally, this will only show *system tags* which af sets to show information about a message's status (see Chapter 13 [Tags], page 58).

originator is the sender of the message. If the sender's real name is available in the message headers then it will be shown here, otherwise their e-mail address will be displayed.

subject is the subject of the message.

It is possible to change the layout of the header lines by setting the variable `header-line-format` (see Section 21.4 [Variables], page 85). Also, the arrow pointer can be changed by setting the variable `header-line-arrow`.

2.4 The Mode Line

Each window's last line is a *mode line* which describes what is going on in that window. When there is only one window, the mode line appears right above the echo area. The mode line starts and ends with dashes, and it contains text like '`Af:` *something*'.

Normally, the mode line looks like this:

`=`*ch*`= Af:` *buf* `==` *count* `== (`*modes*`) ==` *pos* `=`

This gives information about the buffer being displayed in the window: the buffer's name, what modes are in use, whether the buffer has been changed, and how far down the buffer you are currently looking.

ch contains two stars '`**`' if the buffer has been changed (the buffer is "modified"), two plus signs '`++`' if the buffer contains messages whose status has changed, or '`==`' if the buffer has not been changed. For a read-only buffer, it is '`%*`' if the buffer is modified, and '`%%`' otherwise.

buf is the name of the window's *buffer*. In most cases this is the same as the name of a folder you are processing. (see Chapter 19 [Buffers], page 78)

count is the number of messages in the window's buffer.

The buffer displayed in the selected window (the window that the arrow cursor is in) is also af's selected buffer, the one that most commands operate on. When we speak of what some command does to "the buffer", we are talking about the currently selected buffer.

pos tells you whether there are more messages above the top of the window, or below the bottom. If your buffer is small and it is all visible in the window, *pos* is '`All`'. Otherwise, it is '`Top`' if you are looking at the beginning of the buffer, '`Bot`' if you are looking at the end of the buffer, or '*nn*`%`', where *nn* is the percentage of the buffer above the top of the window.

modes lists the *major mode* and any *minor modes* which are in effect in the buffer. At any time, each buffer is in one and only one of the possible major modes. The major modes available include Mail mode (for reading folders), Typeout mode (for displaying information) and Minibuffer mode (for asking the user for input). Each major mode may be supplemented by one or more minor modes, which change the mode's behaviour in some small way. (see Section 3.3 [Major Modes], page 25).

It is possible to change the layout of the mode line by setting the variable `mode-line-format` (see Section 21.4 [Variables], page 85).

3 Characters, Keys and Commands

ASCII consists of 128 character codes. Some of these codes are assigned graphic symbols such as 'a' and '='; the rest are control characters, such as *Control-a* (usually written *C-a* for short). *C-a* gets its name from the fact that you type it by holding down the ⟨CTRL⟩ key while pressing a.

Some control characters have special names, and special keys you can type them with: for example, ⟨RET⟩, ⟨TAB⟩, ⟨LFD⟩, ⟨DEL⟩ and ⟨ESC⟩. The space character is usually referred to below as ⟨SPC⟩, even though strictly speaking it is a graphic character whose graphic happens to be blank.

On ASCII terminals, there are only 32 possible control characters. These are the control variants of letters and '@[]\^_'. In addition, the shift key is meaningless with control characters: *C-a* and *C-A* are the same character, and af cannot distinguish them.

One af character set extension is that characters have an additional modifier, called Meta. Every character has a Meta variant; examples include *Meta-a* (normally written *M-a*, for short), *M-A* (not the same character as *M-a*, but those two characters normally have the same meaning in af), *M-*⟨RET⟩, and *M-C-a*. For reasons of tradition, we usually write *C-M-a* rather than *M-C-a*; logically speaking, the order in which the modifier keys ⟨CTRL⟩ and ⟨META⟩ are mentioned does not matter.

Some terminals have a ⟨META⟩ key, and allow you to type Meta characters by holding this key down. Thus, *Meta-a* is typed by holding down ⟨META⟩ and pressing a. The ⟨META⟩ key works much like the ⟨SHIFT⟩ key. Such a key is not always labelled ⟨META⟩, however, as this function is often a special option for a key with some other primary purpose.

If there is no ⟨META⟩ key, you can still type Meta characters using two-character sequences starting with ⟨ESC⟩. Thus, to enter *M-a*, you could type ⟨ESC⟩ a. To enter *C-M-a*, you would type ⟨ESC⟩ *C-a*. ⟨ESC⟩ is allowed on terminals with ⟨META⟩ keys, too, in case you have formed a habit of using it.

ASCII terminals represent function keys as a special sequence of ASCII characters. Because of this, it is possible, although not simple, to configure af to handle function and arrow keys. (see Chapter 21 [Customisation], page 82)

3.1 Keys

A *key sequence* (*key*, for short) is a sequence of characters that are meaningful as a unit—as "a single command." Some af command sequences are just one character; for example, just *C-n* is enough to move down one line. But af also has commands that take two or more characters to invoke.

If a sequence of events is enough to invoke a command, it is a *complete key*. Examples of complete keys include *C-n*, ⟨RET⟩, *C-x C-f* and *C-x 4 C-f*. If it isn't long enough to be complete, we call it a *prefix key*. The above examples show that *C-x* and *C-x 4* are prefix keys. Every key sequence is either a complete key or a prefix key.

Many single characters constitute complete keys in the standard af command bindings. A few of them are prefix keys. A prefix key combines with the following character to make a longer key sequence, which may itself be complete or a prefix. For example, *C-x* is a prefix key, so *C-x* and the next character combine to make a two-character key sequence. Most of these key sequences are complete keys, including *C-x C-f* and *C-x b*. A few, such as *C-x 4* and *C-x r*, are themselves prefix keys that lead to three-character key sequences. There's no limit to the length of a key sequence, but in practice people rarely use sequences longer than four characters.

By contrast, you can't add more characters onto a complete key. For example, the two-character sequence *C-n C-k* is not a key, because the *C-n* is a complete key in itself. It's impossible to give *C-n C-k* an independent meaning as a command. *C-n C-k* is two key sequences, not one.

All told, the prefix keys in af are *C-h*, *C-t*, *C-x C-x 4*, and ⟨ESC⟩. But this is not cast in concrete; it is just a matter of af standard key bindings. If you customise af, you can make new prefix keys, or eliminate these (see Section 21.6 [Key Bindings], page 87).

If you do make or eliminate prefix keys, that changes the set of possible key sequences. For example, if you redefine *C-n* as a prefix, *C-n C-k* automatically becomes a key (complete, unless you define it too as a prefix). Conversely, if you remove the prefix definition of *C-x 4*, then *C-x 4 f* (or *C-x 4 anything*) is no longer a key.

3.2 Keys and Commands

This manual is full of passages that tell you what particular keys do. But af does not assign meanings to keys directly. Instead, af assigns meanings to named *commands*, and then gives keys their meanings by *binding* them to commands.

Every command has a name chosen by a programmer. The name is usually made of a few English words separated by dashes; for example, `next-line` or `open-message`. The bindings between keys and commands are recorded in various tables called *keymaps* (see Section 21.6 [Key Bindings], page 87).

When we say that "*C-n* moves down vertically one line" we are glossing over a distinction that is irrelevant in ordinary use but is vital in understanding how to customise af. It is the command `next-line` that is programmed to move down vertically. *C-n* has this effect *because* it is bound to that command. If you rebind *C-n* to the command `end-of-buffer` then *C-n* will move to the end of the buffer instead. Rebinding keys is a common method of customisation.

In the rest of this manual, we usually ignore this subtlety to keep things simple. To give the information needed for customisation, we state the name of the command which really does the work in parentheses after mentioning the key that runs it. For example, we will say that "The command *C-n* (`next-line`) moves point vertically down," meaning that `next-line` is a command that moves vertically down and *C-n* is a key that is standardly bound to it.

While we are on the subject of information for customisation only, it's a good time to tell you about *variables*. Often the description of a command will say, "To change this, set the variable `foobar`." A variable is a name used to remember a value. Most of the variables documented in this manual exist just to facilitate customisation: some command or other part of af examines the variable and behaves differently according to the value that you set. Until you are interested in customising, you can ignore the information about variables. When you are ready to be interested, read the basic information on variables, and then the information on individual variables will make sense (see Section 21.4 [Variables], page 85).

3.3 Major Modes

Af has several modes of operation; known as *major modes*. A major mode is in effect a subset of af, where a command may have a different effect, and different commands may be available (This is probably the most pronounced difference between af and Emacs; Emacs major modes just alter its behaviour in minor ways, such as changing key bindings; while af major modes may have a completely different set of commands and behaviour).

There are three major modes currently available in af. *Mail mode* is the mode used for buffers displaying mail messages; most af buffers use mail mode. *Minibuffer mode* is used to support the minibuffer; it has no commands to handle mail messages, but many extra commands to allow editing of text. *Typeout mode* is the mode used for displaying text to the screen. Again, it has no commands to handle mail messages, it simply allows you to browse text displayed on the screen.

Each major mode allows you to bind keys independently of the other major modes (just like in Emacs), so keys may be bound in only one mode, or even be bound to different things in different modes (just as they can in Emacs). For example, in mail mode ⟨SPC⟩ just clears the echo area, in minibuffer mode it inserts a space into the text you are editing, and in typeout mode it scrolls the text up to display the next page.

Each af command may be available in all modes, or just in some of them. For example the command `open-message` is only available in mail mode, since in the other modes you are not dealing with mail messages and the command wouldn't make sense. So you can't run the command `open-message` in typeout mode even by typing *`M-x open-message`*.

A command may also have slightly different behaviour in different modes. For example the command search-forward in mail mode searches through the text of each mail message, looking for some text. In typeout mode it searches through the text you are viewing instead.

This distinction is only really important when customising af; when you are using af it is (hopefully) quite obvious what commands are likely to work. The important thing to remember is that keys may do very different things depending on whether you are browsing a list of messages, entering an argument in the minibuffer, or viewing text.

3.4 Minor Modes

A minor mode is a particular feature which can be turned on or off. For example, `Read Only` mode selects whether you are allowed to modify the contents of a buffer. Unlike the major modes, the minor modes are independent of each other, although they may only be valid in some major modes.

Minor modes are usually specific to a buffer, so they may be turned on in some buffers and off in others. Normally, minor modes are turned on or off automatically in response to some situation. There may also be a command to turn the minor mode off if it is on, or on if it is off; this will be formed from the prefix '`toggle-`' followed by the name of the mode. So the command to turn `Read Only` mode on or off is `toggle-read-only`.

4 Entering Af

The usual way to invoke af is with the shell command '`af`'. Af clears the screen and then displays an initial help message and copyright notice while it processes any startup files (see Section 21.2 [Startup Files], page 84), and then reads your incoming mailbox. (If your mailbox is small, you may not see the initial help message and copyright; it would be very inconvenient for experienced users if af were to pause long enough to let you read the message.) When af has finished reading the mailbox, it displays a final count of how many messages it read, and displays the buffer containing the messages.

Some operating systems discard all type-ahead when af starts up; they give af no way to prevent this. Therefore, it is advisable to wait until af clears the screen before typing your first command.

It is possible to specify folders to be visited, or details of a message you want to send, by giving af arguments in the shell command line (see Section 4.1 [Command Arguments], page 26). But you can do these things from within af too; whether you run af once and suspend it, or separately each time you want to read or send mail is simply a matter of taste.

Arguments starting with '`-`' are *options*. Other arguments specify addresses to send mail to. If you specify addresses (or use any of the '`-c`', '`-b`', or '`-E`' options), then af doesn't read any folders; it simply allows you to send a single mail message to the addresses you supplied, and then exits.

You can use options to specify various other things, such as which folders to read, the subject for mail you're sending, and so on. A few options support advanced usage, such as running `afl` functions in batch mode. The sections of this chapter describe the available options, arranged according to their purpose.

Most options specify how to initialise af, or set parameters for the af session. We call them *initial options*. A few options specify things to do: for example, read folders or load `afl` programs (see Chapter 22 [Afl], page 91). These are called *action options*. These and file names together are called *action arguments*. Af processes all the action arguments in the order they are written.

4.1 Command Line Arguments

Here is a table of the arguments and options that af accepts:

'*address*'
: Send mail to *address*. See Chapter 8 [Sending Mail], page 41. If addresses are supplied, then many of the other arguments are disabled, since af will not be visiting any folders. If the standard input is not a terminal, then af will silently send the message as requested, rather than expecting you to edit the text of the message.

'`-E`'
: This option has two effects. Firstly, it makes af start up to send a single mail message even if no addresses were given as arguments. Secondly, it forces af to allow you to edit the text of the message you send, even if the text is being obtained from the standard input. This doesn't really seem very useful, but occasionally it can be very handy in allowing you to write flexible scripts which send mail using af.

'-H' Force af to present the user with the headers of an outgoing message when editing it, or to force translation of headers in mail sent from the standard input. In effect, it causes af's `edit-initial-headers` variable to be set to `true` regardless of the contents of the user's '`.afrc`' file (see Chapter 8 [Sending Mail], page 41).

'-c *cc*' Specifies one or more addresses to send a carbon copy of a message to. If the addresses contain spaces then you should enclose them in double quotes.

'-b *bcc*' Specifies one or more addresses to send a blind carbon copy of a message to. If the addresses contain spaces then you should enclose them in double quotes.

'-s *subject*'
Specifies the Subject: for an outgoing message. Only one '`-s`' option may be given, and it only takes effect if addresses (or the '`-c`', '`-c`', or '`-E`' option) were given on the command line. If the subject contains spaces then you should enclose it in double quotes.

'-C *content-type*'
Specifies the MIME Content-Type: for an outgoing message. Only one '`-C`' option may be given, and it only takes effect if addresses (or the '`-E`' option) were given on the command line.

'-f *folder*' Read *folder* in place of your incoming mailbox. Multiple '`-f`' options may be given, in which case each folder is read into its own buffer. This option is ignored if addresses or the '`-E`' option were given.

'-u *user*' Read *user*'s incoming mailbox in place of your own. You must have permissions to read *user*'s mailbox. Multiple '`-u`' options may be given, and they may be mixed with '`-f`' options. This option is ignored if addresses or the '`-E`' option were given.

'-F' Interpret any arguments other than options as folders to be read in, rather than addresses to send mail to. This can be very useful if you want to read in a set of folders with '`af -F *`'.

'-w' Create enough windows to display all the folders given with the '`-f`', '`-u`' or '`-F`' options, subject to the maximum number of windows that will fit on the screen (usually about four on a 24-line display). This option is ignored if addresses or the '`-E`' option were given.

'-e' Check whether there is mail in your default mailbox (or, if '`-f`' or '`-u`' are given, any specified mailboxes) and then exit. The exit status is zero if there is mail in any of the mailboxes; one if not. This option is ignored if addresses or the '`-E`' option were given.

'-z' Makes af terminate immediately with an exit status of 1 if your incoming mailbox (or, if '`-f`' or '`-u`' are given, any specified folders) is empty. Otherwise, af will start up normally. This option is ignored if addresses or the '`-E`' option were given.

'-n' Ignore the file '`.afrc`' in the user's home directory. See Section 21.2 [Startup Files], page 84.

'-v' Prints the version of af and then exit. Ignores the effect of any other options.

'-l *loadfile*'
: Specifies a file which should be read and executed on startup before any buffers are created. Used with the '-n' option, this allows the user to specify an alternate startup file, rather than the default of '.afrc' in their home directory.

'-S *file*' Names a file of commands in af's internal language `afl`. The commands in the file are read and executed exactly as if af had been started up interactively and then the af command `load-file` had been executed. When all the commands in *file* have been executed, or if an error is encountered in the file, then af terminates. If the '-S' option is given, then af will not display the screen; only messages will be printed to the standard output. Only one '-S' option may be given, it is ignored if destinations or the '-E' option were specified.

4.2 Command Line Examples

Here are a few examples of af command line usage, with brief explanations.

Read two folders 'foo' and 'bar' into separate windows:

```
af -f foo -f bar -w
```

Check if there is any mail in folders foo and bar:

```
af -e -f foo -f bar
```

Send mail to 'fred' on the local system with subject 'Hi':

```
af -s Hi fred
```

Take a current directory listing, and use it as a base for some mail to user 'fred@foobar.co.uk':

```
ls -l | af -E fred@foobar.co.uk
```

Run the afl script in file 'script.afl' with folder 'foobar' as the default buffer:

```
af -S script.afl -f foobar
```

4.3 Environment Variables

Af uses several *environment variables*. An environment variable is a string passed from the operating system to af, and the collection of environment variables is known as the *environment*. Environment variable names are case sensitive and it is conventional to use upper case letters only.

Because environment variables come from the operating system there is no general way to set them; it depends on the operating system and especially the shell that you are using. For example, here's how to set the environment variable `ORGANIZATION` to 'Utter Chaos' using bash:

```
export ORGANIZATION="Utter Chaos"
```

and here's how to do it in csh or tcsh:

```
setenv ORGANIZATION "Utter Chaos"
```

It should be noted that the environment variables are used to *customise* af, not to configure it. None of these variables are required; and af should work correctly without them. They simply serve to tailor af to your taste.

Here is a list of the environment variables af uses, with a brief description of what they are used for.

`MAIL` The full path to your incoming mailbox.

`HOME` Your home directory.

`FOLDER` The directory where your mail folders are stored. Defaults to '`~/Mail`'.

`SAVEDIR` The directory where your news folders are stored. Defaults to '`~/News`'.

`TMPDIR` The directory where temporary files should be created.

`VISUAL, EDITOR`
The editor to use for editing outgoing mail (or messages if you choose to edit them). `VISUAL` overrides `EDITOR` if they are set to different values.

`PAGER` The program to use when you ask af to display a message via an external pager.

`NAME` Your real name. Used for generating the headers on the messages you send.

`ORGANIZATION`
your organisational affiliation. If set, af will generate an Organization: header on the messages you send.

4.4 Exiting af

There are two commands for exiting af because there are two kinds of exiting: *suspending* af and *killing* af.

Suspending means stopping af temporarily and returning control to its parent process (usually a shell), allowing you to resume reading mail later in the same af job, with the same buffers, same kill ring, and so on.

Killing af means destroying the af job. You can run af again later, but you will get a fresh af; there is no way to resume the same mail reading session after it has been killed.

C-z Suspend af (`suspend-af`)

C-x C-c Kill af (`save-buffers-kill-af`).

M-z Kill af, automatically saving any changed buffers (`save-all-kill-af`).

To suspend af, type *C-z* (`suspend-af`). This takes you back to the shell from which you invoked af. You can resume af with the shell command '`%af`' in most common shells.

On systems that do not support suspending programs, *C-z* starts an inferior shell that communicates directly with the terminal (`shell`). Af waits until you exit the inferior shell. (The way to do that is probably with *C-d* or '`exit`', but it depends on which shell you use). The only way on these systems to get back to the shell from which af was run (to log out, for example) is to kill af.

To kill af, type *C-x C-c* (`save-buffers-kill-af`). A two-character key is used for this to make it harder to type. This command first offers to save any modified file-visiting buffers. If you do not save them all, it asks for confirmation with *yes* before killing af, since any changes not saved will be lost forever.

To kill af, saving all modified file-visiting buffers, type *M-z* (`save-all-kill-af`). We don't recommend using this, since you might accidentally save changes you didn't mean to, but you can use *M-z* to exit if you prefer.

The operating system usually listens for certain special characters whose meaning is to kill or suspend the program you are running. **This operating system feature is turned off while you are in af.** The meanings of *C-z* and *C-x C-c* as keys in af were inspired by the use of *C-z* and *C-c* on several operating systems as the characters for stopping or killing a program, but that is their only relationship with the operating system. You can customise these keys to run any commands of your choice (see Section 21.6 [Key Bindings], page 87).

5 Basic af Commands

We now give the basics of how to read, save, and delete messages, send mail to people and save your changes back to your mailbox.

5.1 Reading Messages

The most basic operation in af is to *open* the current message; to display it's contents via typeout (see Chapter 6 [Typeout], page 35), or some other pager.

RET or *C-o*
: Open the current message, using the pager named in the `pager` variable, usually '`typeout`' (`open-message`).

M-⟨RET⟩ or *C-M-o*
: Open the current message, using the pager named in the `PAGER` environment variable (`page-message`).

C-x C-o
: Open the current message, displaying all the headers and the message body using the pager named in the `pager` variable. The body is not decoded, and multipart or partial messages are not processed in any way. (`open-raw-message`).

Normally, only mail headers not listed in the `headers-not-displayed` variable will be displayed when you open a message. A positive numeric argument makes af display all the message's headers. A negative argument makes af skip all headers, and only display the body of the message. See Section 5.8 [Arguments], page 33, for more information on numeric arguments.

When you open a multipart message, the headers and each textual body part will be displayed to a separate section of typeout. Non-textual body parts will be displayed using an external program if possible, otherwise you will be asked whether to view the text of the body part, save it to a file, pipe it into a command, or skip it. If any body part is an attachment, then af will offer you the choice of viewing, saving or skipping that body part.

Normally, af will ask for confirmation before it displays image, audio, video or application body parts, even if it knows how to. The `confirm-viewing` variable controls this, listing the content types which af will ask for confirmation before displaying.

When you open a partial message, af will try to rebuild and display the entire original message. If af can't rebuild the message then it will warn you about the problem, and offer you the choice of viewing or skipping that part of the message.

If your external pager doesn't need af to pause after displaying the message; then you can make reading messages more convenient by setting the variable `pause-after-paging-message` to `true`.

5.2 Changing the Location of Point

To do anything useful with af, you have to know how to move point (see Section 2.1 [Point], page 20). There are several keys which move point within a buffer.

C-n or *n*
: Move down one line, vertically (`next-line`).

`C-p` or `p` Move up one line, vertically (`previous-line`).

`M-.` Move point to the vertical centre of the window. (`move-to-window-line`). Text does not move on the screen.

A numeric argument says which screen line to place point on. It counts screen lines down from the top of the window (zero for the top line). A negative argument counts lines from the bottom (−1 for the bottom line).

`M-<` or `<` Move to the top of the buffer (`beginning-of-buffer`). With numeric argument *n*, move to *n*/10 of the way from the top. See Section 5.8 [Arguments], page 33, for more information on numeric arguments.

`M->` or `>` Move to the end of the buffer (`end-of-buffer`).

`M-g` Read a number *n* and move point to line number *n* (`goto-line`). Line 1 is the beginning of the buffer.

5.3 Scrolling

Since only part of a large buffer fits in the window, af tries to show the part that is likely to be interesting. The display control commands allow you to specify which part of the buffer you want to see.

`C-l` Clear screen and redisplay, scrolling the selected window to center point vertically within it (`recenter`). A numeric argument *n* says to move point to screen line *n*.

`C-v` Scroll forward (a windowful or a specified number of lines) (`scroll-up`).

`M-v` Scroll backward (`scroll-down`).

The names of all scroll commands are based on the direction that the messages move in the window. Thus, the command to scroll forward is called `scroll-up`, since the messages move up.

When scrolling a windowful at a time, af leaves two lines that were visible before you scrolled still visible afterwards, so that you can retain the context you were in before you scrolled. The number of lines of overlap across a `C-v` or `M-v` is controlled by the variable `next-screen-context-lines`; by default, it is two.

5.4 Deleting Messages

`C-k` or `k` Kill the current message, removing it from the folder (`kill-line`). See Chapter 14 [Killing and Yanking], page 62.

`M-x delete-message`

Mark the current message as deleted. This doesn't actually delete the message until you save the buffer. At that point any messages marked as deleted will be killed, and not written to the folder.

`M-x undelete-message`

Remove the deleted marker from the current message; so that it will not be killed when you save the buffer.

5.5 Message and Position information

Here are commands to get information about messages, and your position in the buffer.

M-= Print line number of point in the buffer (`what-cursor-position`).

C-x = Print a short summary of some of the message's details in the echo area (`message-info`).

C-t ? Print the message's tags in the echo area (`message-tags`). See Chapter 13 [Tags], page 58.

5.6 Quitting

At any time in af except when a command is running, you can type *C-g* (`keyboard-quit`) to quit from what you are doing. If you have typed part of a command, or a numeric argument, then *C-g* will get rid of it. If there isn't a partial command to get rid of, but you are in the minibuffer or typeout then typing *C-g* will exit back to the mail buffer, aborting the command you were running.

This means, that you can always get back to the top level of af by typing *C-g C-g*. This is useful for aborting commands, or if you are unsure where you are in af.

5.7 Help

If you forget what a key does, you can find out with the Help character, which is ⟨C-h⟩. Type *C-h k* followed by the key you want to know about; for example, *C-h k C-n* tells you all about what *C-n* does. ⟨C-h⟩ is a prefix key; *C-h k* is just one of its subcommands (the command `describe-key`). The other subcommands of ⟨C-h⟩ provide different kinds of help. Type ⟨C-h⟩ three times to get a description of all the help facilities. (see Chapter 10 [Help], page 51).

5.8 Numeric Arguments

Any af command can be given a *numeric argument* (also called a *prefix argument*), although it may be ignored. Some commands interpret the argument as a repetition count. For example, *C-n* with an argument of ten moves down ten lines instead of one. With these commands, no argument is equivalent to an argument of one. Negative arguments tell most such commands to move or act in the opposite direction.

If your terminal keyboard has a ⟨META⟩ key, the easiest way to specify a numeric argument is to type digits and/or a minus sign while holding down the ⟨META⟩ key. For example,

```
M-5 C-n
```

would move down five lines. The characters *Meta-1*, *Meta-2*, and so on, as well as *Meta--*, do this because they are keys bound to commands (`digit-argument` and `negative-argument`) that are defined to contribute to an argument for the next command.

Another way of specifying an argument is to use the *C-u* (`universal-argument`) command followed by the digits of the argument. With *C-u*, you can type the argument digits without holding down modifier keys; `C-u` works on all terminals. To type a negative argument, type a minus sign after `C-u`. Just a minus sign without digits normally means −1.

`C-u` followed by a character which is neither a digit nor a minus sign has the special meaning of "multiply by four". It multiplies the argument for the next command by four. `C-u` twice multiplies it by sixteen. Thus, `C-u C-u C-n` moves down sixteen lines. This is a good way to move down "fast", since it moves about 2/3 of a screen in the usual size screen. `C-u` is also a handy way of providing an argument when you don't care about the value.

Many commands care only about whether there is an argument, and not about its value. Other commands care only about the sign of the argument. For example, the command ⟨RET⟩ (`open-message`) with no argument shows only some of the message's headers; with a positive argument, it shows all the headers, and with a negative argument it shows no headers at all. This may seem strange, but it is a convenient way of modifying the behaviour of a command.

We use the term "prefix argument" as well as "numeric argument" to emphasise that you type the argument before the command, and to distinguish these arguments from minibuffer arguments that come after the command.

6 Typeout

Typeout is the mode af uses to display large amounts of text such as mail messages or help information. When af uses typeout, the typeout window appears, covering the entire screen apart from the echo area. Once you exit typeout, the window disappears, "uncovering" the original windows.

If possible, each line of the text is displayed as a single line on the screen. If a line is too long to fit on a single screen line, then it will be displayed on several screen lines, with a '\' at the extreme right margin of all but the last of them. The '\' says that the following screen line is not really a distinct line in the text, but just the *continuation* of a line too long to fit the screen. This is called *line wrapping*, and the lines after the first are often referred to as *continuation lines*.

When you are viewing typeout, the position indicator in the mode line will often say '`MOR`'. This is because typeout does not wait to read the entire input before displaying the first page, and since af doesn't know how many lines there are in the text yet, it will give the position as '`MOR`' to indicate this.

Most of the time, typeout consists of a single *section*. But when typeout is displaying related outputs, such as the body parts of a multipart message, typeout will consist of several sections. Each section of typeout will be displayed separately.

Sometimes, the typeout window will appear, but you will still be able to carry on with some other operation (such as displaying the help options via *C-h C-h C-h*). In these cases the typeout window will disappear when the operation is completed.

Within typeout, only the cursor motion commands, the scrolling commands and the text searching commands can be used. To exit typeout, simply type *C-g*. The other af commands are irrelevant when you are just viewing text. Here is a brief summary of the commands available in typeout:

⟨SPC⟩
: Scroll the text up to show the next page of text. If you are already at the end of the text, then exit typeout (`typeout-scroll`). This is a convenient way of paging through short amounts of text.

C-v
: Scroll forward (a windowful, or a specified number of lines) (`scroll-up`).

M-v
: Scroll backward (a windowful, or a specified number of lines) (`scroll-down`).

C-n or *n*
: Scroll forward one line (`next-line`).

C-p or *p*
: Scroll backward one line (`previous-line`).

C-] or *]*
: Move to the next typeout section (`next-section`).

C-[or *[*
: Move to the previous typeout section (`previous-section`).

C-l
: Redraw the display (`recenter`).

C-s regex ⟨RET⟩
: Search forward through the text for a line matching *regex* (`search-forward`). See Chapter 15 [Searching], page 65.

C-r regex ⟨RET⟩
: Search backward through the text for a line matching *regex* (`search-forward`). See Chapter 15 [Searching], page 65.

7 The Minibuffer

The *minibuffer* is the facility used by af commands to read arguments more complicated than a single number. Minibuffer arguments can be file names, buffer names, af command names, address lists, and many other things, depending on the command reading the argument. You can use many editing commands in the minibuffer to edit the argument.

When the minibuffer is in use, it appears in the echo area, and the terminal's cursor moves there. The beginning of the minibuffer line displays a *prompt* which says what kind of input you should supply and how it will be used. Often this prompt is derived from the name of the command that the argument is for. The prompt normally ends with a colon.

Sometimes a *default argument* appears in parentheses after the colon; it too is part of the prompt. The default will be used as the argument value if you enter an empty argument (e.g., just type ⟨RET⟩). For example, commands that read buffer names always show a default, which is the name of the buffer that will be used if you type just ⟨RET⟩.

The simplest way to enter a minibuffer argument is to type the text you want, terminated by ⟨RET⟩ which exits the minibuffer. You can cancel the command that wants the argument, and get out of the minibuffer, by typing *C-g*.

Sometimes, a *default response* will already be in the minibuffer when you enter it. If it is completely wrong, then you can type *C-a C-k* to remove it. These default responses appear for convenience when you are prompted for a long value that you are likely to want to edit, rather then retype from scratch.

Most commands which use typeout allow you to redirect the output into a file, instead of viewing it. The major exception to this is ⟨RET⟩ (`open-message`), since there are several ways to save a message to a file. To redirect typeout to a file simply give the command a prefix argument of any value; you will be prompted for the file name using the minibuffer.

7.1 Echo Area Conflicts

Since the minibuffer uses the screen space of the echo area, it can conflict with other ways af customarily uses the echo area. Here is how af handles such conflicts:

- If a command gets an error while you are in the minibuffer, this does not cancel the minibuffer. However, the echo area is needed for the error message and therefore the minibuffer itself is hidden for a while. It comes back after a few seconds, or as soon as you type anything.
- If in the minibuffer you use a command whose purpose is to print a message in the echo area, such as *C-x =*, the message is printed normally, and the minibuffer is hidden for a while. It comes back after a few seconds, or as soon as you type anything.
- Echoing of keystrokes does not take place while the minibuffer is in use.

7.2 Inserting Text

To insert printing characters into the minibuffer, just type them. This inserts the characters you type into the buffer at the cursor (that is, at *point*; which has a different appearance in the minibuffer, but the same meaning). The cursor moves forward, and any text after the cursor moves forward too. If the text in the buffer is '`foobar`', with the cursor before the 'b', then if you type `xx`, you get '`fooxxbar`', with the cursor still before the 'b'. This all works because in the minibuffer all the printing characters run the command `self-insert-command`.

Direct insertion works for printing characters and ⟨SPC⟩, but other characters act as editing commands and do not insert themselves. If you need to insert a control character or a character whose code is above 200 octal, you must *quote* it by typing the character *C-q* (`quoted-insert`) first. There are two ways to use *C-q*:

- *C-q* followed by any non-graphic character (even *C-g*) inserts that character.
- *C-q* followed by three octal digits inserts the character with the specified character code.

A numeric argument to *C-q* specifies how many copies of the quoted character should be inserted (see Section 5.8 [Arguments], page 33).

When you have finished entering the text, simply type ⟨RET⟩ (`newline`) to accept the argument and exit the minibuffer.

7.3 Deleting Text

To *delete* a character you have inserted, use ⟨DEL⟩ (`delete-backward-char`). ⟨DEL⟩ deletes the character *before* the cursor (not the one that the cursor is on top of or under; that is the character *after* the cursor). The cursor and all characters after it move backwards. Therefore, if you type a printing character and then type ⟨DEL⟩, they cancel out.

To delete the character that the cursor is under, use *C-d* (`delete-char`). This deletes the character, moving all the characters after the cursor backwards. The cursor is left in place. Therefore *C-d* is equivalent to *C-f DEL*.

To delete the text from the cursor to the end of the line, use *C-k* (`kill-line`). This kills the characters from the one the cursor is under to the end of the line, leaving the cursor at the end of the line (see Section 14.5 [Minibuffer Kills], page 63).

7.4 Editing the Text

To edit the text in the minibuffer, you need to be able to move around in the text. There are several commands available to do this:

C-a Move to the beginning of the line (`beginning-of-line`).

C-e Move to the end of the line (`end-of-line`).

C-f Move forward one character (`forward-char`).

C-b Move backward one character (`backward-char`).

C-t Transpose the two characters on either side of point, moving point forward one character (`transpose-chars`).

7.5 Dealing with Words

The minibuffer includes several commands for dealing with words rather than characters. Often these take the usual command for moving point, but use a Meta key rather then a control key:

M-f Move forward one word (`forward-word`).

M-b Move backward one word (`backward-word`).

M-⟨DEL⟩ Delete the next word (`backward-kill-word`).

M-d Delete the previous word (`kill-word`).

M-l Convert the next word to lower case (`downcase-word`).

M-u Convert the next word to upper case (`upcase-word`).

M-c Capitalise the next word (`capitalize-word`).

7.6 Completion

For certain kinds of arguments, you can use *completion* to enter the argument value. Completion means that you type part of the argument, then af visibly fills in the rest, or as much as can be determined from the part you have typed.

When completion is available, certain keys—⟨TAB⟩, ⟨RET⟩, and ⟨SPC⟩—are rebound to complete the text present in the minibuffer into a longer string that it stands for, by matching it against a set of *completion alternatives* provided by the command reading the argument. *?* is defined to display a list of possible completions of what you have inserted.

For example, when *M-x* uses the minibuffer to read the name of a command, it provides a list of all available af command names to complete against. The completion keys match the text in the minibuffer against all the command names, find any additional name characters implied by the ones already present in the minibuffer, and add those characters to the ones you have given. This is what makes it possible to type *M-x del* ⟨SPC⟩ *m* ⟨RET⟩ instead of *M-x delete-message* ⟨RET⟩ (for example).

Case is often significant in completion, because it is significant in many of the names that you can complete (buffer names and file names). Thus, '`fo`' often does not complete to '`Foo`'. Completion does ignore case distinctions for certain arguments in which case does not matter (such as af command or configuration variable names).

7.6.1 Completion Example

A concrete example may help here. If you type *M-x co* ⟨TAB⟩, the ⟨TAB⟩ looks for alternatives (in this case, command names) that start with '`co`'. There are only two: `copy-region-as-kill` and `copy-tagset-as-kill`. These are the same as far as `copy-`, so the '`co`' in the minibuffer changes to '`copy-`'.

If you type ⟨TAB⟩ again immediately, there are multiple possibilities for the very next character—it could be '`r`' or '`t`'—so no more characters are added; instead, ⟨TAB⟩ just produces a beep. If you now type *?*, af will produce a list of all possible completions to typeout; when you exit typeout, the minibuffer is still waiting for the next character.

If you go on to type *r* ⟨TAB⟩, this ⟨TAB⟩ sees '`copy-r`'. The only command name starting this way is `copy-region-as-kill`, so completion fills in the rest of that. You now have '`copy-region-as-kill`' in the minibuffer after typing just *co* ⟨TAB⟩ *r* ⟨TAB⟩. Note that ⟨TAB⟩ has this effect because in the minibuffer it is bound to the command `minibuffer-complete` when completion is available.

7.6.2 Completion Commands

Here is a list of the completion commands defined in the minibuffer when completion is available.

⟨TAB⟩
: Complete the text in the minibuffer as much as possible (`minibuffer-complete`).

⟨SPC⟩
: Complete the minibuffer text, but don't go beyond one word (`minibuffer-complete-word`).

⟨RET⟩
: Submit the text in the minibuffer as the argument, possibly completing first as described below (`minibuffer-complete-and-exit`).

?
: Print a list of all possible completions of the text in the minibuffer (`minibuffer-list-completions`).

⟨SPC⟩ completes much like ⟨TAB⟩, but never goes beyond the next hyphen or space. If you have '`copy-r`' in the minibuffer and type ⟨SPC⟩, it finds that the completion is '`copy-region-as-kill`', but it stops completing after '`region-`'. This gives '`copy-region-`'. Another ⟨SPC⟩ at this point completes to '`copy-region-as-`'.

7.6.3 Strict Completion

There are three different ways that ⟨RET⟩ can work in completing the minibuffer, depending on how the argument will be used.

- *Strict* completion is used when it is meaningless to give any argument except one of the known alternatives. For example, when *M-x* reads the name of a command to run, it is meaningless to give anything but the name of an existing command or macro. In strict completion, ⟨RET⟩ refuses to exit if the text in the minibuffer does not complete to an exact match.
- *Cautious* completion is similar to strict completion, except that ⟨RET⟩ exits only if the text was an exact match already, not needing completion. If the text is not an exact match, ⟨RET⟩ does not exit, but it does complete the text. If it completes to an exact match, a second ⟨RET⟩ will exit.

  Cautious completion is used for reading file names for files that must already exist.
- *Permissive* completion is used when any string whatever is meaningful, and the list of completion alternatives is just a guide. For example, when *C-x C-f* reads the name of a folder to visit, any folder name is allowed, in case you want to create a folder. In permissive completion, ⟨RET⟩ takes the text in the minibuffer exactly as given, without completing it.

7.7 Minibuffer History

Every argument that you enter with the minibuffer is saved on a *minibuffer history list* so that you can use it again later in another argument. You can think of the minibuffer history as a buffer which you can move through displaying a line at a time.

C-p Move to the next earlier argument string saved in the minibuffer history (`previous-line`).

C-n Move to the next later argument string saved in the minibuffer history (`next-line`).

M-< Move to the first argument string saved in the minibuffer history (`beginning-of-buffer`).

M-> Move to the last argument string saved in the minibuffer history (`end-of-buffer`). This will always move you to the text you were editing before using the minibuffer history.

C-r Move to an earlier saved argument in the minibuffer history that matches the current text (`search-backward`).

C-s Move to a later saved argument in the minibuffer history that matches the current text (`search-forward`)..

The simplest way to reuse the saved arguments in the history list is to move through the history list one element at a time. While in the minibuffer, type *C-p* to "move to" the previous minibuffer input, and use *C-n* to "move to" the next input. Similarly, you can use *C-<* and *C->* to "move to" the first and last elements in the history list.

When you move to a new history line, any text you had typed is stored for you as the last history element, so you can return to it by typing *M->*. To use a history element as the argument, exit the minibuffer as usual with ⟨RET⟩. You can also edit the text before you reuse it; this does not change the history element that you "moved" to, but your new argument does go at the end of the history list in its own right.

There are also commands to search forward or backward through the history. At the moment they search for a history element that starts with the text you have typed in the minibuffer, up to the position of the cursor; so typing '`list- C-r`' will search for a history element that begins with '`list-`'. If a matching history element is found then the cursor will stay in the same column, so that typing *C-r* again will search for another history element beginning with `list-`.

8 Composing and Sending Mail

Af allows you to compose and send mail as well as read it. There are several different commands to send mail, each of which sends mail in a slightly different way.

Af composes messages in three stages. First you are prompted for any details needed for the message, such as the addresses to send it to. Then af starts an editor, so that you can edit the message. Finally, you are prompted for what to do with the message (its *disposition*).

8.1 Commands for Sending Mail

There are seven commands for sending mail in af. These commands are all similar in use, but conceptually different.

`M-m` or *`M-s`*
: Send a message to one or more people. This allows you to compose and send a mail message (`send-mail`). A numeric argument makes af compose a MIME message. Alternatively, you can attach files or compositions to the message later.

`M-a`
: Prompts for a file name and then uses the contents of the file to compose a message to one or more people (`send-file`). You will not normally edit the text of the message. A numeric argument makes af prompt for the MIME Content-Type of the message to compose. If af detects that the file name indicates some content type other than plain text then it will default the Content-Type and prompt for the information anyway.

`M-r`
: Reply to the current message. (`reply-to-message`). This lets you compose and send mail to whoever the current message is from.

`C-M-r`
: Group reply to the current message. (`group-reply-to-message`). This lets you compose and send mail to whoever the message is from, and all the addresses listed in the '`To:`' header. Copies are sent to the addresses listed in the '`Cc:`' header.

`M-f`
: *Forwarding* allows you to compose an annotated copy of the current message and send it to one or more people. (`forward-message`).

`C-M-f`
: Allows you to compose a new message, and the current message will then be attached to the message you are sending (`forward-message-as-attachment`). A numeric argument makes af include the full headers of the message to attach. This is often useful when sending reports of unsolicited commercial email to the sender's postmaster.

`C-x C-d`
: Allows you to compose a new message, and the messages in the region (see Chapter 12 [Mark and Region], page 56) are attached to the outgoing message as a MIME digest (`forward-region-as-digest`). This is useful for forwarding several messages to someone in a convenient format.

`C-t C-d`
: Allows you to compose a new message, and the messages in the tagset (see Chapter 13 [Tags], page 58) are attached to the outgoing message as a MIME digest

(`forward-tagset-as-digest`). This is useful for forwarding several messages to someone in a convenient format.

M-b *Bouncing* means sending an exact copy of the current message to one or more addresses, making it appear that the copy you send them is from the person who sent the message to you (`bounce-message`). You will not normally edit the text of the message.

8.2 Initial Details for Composing Mail

Depending on how you are sending mail, and how you have configured af, you will be prompted for several things when you begin composing a mail message. Normally af will only prompt for '`To:`' and '`Subject:`', and possibly '`Content-Type:`', '`Content-Disposition:`', and '`Content-Description:`' if you give `send-file` an argument, but here are all the possible questions:

'`To:`' The addresses you want send the mail to.

'`Subject:`'
A brief description of what the message is about.

'`Cc:`' Any mail addresses to send additional copies of the message to.

'`Bcc:`' Any mail addresses to send *blind copies* of the mail to. These addresses will not be listed in the headers of the message when it is sent.

'`Content-Type:`'
The MIME content type which describes the contents of the message, for example '`text/plain`', or '`image/jpeg`'. See Section 1.3 [Message Body], page 16.

'`Content-Disposition:`'
Can be set to '`inline`' or '`attachment`', indicating that the message contains an attachment which should not be viewed by default. The optional '`filename`' parameter (which is automatically set when you edit a file) indicates a suggested file name for the attachment. See Section 1.3 [Message Body], page 16.

'`Content-Description:`'
A brief description of the contents of the message body. See Section 1.3 [Message Body], page 16.

'`Signature file`'
Which signature file (see Section 1.4 [Signatures], page 19) do you want to include at the end of the message.

'`Copy message (y/n)`'
If you are replying to a message, you can copy the text of of the original message into the initial text of the reply. This is called *quoting*.

8.3 Editing the Mail Message

When you first edit a message you will normally see either a blank file, or the textual body parts of the original message with each line prefixed with '`> `'. You can then use your editor as usual to compose the text of the message. When you save the message, your '`.signature`' file will be appended to the message, separated by a line containing '`-- `'.

When you edit a message for the second time (or if you have configured af to do so), the headers of the message are inserted into the file you edit, separated by a blank line, and your signature will be visible at the end of the file. You can edit the headers with your editor, and af will process the changes when you exit the editor; reporting any problems to typeout. You can create headers by adding them into the header section of the file, or delete them by deleting them from the file.

You should be very careful when editing a message's headers. If you accidentally insert a blank line before the headers, then af will not find them when you exit the editor, and will therefore think that you have deleted them all. This can normally be fixed by editing the file again, and removing the offending text from the start of the file.

If a message is a multipart MIME message, then instead of using an editor to edit the message, the various body parts of the message will be displayed in compose mode. There you can add or delete body parts, or edit existing ones. See Section 8.5.4 [Non-Text Messages], page 45, for more details.

8.4 Options After Editing the Message

When you finish editing a message and exit the editor, af will display a summary of the message's headers and prompt you with

```
Send, Edit, Attachments, Check spelling, List or Forget?
```

It is a **very** good idea to check the headers that af displays, to make sure that you are sending the message to the people you think you are, that the subject is appropriate, and so on. Your options at this point are:

'`Send`'
: Confirm the message, and give it to the system for delivery.

'`Edit`'
: Edit the message again, this time inserting the headers and any signature into the text for you to edit.

'`Attachments`'
: Enter compose mode on the message you're composing, allowing you to attach files or create new body parts. See Section 8.5.4 [Non-Text Messages], page 45, for more details on compose mode.

'`Check spelling`'
: Run an interactive spelling checker on the body of the message (but not the headers). This option may not be available on systems where there isn't an interactive spelling checker installed.

'`List`'
: List the message to typeout, exactly as it will be given to the system for delivery. This can be useful for checking a message, as it is much quicker than starting up an editor.

'`Forget`'
: Cancel the message and exit mail composition. There is no way to recover a message that you have "forgotten".

8.5 Composing MIME Mail.

Normally, electronic mail is composed in the `us-ascii` character set, which doesn't contain any accented or international characters. Af allows you to compose MIME messages in character sets other than `us-ascii`, and to encode these messages so that they are not damaged in transit.

8.5.1 Internationalised Mail

The default `us-ascii` character set contains only the characters used in American English. If you want to include international characters such as accents, Hebrew, Cyrillic or so on, you will need to make af compose messages in another character set, for example the `iso-8859-1` character set that covers Western European languages, or the `iso-8859-8` character set for Hebrew text.

The `Content-Type:` header defines how the message is to be treated. For text messages it is usually set to `text/plain; charset=us-ascii`. If you change the value to `text/plain; charset=iso-8859-1`, then the message should be interpreted as containing characters in the `iso-8859-1` character set.

The `default-charset` variable controls which character set you compose mail messages in by default. For example, to compose messages in the `iso-8859-1` character set, you would set the `default-charset` variable to `iso-8859-1`. Once you have done this, any messages you compose will be marked as containing characters from that character set. You can override this default by editing the message headers.

The person who installed af at your site may have already configured it to compose messages in the correct character set. In this case you don't need to do anything to make af handle internationalised messages correctly.

Even if you compose messages in an international character set by default, af will still mark them as `us-ascii` if they don't contain any international characters. So you should rarely, if ever, need to change the character set of a message yourself.

8.5.2 Encoding Mail

Unfortunately, the international mail network is often unable to handle messages including international characters properly. To avoid this, af allows you to *encode* the message, converting any international characters into special sequences of ascii characters. Unfortunately, this means that the message will not be properly readable if the recipient doesn't have a MIME-capable mail reader. Alternatively, you can choose not to encode your message, and risk it being damaged in transit. Which choice is better is entirely dependent on your local situation.

The `default-text-encoding` variable controls how international messages should be encoded. The possible values are:

`7bit`
: The message consists only of us-ascii characters.

`8bit`
: The message contains international characters, but will not be encoded.

`quoted-printable`
: The message will be encoded with the MIME `quoted-printable` encoding. This is usually the best encoding for internationalised text messages.

'base64' The message will be encoded with the MIME 'base64' encoding. This encoding is not normally a good choice for text, unless the text consists mostly of international characters.

The person who installed af at your site may have already configured it to encode internationalised messages in the best way. In this case you don't need to do anything to make af encode internationalised messages correctly.

8.5.3 International Headers

Af handles international characters in headers by encoding them in a special form, to avoid damage in transit. This is done automatically whenever af encounters an international character in a header.

Sometimes you will encounter comments like '(**iso-8859-8**)' in a header containing international characters. If you are viewing the header, this means that the header contains international characters in a character set that isn't listed in the `viewable-charsets` variable, and is a warning that af can't correctly display the characters, what you see is a "best effort".

If you are editing the header, then such a comment means that the character set is not the one you are composing the message in. In this case the af will preserve the character set by assuming that any international characters after the comment contain characters in the specified character set. You can change this by adding another comment, such as '(**iso-8859-1**)' to force the character set back to your preference.

If you often need to type international characters into mail, the command `minibuffer-set-iso-keys` may be useful. This command sets the `allow-meta-bindings` variable (see Section 21.6.1 [Keymaps], page 88), and then binds all the meta characters found in the 'iso-8859-*' character sets to insert themselves when typed in the minibuffer. This is often useful when composing headers in languages other than English.

8.5.4 Non-Text Messages

There are two ways to compose non-text messages with af. You can use the `send-file` command, and af will ask you for the content type, disposition, and description if it recognises that the name of the file indicates that it isn't textual (see Section 21.1 [Mime Configuration], page 82). Or you can use an argument to send-file to force af to ask you for the content type, disposition, and description, but that requires you to know the correct content-type. Af will encode the message with the 'base64' encoding if required.

The second way is to supply an argument to the `send-mail` command, or select the `Attachments` option from the menu presented after editing the text of a message. In either case you will enter *compose mode*, where you can attach files or other newly-created data to the message.

Compose mode looks very much like Mail mode, except that each line displays one body part of the message. Each body part can be of any type. The commands for moving around the screen, and viewing and deleting messages that you use in mail mode all work the same way in compose mode.

You can use two commands to add a body part in compose mode. One is `a` or `M-a` (`attach-file`). This prompts for a file name, a Content-Type (which may be defaulted from the file name), and a Content-Description, a textual description of the new body part. The contents of the file are encoded if required, and then used to create a new body part, which is inserted into the compose buffer before point.

The second command is `c` or *M-c* (`compose-body-part`). This prompts for a Content-Type, Disposition, and Description, and then uses an external program to create the body-part. The system needs to be properly configured to create MIME mail, or af won't be able to find out which program to use, and will give an error message, unless the Content-Type is textual, in which case it will use an editor to create the content.

In a similar way you can use `e` or *M-e* (`edit-body-part`) to edit the contents of a body part. Again, if the system is not configured to do this then af will give an error message for body part's that aren't textual.

You can edit the description of a body part using *d* or *M-d* (`edit-description`). You will be presented with the description of the body part in the minibuffer for editing.

You can also edit the top-level headers of the composition using *h* or *M-h* (`edit-headers`). This presents the headers of the message in an editor, just as when creating a non-MIME message. Any text other than headers in the file will be discarded, and a warning given.

When you've finished attaching files or composing body parts, then use *C-M-c* or *q* (`save-composition-and-exit`). This will exit compose mode, creating a new message from the body parts. You will be prompted for the disposition of the message, just as for any other message you've edited.

If you want to abandon your changes in compose mode, then use *C-M-x* (`abandon-composition-and-exit`). You will be asked if you really want to lose all your changes, and if you confirm, then compose mode will exit, ignoring all your changes.

If you entered compose mode by supplying an argument to the `send-mail` command, then the entire command will be aborted. Otherwise you will be prompted for the disposition of the message normally.

8.6 Variables Related to Composing Mail

There are a number of configuration variables related to composing and sending mail. Here is a summary of each variable and its effects:

`addresses-to-ignore`
: Contains mail addresses which are to be ignored in group replies, typically your mail addresses on this and other machines. Default is unset.

`ask-bcc`
: If set to `true`, then af will prompt for addresses to send blind copies to whenever you send mail. Default is `false`.

`ask-cc`
: If set to `true`, then af will prompt for addresses to send copies to whenever you send mail. Default is `false`.

`auto-fold-headers`
: If set to `false`, then af will not automagically fold long header lines onto several lines. Default is `true`.

`compose-line-format`
: Defines the format of each of the body part lines in compose mode, the the same way as `header-line-format` defines the format of header lines in mail mode. Defaults to `%a %t %k %S`.

`copy-on-reply`
: If set to `false`, af will never quote the text of the original message in the initial text of a reply. If set to `true` af will always quote the original message. If set to `ask` then you will be asked each time you reply to a message. Defaults to `ask`.

`copy-preface`
: If set, the value is inserted before any text copied into the body of a replying or forwarding message. Any conversion characters in the value are expanded to their full value. A good value for this variable might be '`On %D, %o wrote:`'. Default is unset.

`copy-prefix`
: The prefix for lines of text copied into an outgoing message when replying or forwarding a message. Defaults to '`> `'.

`edit-initial-headers`
: It set to `true`, headers will be placed into the file to edit the first time you edit a message. When set to `accept`, headers will not be displayed for editing, but any valid headers typed at the start of the message will be accepted. Defaults to `false`.

`edit-initial-signature`
: When set to `true`, any signature will be placed into the editor the first time you edit a message. Defaults to `false`.

`edit-reply-address`
: When set to `true`, allows editing of the destination address when replying to mail. It is intended for use by people at sites with inadequate mail configurations. Defaults to `false`.

`editor`
: The program to use to edit messages. Defaults to the value of the `VISUAL` or `EDITOR` environment variable.

`forward-subject`
: If set, the value is used as a default for the subject of a forwarded or resent message. Any conversion characters in the value are expanded to their full value. A good value for this variable might be '`%s (Forwarded message from %o)`'. Default is unset.

`headers-to-copy`
: A colon separated list of header names which should be included in text copied into an outgoing message when replying to or forwarding a message. Default is unset (no headers are copied).

`multiple-reply-warning`
: If set to `true`, then af will prompt for confirmation if you try to reply to a message that you have already replied to. Defaults to `false`.

`organization`
: If set, the the value is inserted into an '`Organization:`' header in the outgoing message. Defaults to the value of the `ORGANIZATION` environment variable.

`outbound-folder`
: Gives the name of a folder in which to store a copy of all messages sent with af. The messages are stored with headers (except for a '`Message-ID:`') in the named folder, immediately after being sent. Default is unset (don't save messages).

`outbound-threshold`
: Gives the maximum number of lines in a message body which should be silently saved to the `outbound-folder`. Af will prompt for confirmation that a message longer than this limit is to be saved. If set to 0 then all outbound messages will be saved regardless of length. Default is 100 lines.

`preserve-cc-in-group-reply`
: If set to `true`, any `Cc:` header on a message is duplicated in an outgoing group-reply to that message, so that recipients of a carbon-copy will also receive the reply. Default is `true`.

`real-name`
: Your real name, for inclusion in '`From:`' headers. Defaults to the value of the `NAME` environment variable, or your details stored in the file '`/etc/passwd`'.

`reply-address`
: Contains a mail address which forms a '`Reply-To:`' header in all messages you send, so that replies to those messages will be sent to the address you specify. Default is unset.

`signature-file`
: The name of your signature file. If the value begins with '`ask:`', then you will be prompted for the signature file to use, with any value after the '`ask:`' being used as a default. Defaults to '`.signature`' in your home directory.

`signature-separator`
: A string which will be printed on a line between the body of an outgoing message and any automatically-included signature. If not set, then no separator line is printed. Defaults to '`--`'.

`spell-checker`
: The program to use to interactively spell-check the body of a message before you send it. Defaults to '`ispell -x`' if `ispell` is installed on your system, or unset otherwise.

8.7 Defining and Using Mail Aliases

Af will let you set *mail aliases*. These are short mnemonic names which stand for mail addresses or groups of mail addresses. Like many other mail programs, af expands aliases when they occur in the '`To:`', '`From:`', '`Cc:`', '`Bcc:`', and '`Reply-to:`' headers.

The easiest way to define a mail alias within af is with *C-x C-a* (`set-alias`). Af will prompt for the alias and then the real name and the full address of the alias, defaulted from the From: header of the current message. If the alias already exists, af will prompt for confirmation that you want to change the alias.[1]

C-x C-a sets the alias internally; but it also writes the alias to the file '`.afalias`' in your home directory. This file is read whenever af starts up, so your aliases will be available next time you run af. You can also edit your '`.afalias`' file directly; each line should either be blank, be a comment beginning with '`;`', or look like:

alias:*Real Name*:*addresses*

alias is the name of the alias; *Real Name* is the real name of the person or group the alias represents, and *addresses* stands for one or more mail addresses for *alias* to expand into. Separate multiple addresses with spaces or (preferably) commas. You can make an address list span more than one line by starting the second and subsequent lines with a space or tab, in the same way as mail headers are split over several lines of text.

For instance, to make `afbug` stand for `af-bug@csv.warwick.ac.uk` put in these lines:

```
; Set up the af-bug mailing list as an alias
afbug:Af Bug Mailing List:af-bug@csv.warwick.ac.uk
```

Af expands aliases as soon as you finish editing the mail file, so you can check that the alias expanded properly by looking at the list of headers that af shows you when it prompts for the disposition of the mail.

[1] If you edit your '`.afalias`' file, you will find that the old version of the alias will be commented out rather than deleted when you change it. This can be useful if you accidentally overwrite an important alias.

9 Running Commands by Name

The af commands that are used often or that must be quick to type are bound to keys—short sequences of characters—for convenient use. Other af commands that do not need to be brief are not bound to keys; to run them, you must refer to them by name.

A command name is, by convention, made up of one or more words, separated by hyphens; for example, `af-version` or `list-commands`. The use of English words makes the command name easier to remember than a key made up of obscure characters, even though it is more characters to type.

The way to run a command by name is to start with *M-x*, type the command name, and finish it with ⟨RET⟩. *M-x* uses the minibuffer to read the command name. ⟨RET⟩ exits the minibuffer and runs the command. The string '`M-x`' appears at the beginning of the minibuffer as a *prompt* to remind you to enter the name of a command to be run. See Chapter 7 [Minibuffer], page 36, for full information on the features of the minibuffer.

Note that `next-line` is the same command that you invoke with the key *C-n*. You can run any af command by name using *M-x*, whether or not any keys are bound to it.

If you type *C-g* while the command name is being read, you cancel the *M-x* command and get out of the minibuffer, ending up at top level.

To pass a numeric argument to the command you are invoking with *M-x*, specify the numeric argument before the *M-x*. *M-x* passes the argument along to the command it runs. The argument value appears in the prompt while the command name is being read.

Normally, when describing a command that is run by name, we omit the ⟨RET⟩ that is needed to terminate the name. Thus we might speak of *M-x af-version* rather than *M-x af-version* ⟨RET⟩. We mention the ⟨RET⟩ only when there is a need to emphasise its presence, such as when we show the command together with following arguments.

M-x works by running the command `execute-extended-command`, which is responsible for reading the name of another command and invoking it.

10 Help

Af provides extensive help features accessible through the command *C-h* (`help-command`). *C-h* is a prefix key that is used only for documentation-printing commands. The characters that you can type after *C-h* are called *help options*. One help option is ⟨C-h⟩; that is how you ask for help about using *C-h*. To cancel, type *C-g*.

C-h C-h displays a list of the possible help options. Typing *C-h* again displays a longer list of the options, each with a brief description.

10.1 Documentation for a Key

The most basic *C-h* options are *C-h c* (`describe-key-briefly`) and *C-h k* (`describe-key`). *C-h c key* prints in the echo area the name of the command that *key* is bound to. For example, *C-h c C-n* prints '`next-line`'. Since command names are chosen to describe what the commands do, this is a good way to get a very brief description of what *key* does.

C-h k key is similar but gives more information: it displays the documentation string of the command as well as its name. This is too big for the echo area, so typeout is used for the display.

10.2 Help by Command, Mode or Variable Name

C-h f (`describe-function`) reads the name of an af command, keyboard macro or afl function using the minibuffer, then displays that function's documentation string to typeout. For example,

```
C-h f delete-message ⟨RET⟩
```

displays the documentation of `delete-message`. This is the only way to get the documentation of a command that is not bound to any key (one which you would normally run using *M-x*).

C-h v (`describe-variable`) is like *C-h f* but describes configuration variables instead of commands and functions (see Section 21.4 [Variables], page 85).

C-h m (`describe-mode`) is also like *C-h f* but describes major and minor modes rather than commands and functions (see Section 3.3 [Major Modes], page 25).

10.3 Help for Lists of Things

C-h b (`list-bindings`) displays the current key bindings active in af to typeout. The global bindings are shown first, and then those for each of the major modes. This be quite a long list, but might be useful as a wall chart.

C-h d (`list-commands`) lists all of af's commands, with their bindings, to typeout. Bindings which only take effect in mail mode are listed in square brackets (eg '`[`⟨M-s⟩`]`'). Bindings for typeout mode only are listed in braces (eg '`{`⟨C-f⟩`}`'). Bindings that are minibuffer-specific are listed in parentheses (eg '`(`⟨SPC⟩`)`').

C-h C-f lists all of the afl functions to typeout. Only those functions which are not also commands are listed. This is quite a short list at present (see Chapter 22 [Afl], page 91).

C-h C-k `list-kbd-macros` lists all of the named keyboard macros, with their values, to typeout (see Section 21.5 [Keyboard Macros], page 86).

C-h C-v lists all of the af configuration variables, with their values, to typeout (see Section 21.4 [Variables], page 85).

C-h `C-a` (`list-aliases`) lists all the mail aliases you have defined to typeout, along with their values (see Section 8.7 [Aliases], page 48).

10.4 Apropos

A more sophisticated sort of question to ask is, "What is relevant for working with messages?" To ask this question, type *C-h* `a` *message* ⟨RET⟩, which displays a list of all command, function, keyboard macro and variable names that contain '`message`', including `open-message`, `print-message`, and so on. The bindings for each command is listed with that command, in the same format as list-bindings (see Section 10.3 [List Help], page 51). Similarly the values of keyboard macros and variables are listed. The `a` in *C-h* `a` stands for 'Apropos'; *C-h* `a` runs the command `command-apropos`.

Because *C-h* `a` looks only for things whose names contain the string which you specify, you must use ingenuity in choosing the string. If you are looking for commands for reading messages and *C-h* `a` *read-message* ⟨RET⟩ doesn't reveal any, don't give up. Try just *read*, or just *message*. Be persistent. Also note that you can use a regular expression as the argument, for more flexibility (see Section 15.4 [Regex Syntax], page 66).

Here is a set of arguments to give to *C-h* `a` that covers many classes of af commands, since there are strong conventions for naming the standard af commands. By giving you a feel for the naming conventions, this set should also serve to aid you in developing a technique for picking `apropos` strings.

> char, line, word, region, list, buffer, window, file, mode, beginning, end, forward, backward, next, previous, up, down, search, goto, kill, delete, mark, insert, yank, case, set, what, view, describe, default, print, page, message, send.

10.5 Other Help Commands

C-h i (`info`) runs the Info program, which is used for browsing through structured documentation files. The entire af manual is available within Info. Type *h* after entering Info to run a tutorial on using Info.

If something surprising happens, and you are not sure what commands you typed, use *C-h l* (`view-lossage`). *C-h l* prints the last 100 command characters you typed in. If you see commands that you don't know, you can use *C-h c* to find out what they do.

Af has numerous major modes, each of which redefines keys and alters the behaviour of some commands. *C-h j* (`describe-major-mode`) prints documentation on the current major mode to typeout.

The other *C-h* options display assorted useful information. *C-h n* (`view-af-news`) displays information on recent changes to af. *C-h C-w* (`display-no-warranty`) displays details on the complete absence of warranty for af. *C-h C-c* (`describe-copying`) displays the conditions you must obey in distributing copies of af.

11 More Ways to Handle Messages

You can use af to handle messages in more sophisticated ways than by simply displaying them; You can save messages to folders, print a hardcopy of them, pipe them into commands, and more.

Most commands that deal with a single message also have similar commands which deal with a set of messages, selected either by using the mark and region (see Chapter 12 [Mark and Region], page 56), or with tags (see Chapter 13 [Tags], page 58). These similar commands will be covered later.

11.1 Saving Messages

Af allows you to *save* messages by copying them to folders. One way to save a message to a folder is to use `M-+` (`save-message`). Af will prompt you for the folder name, offering a default based on the sender's mail address (see Chapter 18 [Folders], page 72). The message will be appended to the folder you specify, so any messages already in the folder will be left unchanged.

When the message is saved to a folder, the original copy is not deleted. an '`S`' will appear in the system tags of the message, so that you know you have saved it.

`M-+` can take a numeric argument, in which case it only saves the body of the message, not the headers. Obviously the file containing a saved copy of the message body shouldn't be read with af; it isn't a mail folder. Saving the body of a message can be useful when the body is input for some other program (such as a patch file).

When you save message bodies, any MIME encoding is removed from the body parts. Partial messages are rebuilt before they are saved. Af will only save a rebuilt message once, no matter how many parts of the message are included in the messages to save. This is convenient when you are saving a set of message bodies.

11.2 Printing Messages

Af allows you to print a hardcopy of a message to a spooler. The program to use to print hard copies of mail is defined by the `print-command` variable for textual body parts, or the `mime-printer` variable for non-textual body parts.

To print a hardcopy of a message, use `C-p` (`print-message`). Af will usually prompt you to confirm that you want a hardcopy of the message; this feature can be turned off by setting the variable `confirm-print` to `false`. After you print the message, a '`P`' will appear in the system tags of the message, to remind you that you have printed it.

Normally, the mail headers listed in the `headers-not-displayed` variable will not be printed when you print a message. A positive numeric argument makes af print all the message's headers, while a negative argument makes af skip all headers, and only print the body of the message.

When you print messages, any MIME encoding is removed from the body parts. Partial messages are rebuilt before they are printed. Af will only print a rebuilt message once, no matter how many parts of the message are included in the messages to print. This is convenient when you are printing a set of message bodies.

If you print a non-text message, af will use an external program to print any non-text body parts. For this reason, non-textual body parts may not appear in the correct sequence with textual body parts. If you print a non-text body part, and af doesn't know how to handle it, it will assume that you know what you are doing, and print it anyway. The results may well be less than ideal if you print an image or audio body part.

11.3 Piping Messages

Af allows you to *pipe* a message into the standard input of a system command. To do this use *M-|* (`pipe-message`). Af will prompt for the command to pipe the message into, and then execute the command. Any output from the command will be displayed on the screen. Once the command has completed, af will wait for you to press a key before continuing; giving you a chance to read any output from the command.

Normally, the mail headers listed in the `headers-not-displayed` variable will not be piped into the command. A positive numeric argument makes af pass all the message's headers to the command. A negative argument makes af skip all headers, and only pass the body of the message to the command.

When you pipe messages into a command, any MIME encoding is removed from the body parts. Partial messages are rebuilt before they are piped into the command. Af will only pipe a rebuilt message once, no matter how many parts of the message are included in the messages to pipe. This is convenient when you are piping a set of messages into a command

11.4 Other Message-Handling Commands

C-x j
: Create a new buffer containing the body parts of the current message. (`open-body-parts`).

C-x 4 j
: Create a new buffer in the other window containing the body parts of the current message. (`open-body-parts-other window`).

C-x C-e
: Edit the text of the current message.

If a message contains one or more body parts, or an encapsulated message, then af can create a buffer containing each body part as a separate message. Similarly, af can rebuild the original message from a single part if all the parts are present in the current buffer. Or if the message is an old, pre-MIME, mail digest, then af is still capable of extracting the parts of the message.

To extract or rebuild the parts of a message body use *C-x j* (open-body-parts) or *C-x 4 j* (open-body-parts-other-window). This will create a new buffer containing the extracted or rebuilt messages, and switch to that buffer. The buffer will not be visiting any folder, and so will not be saved to a folder when you exit af.

The new buffer will be in `Body Parts` minor mode, which means that the `body-part-line-format` variable will be used to display each message in the buffer, rather than the `header-line-format`. This allows you to show different details for body parts to those you show for messages.

Af will allow you to edit a message in a folder with *C-x C-e* (`edit-message`). Af will save the message to a temporary file and execute an editor to edit the file. When you finish editing, af will read back the temporary file, and update its internal copy of the message. When you save the buffer, the modified message will be written to the folder.

With an argument, *C-x C-e* will only pass the body of the message to the editor, and will preserve the message header unchanged.

11.5 Running Shell Commands from Af

Af has commands for running inferior shell processes, displaying the result to typeout or letting the inferior process take control of the terminal.

M-! cmd ⟨RET⟩
: Run the shell command line *cmd* and display the output to typeout (`shell-command-to-typeout`).

C-x ! cmd ⟨RET⟩
: Start an inferior shell to run the command line *cmd*. The inferior shell will take control of the terminal until it exits. (`shell-command`)

M-x shell
: Run an inferior shell interactively under af. The inferior shell will take control of the terminal until it exits (`shell`).

12 The Mark and the Region

Many af commands operate on an arbitrary contiguous part of the current buffer. To specify the text for such a command to operate on, you set *the mark* at one end of it, and move point to the other end. The text between point and the mark is called *the region*.

You can move point or the mark to adjust the boundaries of the region. It doesn't matter which one is set first chronologically, or which one comes earlier in the buffer. Once the mark has been set, it remains where you put it until you set it again at another place. Each af buffer has its own mark, so that when you return to a buffer that had been selected previously, it has the same mark it had before.

Many commands that insert messages (or text in the minibuffer), such as *C-y* (`yank`) and *C-x i* (`insert-file`), position point and the mark at opposite ends of the inserted text, so that the region contains the text just inserted.

12.1 Setting the Mark

There are two commands to set the mark in a buffer. These commands work identically in mail buffers, typeout, and the minibuffer.

C-⟨SPC⟩ or *C-@* or *M-*⟨SPC⟩
  Set the mark where point is (`set-mark-command`).

C-x C-x Interchange mark and point (`exchange-point-and-mark`).

For example, suppose you wish to save part of a mail buffer to a folder, using the *C-x +* (`save-region`) command, which operates on the messages in the region. You can first go to the beginning of the messages to be saved, type *C-*⟨SPC⟩ to put the mark there, move to the end, and then type *C-x +*. Or, you can set the mark at the end of the text, move to the beginning, and then type *C-x +*.

The most common way to set the mark is with the *C-*⟨SPC⟩ command (`set-mark-command`). This sets the mark where point is. Then you can move point away, leaving the mark behind.

Af doesn't show you where the mark is located. You have to remember. The usual solution to this problem is to set the mark and then use it soon, before you forget where it is. Alternatively, you can see where the mark is with the command *C-x C-x* (`exchange-point-and-mark`) which puts the mark where point was and point where the mark was. The extent of the region is unchanged, but point is now at the previous position of the mark.

C-x C-x is also useful when you are satisfied with the position of point but want to move the mark; do *C-x C-x* to put point at that end of the region, and then move it. A second use of *C-x C-x*, if necessary, puts the mark at the new position with point back at its original position.

There is no such character as *C-*⟨SPC⟩ in ASCII; when you type ⟨SPC⟩ while holding down ⟨CTRL⟩, what you get on most ordinary terminals is the character *C-@*. This key is actually bound to `set-mark-command`. But unless you are unlucky enough to have a terminal where typing *C-*⟨SPC⟩ does not produce *C-@*, you might as well think of this character as *C-*⟨SPC⟩. If you are on a terminal where *C-*⟨SPC⟩ doesn't work, you could use *M-*⟨SPC⟩ instead; it is also bound to `set-mark-command`.

12.2 Operating on the Region

Once you have a region in a mail buffer, here are some of the commands that operate on the region. Note that these commands all have the word `region` in their names.

C-w Kill the messages in the region (`kill-region`). See Chapter 14 [Killing and Yanking], page 62.

M-w Copy the region into the kill buffer as if it had been killed (`copy-region-as-kill`). See Chapter 14 [Killing and Yanking], page 62.

C-x + folder ⟨RET⟩
Save the messages in the region into *folder* (`save-region`). This command handles the argument in the same way as *M-+* (`save-message`). See Section 11.1 [Saving Messages], page 53.

C-x p Print a hardcopy of all the messages in the region (`print-region`). This command handles the argument in the same way as *M-p* (`print-message`). See Section 11.2 [Printing Messages], page 53.

C-x | command ⟨RET⟩
Pipe the messages in the region into a single instance of the shell command *command* (`pipe-region`). Again, this command handles the argument in the same way as *M-|* (`pipe-message`) See Section 11.3 [Piping Messages], page 54.

12.3 Using the Region in Typeout

Typeout is only used to display text, and not to edit or delete anything. For that reason, there are no commands which operate on the region available in typeout. It is still possible to set the mark in the typeout buffer; but the only thing you can do with it is to use *C-x C-x* (`exchange-point-and-mark`) to move back to the mark.

12.4 Using the Region in the Minibuffer

The minibuffer supports the mark and region, but only a subset of the commands which are available in a mail buffer to process the region. Since the minibuffer deals with text rather than messages, the region delimits a block of text in the minibuffer.

Just like any other buffer, the minibuffer has its own mark. The mark is not preserved between uses of the minibuffer, or when you move between minibuffer history entries, so you will usually need to set the mark in the minibuffer before you use it.

There are two commands which operate on the region in the minibuffer:

C-w Delete the text in the region (`kill-region`). See Section 14.5 [Minibuffer Kills], page 63.

M-w Copy the text in the region into the kill buffer as if it had been killed (`copy-region-as-kill`). See Section 14.5 [Minibuffer Kills], page 63.

13 Tags

The region allows you to conveniently process blocks of messages, but what if you want to handle an arbitrary set of messages? Af uses tags to let you mark a set of messages and process them quickly.

Tags are only available in mail buffers; they wouldn't make any sense in the minibuffer or the mail buffer.

13.1 Tag Concepts

A *tag* is a single-character mark which is set on a message. A message can have any number of tags set on it. There are two types of tags: *system tags* which are automatically set by af for various reasons, and *user tags* which you can set and unset yourself. You cannot set or remove system tags, but you can use them in exactly the same ways as you can user tags.

There are many commands which operate on tagged messages; and you can use either single tags or a combination of them to specify a set of messages to operate on. Tags are central to advanced use of af; they provide a way for you to easily set up a set of messages, which you can then work with in several ways.

13.1.1 System Tags

The system tags are all upper-case letters, which give information on the status of messages. The possible system tags are:

`D` The message has been marked for deletion with *`M-x delete-message`*.

`E` Af found an error in the message's headers when the folder was visited. This is not normally critical, but you may not be able to reply to the message; or it may even (rarely) indicate a corrupted folder.

`F` You have forwarded or bounced the message (see Chapter 8 [Sending Mail], page 41).

`M` The message is in MIME format; and isn't text that you can display on your terminal. When you read the message, af will display each body part separately if there are more that one, and will use external commands to display any non-textual body parts. If af doesn't know how to display a given body part then it will ask you what to do.

`T` The message is in MIME format, and is textual, but is either not plain text, or is in a character set not listed in the `viewable-charsets` variable. When you read the message then it will be displayed as usual, which may not be ideal.

`N` The message is new; the message has arrived since last time you visited the folder and hasn't yet been read.

`O` The message is old but unread. The message was present last time you visited the folder, but you haven't read it yet.

`P` A hardcopy of the message has been printed to a spooler.

'R' You have replied to the message (see Chapter 8 [Sending Mail], page 41).

'S' The message has been saved to a folder.

13.1.2 User Tags

The user tags you can use are the lower case letters, and '+'. '+' is sometimes called the *default tag*, since it is the default whenever you are asked about tags.

Normally, the user tags are considered transient, and are lost when you exit af. If you would like to keep some or all of the tags across mail reading sessions, then you can set the variable `persistent-tags` to the tags you would like to be preserved. For example, setting the value of `persistent-tags` to 'aeiou', would mean that the user tags 'a', 'e', 'i', 'o', and 'u' will be preserved whenever you save a folder, but the other tags will be lost. You cannot make the default tag ('+') persistent.

13.1.3 Tag Lists

Whenever you are setting or removing tags, af will expect you to enter a *tag list*. Tag lists consist of one or more user tags, which may be separated by spaces. For example, 'abcd' and 'a b c d', both specify a list of four tags, 'a', 'b', 'c', and 'd'.

Whenever you are prompted for a tag list, you can just hit return to use the default tag ('+').

13.1.4 Tag Expressions

When you are using tags to define a set of messages to apply some command to, you will be expected to enter a *tag expression*. That tag expression is then compared to the tags of each message in the buffer, and the command will operate on the message if the expression is true for that message. We often refer to the messages which match a tag expression as the *tagset*.

Tag expressions are made up of sub-expressions, which may be system or user tags. You can use operators to combine subexpressions, and build arbitrarily complex expressions. Any tag will evaluate to true if the message being checked has that tag set, false otherwise. The possible operators are, in decreasing precedence:

'(...)' Override any precedence, and force the contents of the parentheses to be evaluated as a unit. '(a)' is equivalent to 'a'.

'!' Logical not; true if the expression it precedes is false. '!a' is true if 'a' is not set.

'&' Logical and; true if both sides of the expression are true. 'a & b' is true if both 'a' and 'b' are set.

'^' Logical exclusive-or; true if only one side of the expression is true. 'a ^ b' is true if either 'a' or 'b' is set, but not both.

'|' Logical or; true if either side of the expression is true. 'a | b' is true if either 'a' or 'b' is set, or if both are set.

Here are some examples of tag expressions, with their meanings.

```
a & b | c & d
```

True if '`a`' and '`b`' are both set, or '`c`' and '`d`' are both set.

```
a & (b | c) & d
```

True if '`a`', '`d`', and either '`b`' or '`c`' are set.

```
a & !b
```

True if '`a`' is set and '`b`' is not set.

```
a & !(b | c)
```

True if '`a`' is set and neither of '`b`' or '`c`' are set.

13.2 Setting and Removing Tags

`C-t t` tags ⟨RET⟩
: Set *tags* on the current message (`tag-message`).

`C-t u` tags ⟨RET⟩
: Remove *tags* from the current message (`untag-message`).

`C-t r` tags ⟨RET⟩
: Remove *tags* from all the messages in the buffer (`remove-tag`).

`M-x tag-thread` tags ⟨RET⟩

To tag the current message, use *`C-t t`*. You will be prompted for the tags to set; simply type in the tag list to set. The tags in the list will be added to those already set on the message.

To remove one or more tags from the current message use *`C-t u`*. You will be prompted for the tags to remove. The tags you specify will be removed from the message's tags.

To remove one or more tags from all the messages in the buffer, use *`C-t r`*. You will be prompted for the tags to remove. The tags you specify will be removed from all the messages in the buffer.

The experimental command *`M-x tag-thread` tags* ⟨RET⟩ tags all the messages in a *thread* of conversation, a set of messages which are all sent as replies to other related messages. (This is similar to the concept of threading found in most news readers.) This command works, but at the moment there are too many mail composers which don't include references to messages when they generate a reply, so that the threads are often broken by replies which af can't detect as a part of the thread. It is probably best not to use this command for the moment, unless you can be sure that all the participants are using a mail composer that does include references to messages in replies.

There is also a command to set tags on messages which match search criteria. See Section 15.3 [Search and Tag], page 66.

13.3 Using Tags

Once you have tagged messages in the region, you can use single tags or tag expressions to operate on them. Here are some of the ways in which you can operate on tagged messages:

C-t C-k tag-expr ⟨RET⟩
: Kill the messages in the tagset defined by *tag-expr* (`kill-tagset`). See Chapter 14 [Killing and Yanking], page 62.

C-t w tag-expr ⟨RET⟩
: Copy the messages in the tagset into the kill buffer as if they had been killed (`copy-tagset-as-kill`). See Chapter 14 [Killing and Yanking], page 62.

C-t + tag-expr ⟨RET⟩ *folder* ⟨RET⟩
: Save the messages in the tagset into *folder* (`save-tagset`). This command handles the argument in the same way as *M-+* (`save-message`). See Section 11.1 [Saving Messages], page 53.

C-t p tag-expr ⟨RET⟩
: Print a hardcopy of all the messages in the tagset (`print-tagset`). This command handles the argument in the same way as *M-p* (`print-message`). See Section 11.2 [Printing Messages], page 53.

C-t | tag-expr ⟨RET⟩ *folder* ⟨RET⟩
: Pipe the messages in the tagset into a single instance of the shell command *command* (`pipe-tagset`). Again, this command handles the argument in the same way as *M-|* (`pipe-message`) See Section 11.3 [Piping Messages], page 54.

14 Killing and Yanking

Killing means erasing messages (or text in the minibuffer) and copying them into the *kill ring*, from which it can be retrieved by *yanking* it. Some systems use the terms "cutting" and "pasting" for these operations.

The commonest way of moving or copying messages within af is to kill them, and later yank them elsewhere one or more times. This is very safe because af remembers several recent kills, not just the last one. It is versatile, because the many commands for killing messages can also be used for moving them.

Af has only one kill ring for all mail buffers, so you can kill messages in one buffer and yank them in another buffer. The minibuffer has a separate kill ring for storing text.

All commands which delete messages from the buffer save it in the kill ring (even `delete-message` when you save the folder) so that you can move or copy it to other parts of the buffer. These commands are known as *kill* commands.

14.1 Killing Messages

There are two ways to add messages to the kill ring; killing and copying. *Killing* means deleting messages and adding them to the kill ring, while *copying* means copying messages to the kill ring without deleting them.

There are commands to kill a single message (see Section 5.4 [Deleting Messages], page 32), commands to kill or copy the region (see Section 12.2 [Using the Region], page 57), and commands to kill or copy a tagset (see Section 13.3 [Using Tags], page 60).

14.2 Yanking Messages

All killed messages are recorded in the *kill ring*, a list of blocks of messages that have been killed. There is only one kill ring, shared by all mail buffers, so you can kill messages in one buffer and yank them in another buffer. This is the usual way to move messages from one folder to another (See Section 11.1 [Saving Messages], page 53, for another way).

The command *C-y* (`yank`) reinserts the most recently killed messages. It leaves the cursor at the end of the text. It sets the mark at the beginning of the text (see Chapter 12 [Mark and Region], page 56).

C-y can take a numeric argument, in which case it select an earlier kill to yank, rather than the last (see Section 14.4 [Earlier Kills], page 63).

14.3 Appending Kills

Normally, each kill command pushes a new entry onto the kill ring. However, two or more kill commands in a row combine their messages into a single entry, so that a single *C-y* yanks all the messages as a unit, just as they were before they were killed.

Thus, if you want to yank messages as a unit, you need not kill all of them with one command; you can keep killing line after line, until you have killed them all, and you can still get them all back at once.

Commands that kill forward from point add the messages onto the end of the previous killed messages. Commands that kill backward from point add them onto the beginning. This way, any sequence of mixed forward and backward kill commands puts all the killed messages into one entry without rearrangement. Numeric arguments do not break the sequence of appending kills.

14.4 Yanking Earlier Kills

To recover killed messages that are no longer the most recent kill, use the *M-y* command (`yank-pop`). It takes the messages previously yanked and replaces them with the messages from an earlier kill. So, to recover the messages of the next-to-the-last kill, first use *C-y* to yank the last kill, and then use *M-y* to replace it with the previous kill. *M-y* is allowed only after a *C-y* or another *M-y*.

You can understand *M-y* in terms of a "last yank" pointer which points at an entry in the kill ring. Each time you kill, the "last yank" pointer moves to the newly made entry at the front of the ring. *C-y* yanks the entry which the "last yank" pointer points to. *M-y* moves the "last yank" pointer to a different entry, and the text in the buffer changes to match. Enough *M-y* commands can move the pointer to any entry in the ring, so you can get any entry into the buffer. Eventually the pointer reaches the end of the ring; the next *M-y* moves it to the first entry again.

M-y moves the "last yank" pointer around the ring, but it does not change the order of the entries in the ring, which always runs from the most recent kill at the front to the oldest one still remembered.

M-y can take a numeric argument, which tells it how many entries to advance the "last yank" pointer by. A negative argument moves the pointer toward the front of the ring; from the front of the ring, it moves "around" to the last entry and continues forward from there.

Once the messages you are looking for are brought into the buffer, you can stop doing *M-y* commands and it will stay there. They are a copy of the kill ring entry, so altering them in the buffer does not change what's in the ring. As long as no new killing is done, the "last yank" pointer remains at the same place in the kill ring, so repeating *C-y* will yank another copy of the same previous kill.

If you know how many *M-y* commands it would take to find the messages you want, you can yank those messages in one step using *C-y* with a numeric argument. *C-y* with an argument restores the text the specified number of entries back in the kill ring. Thus, *C-u 2 C-y* gets the next to the last block of killed messages. It is equivalent to *C-y M-y*. *C-y* with a numeric argument starts counting from the "last yank" pointer, and sets the "last yank" pointer to the entry that it yanks.

The length of the kill ring is controlled by the variable `kill-ring-max`; no more than that many blocks of killed messages are saved.

14.5 Killing and Yanking in the Minibuffer

The minibuffer is used for editing text rather than handling mail, so it has it's own kill ring. Any command which kills more than one character at a time (the `word` and `line` killing commands) will store the killed text in the minibuffer kill ring. You can yank the killed text back in exactly the same way as killed messages using *C-y*.

The minibuffer kill commands will append successive kills to the kill ring, just like the mail buffer kill commands do. If you use *M-d* (`delete-word`) three times, then the three words you kill will all be stored in a single kill ring entry; and a single yank will insert them into the text at point.

For simplicity, the minibuffer kill ring only has one entry, so you can't use *M-y* in the minibuffer. If many people find this a problem, then it may change at some point in the future.

15 Searching Mail Folders

Af has search commands, which allow you to search through the messages in a buffer and find those which match a regular expression. You can also search for messages which match a tag expression.

15.1 Searching for Regular Expressions

`C-s` *regex* ⟨RET⟩
: Search for *regex* (`search-forward`).

`C-r` *regex* ⟨RET⟩
: Search backward for *regex* (`search-backward`).

To do a search on a buffer (whether typeout or a mail buffer), use `C-s` or `C-r`. Af will prompt you for the regular expression to search for, and then the search takes place. If no messages match the regular expression then the search will fail with an error.

A second search immediately after the first, will not match the current message, so repeated searches will move through all the messages which match the regular expression. To make this more convenient, the search expression is defaulted to the last one you entered.

The search commands with a numeric argument will only search the headers of the messages. This is often convenient when (for example) looking for messages which are from a particular person.

With a negative numeric argument the search commands will only search the bodies of the messages. This can be useful when (for example) looking for messages which mention your machine's host name (which is included in the headers of all messages).

15.2 Searching for Tagged Messages

`C-t C-s` *tagexpr* ⟨RET⟩
: Search for messages matching *tagexpr* (`tag-search-forward`).

`C-t C-r` *tagexpr* ⟨RET⟩
: Search backward for messages matching *tagexpr* (`tag-search-backward`).

To search for a message matching a tag expression use `C-t C-s` or `C-t C-r`. Af will prompt you for the tag expression (see Section 13.1.4 [Tag Expressions], page 59) to search for; and then the search takes place. If no messages match the tag expression then the search will fail with an error.

Just as with regular expression searches, a second search immediately after the first, will not match the current message, so repeated searches will move through all the messages which match the tag expression. To make this more convenient, the search expression is defaulted to the last one you entered.

15.3 Tagging Matching Messages

As well as just searching for a regular expression and moving point to the first matching message, af can tag all the messages which match a regular expression. To do this use `C-t s` (`search-and-tag`). You will be prompted for the regular expression to search for, and the tags to set on the matching messages (see Section 13.2 [Setting Tags], page 60). Once the search has finished, af will report how many messages were tagged.

With a numeric argument this command will only search the headers of the messages. This is often convenient when (for example) looking for messages which are from a particular person or mailing list.

15.4 Syntax of Regular Expressions

Regular expressions have a syntax in which a few characters are special constructs and the rest are *ordinary*. An ordinary character is a simple regular expression which matches that same character and nothing else. The special characters are '`$`', '`^`', '`.`', '`*`', '`+`', '`?`', '`[`', '`]`' and '`\`'. Any other character appearing in a regular expression is ordinary, unless a '`\`' precedes it.

For example, '`f`' is not a special character, so it is ordinary, and therefore '`f`' is a regular expression that matches the string '`f`' and no other string. (It does *not* match the string '`ff`'.) Likewise, '`o`' is a regular expression that matches only '`o`'. (When case distinctions are being ignored, these regular expressions also match '`F`' and '`O`', but we consider this a generalization of "the same string", rather than an exception.)

Any two regular expressions *a* and *b* can be concatenated. The result is a regular expression which matches a string if *a* matches some amount of the beginning of that string and *b* matches the rest of the string.

As a simple example, we can concatenate the regular expressions '`f`' and '`o`' to get the regular expression '`fo`', which matches only the string '`fo`'. Still trivial. To do something nontrivial, you need to use one of the special characters. Here is a list of them.

`.` (Full stop)
: is a special character that matches any single character except a newline. Using concatenation, we can make regular expressions like '`a.b`' which matches any three-character string which begins with '`a`' and ends with '`b`'.

`*`
: is not a construct by itself; it is a postfix operator, which means to match the preceding regular expression repetitively as many times as possible. Thus, '`o*`' matches any number of '`o`'s (including no '`o`'s).

  '`*`' always applies to the *smallest* possible preceding expression. Thus, '`fo*`' has a repeating '`o`', not a repeating '`fo`'. It matches '`f`', '`fo`', '`foo`', and so on.

`+`
: is a postfix character, similar to '`*`' except that it must match the preceding expression at least once. So, for example, '`ca+r`' matches the strings '`car`' and '`caaaar`' but not the string '`cr`', whereas '`ca*r`' matches all three strings.

`?`
: is a postfix character, similar to '`*`' except that it can match the preceding expression either once or not at all. For example, '`ca?r`' matches '`car`' or '`cr`'; nothing else.

[...] is a *character set*, which begins with '[' and is terminated by ']'. In the simplest case, the characters between the two brackets are what this set can match.

Thus, '[ad]' matches either one 'a' or one 'd', and '[ad]*' matches any string composed of just 'a's and 'd's (including the empty string), from which it follows that 'c[ad]*r' matches 'cr', 'car', 'cdr', 'caddaar', etc.

You can also include character ranges in a character set, by writing two characters with a '-' between them. Thus, '[a-z]' matches any lower-case letter. Ranges may be intermixed freely with individual characters, as in '[a-z$%.]', which matches any lower case letter or '$', '%' or '.'.

Note that the usual regex special characters are not special inside a character set. A completely different set of special characters exists inside character sets: ']', '-' and '^'.

To include a ']' in a character set, you must make it the first character. For example, '[]a]' matches ']' or 'a'. To include a '-', write '-' as the first or last character of the set. Thus, '[]-]' matches both ']' and '-'.

To include '^', make it other than the first character in the set.

[^ ...] '[^' begins a *complemented character set*, which matches any character except the ones specified. Thus, '[^a-z0-9A-Z]' matches all characters *except* letters and digits.

'^' is not special in a character set unless it is the first character. The character following the '^' is treated as if it were first ('-' and ']' are not special there).

^ is a special character that matches the empty string, but only at the beginning of a line in the text being matched. Otherwise it fails to match anything. Thus, '^foo' matches a 'foo' which occurs at the beginning of a line.

*$* is similar to '^' but matches only at the end of a line. Thus, 'xx*$' matches a string of one 'x' or more at the end of a line.

\ has two functions: it quotes the special characters (including '\'), and it introduces additional special constructs.

Because '\' quotes special characters, '\$' is a regular expression which matches only '$', and '\[' is a regular expression which matches only '[', etc.

For the most part, '\' followed by any character matches only that character. However, there are several exceptions: two-character sequences starting with '\' which have special meanings. The second character in the sequence is always an ordinary character on its own. Here is a table of "\" constructs.

\{ ... \} is a postfix construct, similar to '*' except that it allows you to specify the number of times the preceding expression must be matched. So, for example, 'ca\{3\}r' will match only the string 'caaar'.

If you add a comma after the number of times the expression must be matched, then the expression must be matched at least as many times as you specified. So 'ca\{2,\}r' will match the strings 'caar', 'caaar', 'caaaar', and so on.

You can also add a maximum value after the comma, to specify a range of values. So `ca\{1,3\}r` will match only the strings `car`, `caar` and `caaar`.

`\|` specifies an alternative. Two regular expressions *a* and *b* with `\|` in between form an expression that matches anything that either *a* or *b* matches.

Thus, `foo\|bar` matches either `foo` or `bar` but no other string.

`\|` applies to the largest possible surrounding expressions. Only a surrounding `\( ... \)` grouping can limit the scope of `\|`.

`\( ... \)` is a grouping construct that serves three purposes:

1. To enclose a set of `\|` alternatives for other operations. Thus, `\(foo\|bar\)x` matches either `foox` or `barx`.
2. To enclose a complicated expression for the postfix operators `*`, `+` and `?` to operate on. Thus, `ba\(na\)*` matches `bananana`, etc., with any (zero or more) number of `na` strings.
3. To mark a matched substring for later reference with `\`*N*.

`\`*N* When you use `\( ... \)` in an expression, you can look for another match for the exact same text that was matched inside the `\( ... \)`. The two-character sequence `\`*N* will match the same text as was matched by the *N*th use of `\( ... \)`. The first nine uses are remembered, and are assigned the numbers `1` to `9`. So `\1` matches the text that was matched by the first use of `\( ... \)`.

For example, `\([a-z]\)\1` matches any two consecutive lower case characters. The `\([a-z]\)` matches any lower case character, while the `\1` must match the same character.

If a use of `\( ... \)` matches more than once, which often happens if it is followed by `*` or `+`, only the last match is stored for use with `\`*N*.

Here is a moderately complicated regex, which you might use to find messages from the af-bug or af-user mailing lists.

```
^From:.*af-\(bug\|user\)@csv.warwick.ac.uk
```

15.5 Searching and Case

Searches in af normally ignore the case of the text they are searching through. Thus, if you specify searching for `foo`, then `Foo` and `foo` are also considered a match. Regular expressions, and in particular character sets, are included: `[ab]` would match `a` or `A` or `b` or `B`.

If you set the variable `case-fold-search` to `false`, then all letters must match exactly, including case.

16 Sorting Messages

Af provides commands for sorting some or all of the messages in a mail buffer. Each command prompts for a sort order, and then sorts the selected messages into that order.

M-x sort-buffer order RET
: Sort all of the messages in the buffer into *order* order.

M-x sort-region order RET
: Sort the messages in the region into *order* order.

M-x sort-tagset order RET
: Sort the messages in the tagset into *order* order. The messages in the tagset are sorted correctly, but messages which are not in the tagset are not moved at all.

The sort orders which af understands are:

`(reverse-)address`
: The address of the sender of the message, sorted case-independent. Any full name or routing information in the address is ignored.

`(reverse-)date`
: The date the message was sent (or received if the date sent isn't known), from oldest to newest (newest to oldest). The variable `show-dates-in-local-time` indicates whether dates should be compared as they are, or converted to your local time before they are compared (the default).

`(reverse-)lines`
: The number of lines in the message, from smallest to largest (largest to smallest).

`(reverse-)mailbox`
: The (reverse) order of the messages in the folder when it was read. Messages which were yanked into the buffer after it was read may appear anywhere in the buffer.

`(reverse-)sender`
: The full name of the sender of the message, sorted case-independent.

`(reverse-)status`
: The system tags of the message (see Section 13.1.1 [System Tags], page 58). Messages with no system tags are sorted after those that have them.

`(reverse-)subject`
: The subject of the message, sorted alphabetically and case-independent. The prefix 'Re: ' is handled specially in subject lines, so that replies to a message will sort immediately after (before) the original message.

`(reverse-)tags`
: The user tags of the message (see Section 13.1.2 [User Tags], page 59). Messages with no user tags are sorted after those that have them.

When you sort, messages which sort identically will remain in their original order, so sorting a buffer by date and then subject will result in the messages being in date order within each subject. This is often a good approximation to the "threading" that news readers such as `trn` do, and is handy for reading mailing lists.

Af does not sort your folders by default when it reads them. If you have a preference for reading buffers in a certain order, then you can set the configuration variable `initial-buffer-sort` to the sort ordering you prefer.

If you are familiar with other mail readers it is worth noting that af, unlike most other mail readers, saves the messages back to the folders in whatever order you sort them into. This can often be convenient when managing large folders.

17 Narrowing

Narrowing means focusing in on some portion of the buffer, making the rest temporarily inaccessible. The portion which you can still get to is called the *accessible part*. Canceling the narrowing, which makes the entire buffer once again accessible, is called *widening*. The amount of narrowing in effect in a buffer at any time is called the buffer's *restriction*.

Narrowing can make it easier to concentrate on a set of messages by eliminating clutter. It can also be used to restrict the range of operation of a search command or repeating keyboard macro.

C-x n Narrow down to between point and mark (`narrow-to-region`).

C-t n expression RET
Narrow to the tagset defined by tag-expression (`narrow-to-tagset`).

C-x w Widen to make the entire buffer accessible again (`widen`).

When you have narrowed down to a part of the buffer, that part appears to be all there is. You can't see the rest, you can't move into it (motion commands won't go outside the accessible part), you can't change it in any way. However, it is not gone, and if you save the folder all the inaccessible messages will be saved. The word '`Narrow`' appears in the mode line whenever narrowing is in effect.

One narrowing command is *C-x n* (`narrow-to-region`). It sets the current buffer's restrictions so that the messages in the region remain accessible but all messages before or after the region are inaccessible. Point and mark do not change.

Another narrowing command is *C-t n* (`narrow-to-tagset`). It prompts for a tag expression, and then set the current buffer's restrictions so that the messages in the tagset remain accessible but all the other messages are inaccessible.

The way to cancel narrowing is to widen with *C-x w* (`widen`). This makes all the messages in the buffer accessible again.

Narrowing can be very useful if you have a set of messages that you want to deal with independently of any others; such as all the messages from a mailing list, or from a certain user. Use *C-t s* (`search-and-tag`) to tag the messages you are interested in, and then use *C-t n* to narrow the buffer to only the tagged messages. When you are done with the messages, use *C-x w* to widen to buffer.

Another use for narrowing is when you are using tags to kill messages. You can use *C-t n* to narrow to the messages you intend to kill, and then conveniently scan the messages to make sure that you are happy to kill them all; using *C-t u* to untag any messages that you decide you want to keep. Finally you can either kill the tagset with *C-t C-k* (`kill-tagset`), or set the mark and point at opposite ends of the buffer and use *C-w* (`kill-region`). When you have finished, use *C-x w* to widen the buffer.

18 Folder Handling

The operating system stores data permanently in named *files*. We refer to files which contain mail messages as *folders*. In general, the mail messages you process with af come from a folder which will need to be updated to reflect any changes you make.

Unlike most mail readers, af can read in and process several folders simultaneously, which makes it convenient to store messages in several folders rather than leaving them all in your incoming mailbox. Af can actually be used quite conveniently as a small database by using folders to store related messages.

By default, af reads in your incoming mailbox when it starts. You can also use command-line arguments to tell af to read a different folder (see Section 4.1 [Command Arguments], page 26). Or you can tell an existing af to read another folder and prepare a buffer containing a copy of the folder's messages. This is called *visiting* the folder. Af commands apply to the messages in the buffer; that is, to the copy inside af. Your changes appear in the folder itself only when you *save* the buffer back into the folder.

18.1 File Names

Most af commands that operate on a file require you to specify the file name. (Saving and reverting are exceptions; the buffer knows which file name to use for them.) You enter the file name using the minibuffer (see Chapter 7 [Minibuffer], page 36). *Completion* is available, to make it easier to specify long file names (see Section 7.6 [Completion], page 38).

Many people prefer to keep all their mail folders in one directory; any folders saved from Usenet news (news folders) in another, and so on. To make this kind of operation more convenient, af interprets several characters specially if they appear at the start of the file name.

'~*user*'
: *user*'s home directory, or your home directory if *user* is not given. For example '`~/foo`' is the file '`foo`' in your home directory, while '`~malc/foo`' is the file '`foo`' in malc's home directory.

'%*user*'
: *user*'s incoming mailbox, or your incoming mailbox if *user* is not given. So '`%`' is the your incoming mailbox, while '`%malc`' is malc's incoming mailbox. Note that you won't usually have the necessary permission to read other people's incoming mailboxes.

'`+`'
: Your *folder directory*, as specified by the variable `folder`. This is either extracted from the environment variable `FOLDER`, or defaulted to '`~/Mail`'. So '`+af`' will usually expand to '`~/Mail/af`'. You will need to create the directory yourself if you plan to use it; af won't create it for you.

'`=`'
: Your *news folder directory*, where saved news articles are stored, as specified by the variable `news-folder`. This is either extracted from the environment variable `SAVEDIR`, or defaulted to '`~/News`'. So '`=Comp.mail.headers`' will usually expand to '`~/News/Comp.mail.headers`'. Normally your news-reader will have created this directory for you.

'`\`'
: A leading '`\`' in a file name is ignored, but it prevents any special meaning being taken from the next character in the filename. To start a file name with a '`\`', use '`\\`'. For example '`\+af`' is the file '`+af`' in the current directory.

Af has a *current directory*, sometimes referred to as the *working directory*, which is normally the directory you were in when you ran af. When you enter a folder name without a directory or a special prefix, the current directory is used. If you specify a directory in a relative fashion, with a name that does not start with either a special prefix or a slash, it is interpreted with respect to the current directory.

For example, if the current directory is '`/home/malc/src/af`' and you type just '`foo`', which does not specify a directory, it is short for '`/home/malc/src/af/foo`'. '`../../.afrc`' would stand for '`/home/malc/.afrc`'. '`new/foo`' would stand for the file name '`/home/malc/src/af/new/foo`'.

The command *`M-x pwd`* prints af's current directory, and the command *`M-x cd`* sets it (to a value read using the minibuffer). Af's current directory can only be changed by using the `cd` command.

18.2 Reading Folders Over a Network

Af supports a mail protocol called POP3, which allows you to read a folder over a network. This allows people to read mail on a small machine which doesn't have a mail system itself; you simply connect to the POP *server* where your mail is stored, and read your mail over the network.

Your incoming mailbox may live on a POP server, or you may have a separate POP3 mailbox somewhere. To access a POP3 folder use a filename beginning in a colon. Two forms of POP3 folder name are acceptable: '`:`*server*' or '`:`*user*`@`*server*'. The first form implies that the user name to use on *server* is the same one that you are currently logged in as; the second explicitly specifies the user name to use to connect to the POP server.

Once af has opened a connection to the POP server, you are prompted for a password. The password won't be echoed (for security reasons), and it won't be stored in the minibuffer history either. If you don't know your password then you will need to check with your system administrator to find out what it is. The password you use to log in is often worth trying.

Once af has read the POP folder (which may like some time; network connections are often slow), it displays the buffer as normal. However, the buffer containing the folder will have the POP3 minor mode set. This disallows several af features, notably killing and yanking messages in the buffer. The reason for this is that the POP3 protocol was designed to work well with existing mail readers, and it doesn't have the functionality to cope with af's method of folder management.[1] One way of working around this is to use a pending folder (see Section 18.4 [Pending Folders], page 75).

18.3 Visiting Files

`C-x C-f` Visit a folder (`find-file`).

`C-x C-r` Visit a folder for viewing, without allowing changes to it (`find-file-read-only`).

`C-x C-v` Visit a different folder instead of the one visited last (`find-alternate-file`).

[1] This isn't really POP3's fault; the design of af and of other mailers is very different in several areas.

C-x 4 C-f Visit a file, in another window (`find-file-other-window`).

Visiting a folder means reading its contents into an af buffer so you can process them. Af makes a new buffer for each folder that you visit. We say that this buffer is visiting the folder that it was created to hold. Af constructs the buffer name from the folder name by throwing away the directory, keeping just the name proper. For example, a file named '`/home/malc/Mail/af`' would get a buffer named '`af`'. If there is already a buffer with that name, a unique name is constructed by appending '`<2>`', '`<3>`', or so on, using the lowest number that makes a name that is not already in use.

Each window's mode line shows the name of the buffer that is being displayed in that window, so you can always tell what buffer you are processing.

The changes you make with af commands are made in the af buffer. They do not take effect in the folder that you visited, or any place permanent, until you *save* the buffer. Saving the buffer means that af writes the current contents of the buffer into its visited folder. See Section 18.6 [Saving], page 76.

If a buffer contains changes that have not been saved, we say the buffer is *modified*. This is important because it implies that some changes will be lost if the buffer is not saved. The mode line displays two stars near the left margin to indicate that the buffer is modified.

If the status of one or more messages in a buffer has changed (perhaps they were new before you read the folder), but no other changes have been made, then we say that the buffer is *status modified*. This is not so important as being modified, but some information will be lost if the buffer is not changed. The mode line displays two plus signs near the left margin to indicate that the buffer is status modified.

To visit a folder, use the command *C-x C-f* (`find-file`). Follow the command with the name of the folder you wish to visit, terminated by a ⟨RET⟩.

The folder name is read using the minibuffer (see Chapter 7 [Minibuffer], page 36), with defaulting and completion in the standard manner (see Section 18.1 [File Names], page 72). While in the minibuffer, you can abort *C-x C-f* by typing *C-g*.

If the specified folder does not exist and could not be created, or cannot be read, then you get an error, with an error message displayed in the echo area.

While af is reading the folder, it reports on how many messages it has read, usually after every five messages read. You can control how often af updates the count by setting the variable `message-count-update`. Once the folder has been read, af will print a message like this:

```
(Read 15 messages)
```

This confirms that af has successfully read the folder into a buffer. The message may also sometimes look like this:

```
(Read 15 messages; including 1 with bad headers)
```

This means that the folder was read successfully, but some of the messages in the folder had mail headers that af couldn't interpret properly. This is not usually critical, but some people prefer to know about errors in their mailboxes.

If the variable `first-unread-message` is set to `true`; then af will move point to the first message in the buffer that you haven't read, if there are any. Otherwise you will begin processing the buffer with point on the first message.

If you visit a file that is already in af, `C-x C-f` does not make another copy. It selects the existing buffer containing that file.

What if you want to create a new folder? Just visit it. Af prints '(New File)' in the echo area, but in other respects behaves as if you had visited an existing empty file. If you make any changes and save them, the file is created.

If you visit a file that the operating system won't let you modify, Af makes the buffer read-only, so that you won't go ahead and make changes that you'll have trouble saving afterward. You can make the buffer writable with `C-x C-q` (`toggle-read-only`).

Occasionally you might want to visit a folder as read-only in order to protect yourself from entering changes accidentally; do so by visiting the file with the command `C-x C-r` (`find-file-read-only`).

If you accidentally visit the wrong folder (perhaps you mistyped the file name), use the `C-x C-v` command (`find-alternate-file`) to visit the file you really wanted. `C-x C-v` is similar to `C-x C-f`, but it kills the current buffer (after first offering to save it if it is modified).

`C-x 4 f` (`find-file-other-window`) is like `C-x C-f` except that the buffer containing the specified file is selected in another window. The window that was selected before `C-x 4 f` continues to show the same buffer it was already showing. If this command is used when only one window is being displayed, that window is split in two, with one window showing the same buffer as before, and the other one showing the newly requested file. See Chapter 20 [Windows], page 80.

18.4 Pending Folders

If you are reading mail via POP3, you may prefer to store your mail on your local machine, rather than on a POP server. You can do this conveniently with af by using a *pending folder*, a folder which contains two folders. The simplest way of doing this is to set the variable `pending-folder` to the name of the folder where all your incoming mail should be stored; say '`+mailbox`'. You must set the `pending-folder` variable in your '`.afrc`' file, or it will have no affect (see Section 21.4 [Variables], page 85).

Once the `pending-folder` variable is set, it changes the behaviour of af when you start without specifying folders to read or addresses to send mail to. Instead of reading your incoming mailbox into its own buffer, it reads the folder specified in the `pending-folder` variable, and then appends the contents of your incoming mailbox to the buffer.

The buffer is considered to contain both the *pending folder* and your incoming mailbox; it will be resynchronised from both, and visiting either folder will switch to the buffer displaying the pending folder. When you save the pending folder your incoming mailbox will be cleared, ready for any new mail. If you don't save the buffer, both folders are left untouched.

One advantage of pending folders is that they are normal disk folders. If your incoming mailbox is read via POP3, using a pending folder will allow you to avoid the limitations of buffers in POP3 mode, and use all of af's features in the buffer. Once you've set it up, using a default pending folder is transparent enough that you'll probably never notice the difference.

You can also read a folder into an existing buffer, making the buffer pending on the folder you read. To do this use the command *M-x read-pending-file*. You will be prompted for the name of the file to make the buffer pending on. This has exactly the same effect as using the `pending-folder` variable.

18.5 Resynchronizing Buffers

As we have discussed, buffers are copies of the contents of a folder. So what can we do to handle new mail arriving in the folder after we have read it? The solution to this is *resynchronizing* the buffer, so that any new messages in the file on disk are incorporated into the buffer.

When a buffer is resynchronised, any new messages are appended to the buffer, and a message will appear in the echo area to let you know that new messages were read into the buffer.

Af will check for new messages whenever you try to save a buffer to disk. If there are new messages then the buffer is resynchronised, and the save will fail.

Af also checks for new mail in each folder you are visiting on a regular basis. If any buffers are out of date, then they are resynchronised. How often af does this is controlled by the `resync-time` variable, normally it is done every 600 seconds (10 minutes).

To manually check for new messages in a buffer's folder, use *C-x r* (`resync-buffer`). If the buffer needs to be resynchronised, then point will be set on the first new message.

18.6 Saving Files

Saving a buffer in af means writing its contents back into the folder that was visited in the buffer.

C-x C-s Save the current buffer in its visited file (`save-buffer`).

C-x s Save any or all buffers in their visited files (`save-some-buffers`).

M-x save-all-buffers
Save all modified buffers in their visited files.

M-~ Forget that the current buffer has been changed (`not-modified`).

C-x C-w Save the current buffer in a specified file (`write-file`).

When you wish to save the folder and make your changes permanent, type *C-x C-s* (`save-buffer`). After saving is finished, *C-x C-s* displays a message like this:

```
(Wrote 12 messages)
```

If the selected buffer is not modified or status modified (no changes have been made in it since the buffer was created or last saved), saving is not really done, because it would have no effect. Instead, *C-x C-s* displays a message like this in the echo area:

```
(No changes need to be written)
```

The command *C-x s* (`save-some-buffers`) offers to save any or all modified buffers. It asks you what to do with each buffer. These questions are also asked if you exit af by typing `C-x C-c`. The buffer is automatically resynchronised before being saved, and if any new messages are found the save will be abandoned.

The command `M-x save-all-buffers` saves all modified buffers. This can be dangerous if you have several buffers, since you may not want to save one of them, so be careful!

If you have changed a buffer but you do not want to save the changes, you should take some action to prevent it. Otherwise, each time you use `C-x s` or `C-x C-c`, you are liable to save this buffer by mistake. One thing you can do is type `M-~` (`not-modified`), which clears out the indication that the buffer is modified. If you do this, none of the save commands will believe that the buffer needs to be saved. ('~' is often used as a mathematical symbol for 'not'; thus `M-~` is 'not', metafied.) Alternatively, you can cancel all the changes made since the file was visited or saved, by reading the text from the file again. This is called *reverting*. See Section 18.7 [Reverting], page 77.

If you wish to mark the buffer as visiting a different folder and save it right away, use `C-x C-w` (`write-file`). It prompts for the new folder, marks the buffer as visiting that folder, and saves the buffer. If the folder already exists, af will ask for confirmation that it is ok to overwrite the file with the buffer.

18.7 Reverting a Buffer

If you have made extensive changes to a folder and then change your mind about them, you can get rid of them by reading in the previous version of the folder. To do this, use `M-x revert-buffer`, which operates on the current buffer. Since reverting a buffer unintentionally could lose a lot of work, you must confirm this command with `yes`.

Reverting marks the buffer as "not modified" until another change is made.

18.8 Inserting a Folder

Sometimes you may want to merge two folders. An easy way to do this is to use `C-x i` (`insert-file`), which inserts the contents of the specified folder into the buffer at point, leaving mark at the start of the inserted contents and point after them. The buffer will not be visiting the folder you inserted, and so will be saved to its original folder.

19 Using Multiple Buffers

The messages you are processing in af reside in an object called a *buffer*. Each time you visit a folder, a buffer is created to hold the messages in the folder.

At any time, one and only one buffer is *selected*. It is also called the *current buffer*. Often we say that a command operates on "the buffer" as if there were only one; but really this means that the command operates on the selected buffer (most commands do).

When af has multiple windows, each window has a chosen buffer which is displayed there, but at any time only one of the windows is selected and its chosen buffer is the selected buffer. Each window's mode line displays the name of the buffer that the window is displaying (see Chapter 20 [Windows], page 80).

Each buffer has a name, which can be of any length, and you can select any buffer by giving its name. Most buffers are made by visiting files, and their names are derived from the files' names. But you can also create an empty buffer with any name you want. A newly started af has a buffer named '`*scratch*`' which can be used as a temporary place to store messages. The distinction between upper and lower case matters in buffer names.

Each buffer records individually what folder it is visiting, whether it is modified, and what major mode and minor modes are in effect in it (see Section 3.3 [Major Modes], page 25).

19.1 Creating and Selecting Buffers

`C-x b` *buffer* ⟨RET⟩
: Select or create a buffer named *buffer* (`switch-to-buffer`).

`C-x 4 b` *buffer* ⟨RET⟩
: Similar, but select *buffer* in another window (`switch-to-buffer-other-window`).

To select the buffer named *bufname*, type `C-x b` *bufname* ⟨RET⟩. This runs the command `switch-to-buffer` with argument *bufname*. You can use completion on an abbreviation for the buffer name you want (see Section 7.6 [Completion], page 38). An empty argument to `C-x b` specifies the most recently selected buffer that is not displayed in any window.

Most buffers are created by visiting files, but you can also create a buffer explicitly by typing `C-x b` *bufname* ⟨RET⟩. This makes a new, empty buffer which is not visiting any file, and selects it. If you try and save a buffer created in this way, you are asked for the file name to use.

Note that `C-x C-f`, and any other command for visiting a file, can also be used to switch to an existing file-visiting buffer. See Section 18.3 [Visiting], page 73.

19.2 Listing Existing Buffers

To display a list of all the buffers that exist, type `C-x C-b`. Each line in the list shows one buffer's name, major mode and visited file. The buffers are listed in the order, most recently visited first.

'*' or '+' at the beginning of a line indicates the buffer is modified or status modified. If several buffers are modified, it may be time to save some with *C-x s* (see Section 18.6 [Saving], page 76). '%' indicates a read-only buffer. '.' marks the selected buffer. Here is an example of a buffer list:

```
MR Buffer         Size  Mode           File
-- ------         ----  ----           ----
.*  malc            12  Mail           /var/spool/mail/malc
 +  af              21  Mail           /home/malc/Mail/af
    *scratch*        0  Mail
```

19.3 Killing Buffers

If you continue an af session for a while, you may accumulate a large number of buffers. You may then find it convenient to *kill* the buffers you no longer need. On most operating systems, killing a buffer releases its space back to the operating system so that other programs can use it. Here are some commands for killing buffers:

C-x k bufname ⟨RET⟩
: Kill buffer *bufname* (`kill-buffer`).

M-x kill-some-buffers
: Offer to kill each buffer, one by one.

C-x k (`kill-buffer`) kills one buffer, whose name you specify in the minibuffer. The default, used if you type just ⟨RET⟩ in the minibuffer, is to kill the current buffer. If you kill the current buffer, another buffer is selected; one that has been selected recently but does not appear in any window now. If you ask to kill a file-visiting buffer that is modified, then you must confirm with *yes* before the buffer is killed.

The command *M-x kill-some-buffers* asks about each buffer, one by one. An answer of *y* means to kill the buffer. Killing the current buffer or a buffer containing unsaved changes selects a new buffer or asks for confirmation just like `kill-buffer`.

19.4 Inserting a Buffer

Sometimes you may want to merge two buffers. An easy way to do this is to use `M-x insert-buffer`), which inserts the contents of the specified buffer into the current buffer at point, leaving mark at the start of the inserted contents and point after them. The buffer will still be visiting the folder you originally read, and so will be saved to its original folder.

20 Multiple Windows

Af can split the screen into two or many windows. Multiple windows can display parts of different buffers, or different parts of one buffer.

20.1 Concepts of Af Windows

Each Af window displays one Af buffer at any time. A single buffer may appear in more than one window; if it does, any changes in its text are displayed in all the windows where it appears. But the windows showing the same buffer can show different parts of it, because each window has its own value of point.

At any time, one of the windows is the *selected window*; the buffer this window is displaying is the current buffer. The arrow cursor shows the location of point in this window, but is not visible in any other window.

Commands to move point affect the value of point for the selected af window only. They do not change the value of point in any other af window, even one showing the same buffer. The same is true for commands such as *C-x b* to change the selected buffer in the selected window; they do not affect other windows at all. However, there are other commands such as *C-x 4 b* that select a different window and switch buffers in it.

When multiple windows show the same buffer, they can have different regions, because they can have different values of point and mark.

Each window has its own mode line, which displays the buffer name, modification status, size in messages, and major and minor modes of the buffer that is displayed in the window. See Section 2.4 [Mode Line], page 22, for full details on the mode line.

20.2 Creating and Using Windows

C-x 2 Split the current window vertically, creating a new window (`split-window-vertically`).

C-x o Select the "other" window (`other-window`). That is *o*, not zero.

M-x previous-window RET
Select the previous window.

C-M-v Scroll the next window (`scroll-other-window`).

The command *C-x 2* (`split-window-vertically`) breaks the selected window into two windows, one above the other. Both windows start out displaying the same buffer, with the same value of point. By default the two windows each get half the height of the window that was split; a numeric argument specifies how many lines to give to the top window.

To select a different window, type *C-x o* (`other-window`). That is an *o*, for 'other', not a zero. When there are more than two windows, this command moves through all the windows from top to bottom. After the bottommost window, it goes back to the one at the top. A numeric argument means to move several steps in the cyclic order of windows. A negative argument moves around the cycle in the opposite order.

You can also select a different window with *M-x previous window*. When there are more than two windows, this command moves through all the windows from bottom to top. After the topmost window, it goes back to the one at the bottom. A numeric argument means to move several steps in the cyclic order of windows. A negative argument moves around the cycle in the opposite order.

The usual scrolling commands (see Section 5.3 [Scrolling], page 32) apply to the selected window only, but there is one command to scroll the next window. *C-M-v* (`scroll-other-window`) scrolls the window that *C-x o* would select. It takes arguments, positive and negative, like *C-v*.

20.3 Deleting and Rearranging Windows

C-x 0 Delete the selected window (`delete-window`). That is a zero.

C-x 1 Delete all windows except the selected window (`delete-other-windows`).

C-x ^ Make the selected window taller (`enlarge-window`).

M-x shrink-window
Make the selected window smaller.

To delete a window, type *C-x 0* (`delete-window`). (That is a zero.) The space occupied by the deleted window is given to an adjacent window. Once a window is deleted, its attributes are forgotten; only restoring a window configuration can bring it back. Deleting the window has no effect on the buffer it used to display; the buffer continues to exist, and you can select it in any window with *C-x b*.

C-x 1 (`delete-other-windows`) is more powerful than *C-x 0*; it deletes all the windows except the selected one; the selected window expands to use the whole screen except for the echo area.

To readjust the division of space among vertically adjacent windows, use *C-x ^* (`enlarge-window`). It makes the currently selected window get one line bigger, or as many lines as is specified with a numeric argument. With a negative argument, it makes the selected window smaller. The extra screen space given to a window comes from one of its neighbors. You cannot enlarge the window if it would make a neighbouring window less than two lines tall.

You can also readjust the size of the selected window with *M-x shrink-window*. It makes the currently selected window get one line smaller, or as many lines as is specified with a numeric argument. With a negative argument, it makes the selected window bigger. The extra screen space released by a window is given to of its neighbors. You cannot shrink the window if it would make the window less than two lines tall.

21 Customisation

This chapter talks about various topics relevant to adapting the behavior of af in minor ways. All kinds of customisation affect only the particular af session that you do them in. They are completely lost when you kill af, and have no effect on other af sessions you may run later. The only way an af session can affect anything outside of it is by writing a file; in particular, the only way to make a customisation 'permanent' is to put something in your '`.afrc`' file to do the customisation in each session. (see Section 21.2 [Startup Files], page 84).

You can easily customise af in several different ways:

Mime Configuration
: Af can be customised to handle MIME messages by modifying a set of files, which are in a standard format used by most mail readers.

Setting Variables
: Af has many *configuration variables*, which you can set to alter the behaviour of certain commands. (see Section 21.4 [Variables], page 85).

Keyboard Macros
: Keyboard macros allow you to combine af commands to create new ones. Keyboard macros are not very powerful, but they are a very simple way to add new commands to af.

Binding Keys
: You can change which commands are bound to which keys, so you can change the effect of a key press.

Afl
: You will also eventually be able to write powerful functions in af's extension language afl. This is not yet supported.

21.1 Mime Configuration

Af reads it's MIME configuration from three files; the '`mailcap`' file, which describes how to handle various message content types; the '`mime.types`' file, which lists the available content types, and the filename suffixes used for files containing that content-type; and the '`mime.charsets`' file, which lists the character sets af will support.

Af will look for each of these files in a standard set of places. First it looks for the file prefixed with a '`.`' in your home directory ('`.mailcap`', '`.mime.types`', or '`.mime.charsets`'). Then it looks in the af configuration directory. Then it looks in '`/etc`', '`/usr/etc`', and '`/usr/local/etc`'. All the entries in any files found are merged, with the first entries found taking precedence.

The location of '`mailcap`' files can be overridden by setting the `MAILCAPS` environment variable to a colon-separated list of the files to read mailcap entries from.

All these files have a similar format. Blank lines, and lines beginning with '`#`' are ignored. Each other line describes a single entry. Long lines can be continued by ending them with a '`\`'.

21.1.1 Mailcap files

The '`mailcap`' file lists how to handle a body part of a given content-type. Each entry consists, at a minimum, of a content-type and the command to display that type, separated by a semicolon. For example,

```
audio/basic; cat > /dev/audio
```

would mean that to handle the `audio/basic` content type, run the command `cat > /dev/audio`. The subtype of a content type (the part after the '`/`' may be given as '`*`', in which case the entry will match any subtype.

The command may include escape sequences, which are handled specially. The most commonly used is `%s`, which is replaced by the name of a temporary file, where the decoded text of the body part has been stored. If this escape is not used, then the decoded text is written to the standard input of the command.

In addition to this basic format, you can add more *optional fields* to the entry. These either modify the behaviour of the command, or add new facilities. Each optional field is given either in the form *name* or *name*`=`*value*, and is separated from the entry or the previous optional field by a '`;`'.

The commonly-used optional fields are

`test=`*command*
: The mailcap entry will be ignored if the result of running the command is not success (a return value of 0 on Unix systems).

`print=`*command*
: The command to print a body part. Any output from the print command will be formatted as if it were text and then sent to the system spool command. If the output of the print command is PostScript text, or any other format that must be sent to the printer as it stands, or if you need to use a special spool command to print the output, then it is safer to make the print command itself write directly to your system's spool command.

`desc=`*text*
: Gives a description of the content type.

`template=`*format*
: Allows you to specify a format for the name of the temporary file used in the `%s` escape in commands. Any %s in *format* is replaced with characters to make the name unique. This option is most often used to force the file to have a suffix which is required by the display program given in the command.

`needsterminal`
: Indicates that the display command is interactive, and will need to take control of the terminal when it is executed.

`copiousoutput`
: Indicates that the output of the command should be displayed via the pager named in the `pager` variable.

`textualnewlines`
: Indicates that the content type is essentially textual, even though it may not be a subtype of text.

`compose` Gives a program to create content of this type. Af uses this value to create a body part in Compose mode, but it is rarely defined in system mailcap files.

`composetyped`
Similar to compose, except that the composing program also returns the headers of the body part.

`edit` The program to use to edit content of this type; used by af's Compose mode, but rarely defined in system mailcap files.

For example, in the entry

```
text/enriched; richtext -e; print="richtext -e"; copiousoutput
```

the output would be displayed via a pager because the `copiousoutput` optional field has been specified.

21.1.2 The Mime.types File

The '`mime.types`' file lists the known MIME content types. In addition, it lists any filename suffixes which will indicate that a file is of that type. So the entry

```
image/jpeg      .jpg .jpeg
```

indicates that the MIME content-type `image/jpeg` exists, and that files whose names end in `.jpg` or `.jpeg` are likely to contain data of type `image/jpeg`.

21.1.3 The Mime.charsets File

The '`mime.charsets`' file lists the known MIME character sets. If a character set is not viewable, but is listed in the '`mime.charsets`' file, then af can safely assume that the character set is a superset of ASCII, and that displaying the ASCII characters in the message is safe.

21.2 The Startup File, '`~/.afrc`'

When af is started, it normally loads an afl program from the file '`.afrc`' in your home directory. We call this file your *Startup file* because it is read when you start af. You can use the command line switches '`-n`' and '`-l`' to tell af whether to load a startup file, and which one (see Chapter 4 [Entering Af], page 26).

Your site may also have a *global startup file*; this is named '`afrc.afl`', if it exists. The af distribution contains no such library; your site may create one for local customisations. If this library exists, it is loaded whenever you start af; and then your own startup file, if any, is loaded.

By far the simplest way to create a startup file is for af to write one for you. The command *`M-x write-configuration`* will prompt you for a file name (defaulting to '`~/.afrc`'), and then write an afl program into that file. When you run that program (by loading the file), it will recreate any changed variables or key bindings, or any new keymaps or named keyboard macros that were in existence when you ran *`M-x write-configuration`*.

You can use *`M-x write-configuration`* to create a startup file, and then modify that file with an editor; it is a text file. This is also a good way to become familiar with afl.

21.3 Emulation

Af can be programmed to emulate (more or less) several other mail readers. The commands to do this are stored in several afl libraries (see Chapter 22 [Afl], page 91). Note that the only way to turn an emulation off once you've loaded it is to exit af and start again.

The emulations aren't in any way intended to be true emulations of the mail readers. The intent is to let people who are already used to some other mail reader get started with af quickly, by making most of the common keys do what they expect.

To set up af to work like another mail reader there must be an emulation library for the mail reader. To load the library for (as an example) `elm`, type type ***`M-x load-library`*** ⟨RET⟩ *`elm`* ⟨RET⟩.

The available emulations are:

`elm` Reconfigures af to do a fairly close emulation of `elm`.

`mush` Not a perfect emulation by any means, but enough to get people used to `mush` started.

`pine` A close enough emulation to satisfy advanced PINE users. Doesn't hold your hand **nearly** enough for many of PINE's target users.

`ream` Not too far from the target, anyone who's used to using `ream` will probably find this emulation helpful.

`vm` Silly though it seems, the VM emulator is quite different from af's normal personality. If you've been using the VM package under Emacs to read mail, then this library may help.

21.4 Variables

Af presents two interfaces to its variables: The underlying system which is used by Afl (see Chapter 22 [Afl], page 91), or a simpler system which af uses to allow users to handle configuration variables.

A *variable* is a symbol which has a value. The symbol's name is also called the name of the variable. Variable names usually consist of words separated by hyphens. Af's configuration variables can each hold a specified kind of value. Sometimes this is a simple type such as a number, or a string, but sometimes a variable will require a more complex value, such as the formats for the screen display.

To examine the value of a single configuration variable, use *`C-h v`* (`describe-variable`), which reads a variable name using the minibuffer, with completion. It displays both the value of and the documentation for the variable. If you want to see the values of all the configuration variables, then use *`C-h C-v`* (`list-variables`), which lists all the configuration variables and their values to typeout.

The most convenient way to set a specific variable is with *`C-x a`* (`set-variable`). This reads the variable name with the minibuffer (with completion), and then reads a new value using the minibuffer a second time. For example,

```
C-x a ask-cc ⟨RET⟩ true ⟨RET⟩
```

sets `ask-cc` to true.

Whenever you use `C-x a`, the value of the variable will be checked when you enter it. You can't break af by changing variables, although you can make it behave very strangely if you set some variables to odd values.

When a variable's description says that it contains a list of items, then `C-x a` will expect the items in the list to be separated by colons. If you set the variable by afl of course, then you would use a list object as the value (see Chapter 22 [Afl], page 91). For example

```
C-x a viewable-charsets ⟨RET⟩ us-ascii:iso-8859-1 ⟨RET⟩
```

set the value of `viewable-charsets` to the list containing '`us-ascii`' and '`iso-8859-1`'.

It is usually worth using *`C-h v`* (`describe-variable`) to check the on-line help for a variable you intend to change; often the description is far more detailed than is given in this manual.

21.5 Keyboard Macros

A *keyboard macro* is a command defined by the user to stand for another sequence of keys. For example, if you discover that you are about to type *`C-t t a RET C-n`* forty times, you can speed your work by defining a keyboard macro to do *`C-t t a RET C-n`* and calling it with a repeat count of forty.

`C-x (`
: Start defining a keyboard macro (`start-kbd-macro`).

`C-x )`
: End the definition of a keyboard macro (`end-kbd-macro`).

`C-x e`
: Execute the most recent keyboard macro (`call-last-kbd-macro`).

`C-u C-x (`
: Re-execute last keyboard macro, then add more keys to its definition.

`C-x q`
: When this point is reached during macro execution, ask for confirmation (`kbd-macro-query`).

`M-x name-last-kbd-macro`
: Give a command name (for the duration of the session) to the most recently defined keyboard macro.

Keyboard macros differ from ordinary af commands in that they are written in the af command language rather than in C or afl. This makes it easier for the novice to write them, and makes them more convenient as temporary hacks. However, the af command language is not powerful enough as a programming language to be useful for writing anything intelligent or general (Neither is afl yet, but that will change).

You define a keyboard macro while executing the commands which are the definition. Put differently, as you define a keyboard macro, the definition is being executed for the first time. This way, you can see what the effects of your commands are, so that you don't have to figure them out in your head. When you are finished, the keyboard macro is defined and also has been, in effect, executed once. You can then do the whole thing over again by invoking the macro.

21.5.1 Basic Use

To start defining a keyboard macro, type the *`C-x (`* command (`start-kbd-macro`). From then on, your keys continue to be executed, but also become part of the definition of the macro. '`Def`' appears in the mode line to remind you of what is going on. When you are finished, the *C-x)* command (`end-kbd-macro`) terminates the definition (without becoming part of it!). For example,

```
C-x ( M-+ +foobar RET C-n C-x )
```

defines a macro to save the current message to the folder '`+foobar`', and then move down a line.

The macro thus defined can be invoked again with the *C-x e* command (`call-last-kbd-macro`), which may be given a repeat count as a numeric argument to execute the macro many times. *C-x)* can also be given a repeat count as an argument, in which case it repeats the macro that many times right after defining it, but defining the macro counts as the first repetition (since it is executed as you define it). Therefore, giving *C-x)* an argument of 4 executes the macro immediately 3 additional times. An argument of zero to *C-x e* or *C-x)* means repeat the macro indefinitely (until it gets an error or you type *C-g*).

If you wish to repeat an operation at regularly spaced places in the text, define a macro and include as part of the macro the commands to move to the next place you want to use it. For example, if you want to change each line, you should position point at the start of a line, and define a macro to change that line and leave point at the start of the next line. Then repeating the macro will operate on successive lines.

After you have terminated the definition of a keyboard macro, you can add to the end of its definition by typing *C-u C-x (*. This is equivalent to plain *C-x (* followed by retyping the whole definition so far. As a consequence it re-executes the macro as previously defined.

21.5.2 Naming Keyboard Macros

If you wish to save a keyboard macro for longer than until you define the next one, you must give it a name using *M-x name-last-kbd-macro*. This reads a name as an argument using the minibuffer and defines that name to execute the macro. Defining the macro in this way makes it a valid command name for calling with *M-x* or for binding a key to with `global-set-key` (see Section 21.6.1 [Keymaps], page 88). If you specify a name that has a prior definition other than another keyboard macro, an error message is printed and nothing is changed.

21.5.3 Executing Macros with Variations

Using *C-x q* (`kbd-macro-query`), you can make a macro ask you each time around whether to make a change. While defining the macro, type *C-x q* at the point where you want the query to occur. During macro definition, the *C-x q* does nothing, but when you run the macro later, *C-x q* asks you interactively whether to continue.

The valid responses are ⟨SPC⟩ to continue, and ⟨RET⟩ to skip the rest of this repetition of the macro and start right away with the next repetition. ⟨ESC⟩ means to skip the rest of this repetition and cancel further repetitions. *C-l* redraws the screen and asks you again for a character to say what to do.

21.6 Customising Key Bindings

This section describes *key bindings* which map keys to commands, and the *keymaps* which record key bindings. It also explains how to customise key bindings.

21.6.1 Keymaps

The bindings between key sequences and command functions are recorded in data structures called *keymaps*. Af has many of these, each used on particular occasions.

Recall that a *key sequence* (*key*, for short) is a sequence of key presses that have a meaning as a unit. A key sequence gets its meaning from its *binding*, which says what command it runs. The function of keymaps is to record these bindings.

The *global* keymap is the most important keymap because it is always in effect. The global keymap defines keys which are common to most or all of the major modes. Each major mode has its own keymap which overrides the global definitions of some keys.

For example, the key `C-n` move to the next line because the global keymap binds it to the command `next-line`. Commands to rebind keys, such as `M-x global-set-key`, actually work by storing the new binding in the proper place in the keymaps. See Section 21.6.4 [Rebinding], page 89.

Meta characters work differently; if the `allow-meta-bindings` variable is set to `true`, af will search for a binding for the meta character. If not, or if no binding is found, af translates each Meta character into a pair of characters starting with ⟨ESC⟩. When you type the character `M-r` in a key sequence, af will replace it with ⟨ESC⟩ `r`. A meta key comes in as a single key press, but may become two events for purposes of key bindings. The reason for this is historical; most older terminals didn't support a meta key.

A keymap records definitions for single characters. Interpreting a key sequence of multiple characters involves a chain of keymaps. The first keymap gives a definition for the first character; this definition is another keymap, which is used to look up the second character in the sequence, and so on.

You can find out what keymaps are defined in af with the command `M-x list-keymaps`. This lists the keymaps, with the prefix keys they handle, to typeout.

21.6.2 Prefix Keymaps

A prefix key such as `C-x` or ⟨ESC⟩ has its own keymap, which holds the definition for the character that immediately follows that prefix.

The definition of a prefix key is usually the keymap to use for looking up the following event. Thus, the binding of `C-x` is the keymap `control-x-prefix`, the keymap for `C-x` commands. The definitions of `C-c`, `C-t`, `C-x`, `C-h` and ⟨ESC⟩ as prefix keys appear in the global map, so these prefix keys are always available.

The `C-c` prefix has been assigned as a "user prefix". By default, nothing will be bound in the `C-c` keymap; it is reserved for your own key bindings. There is a separate `C-c` prefix keymap for each of af's major modes. See Section 21.6.3 [Local Keymaps], page 88.

21.6.3 Local Keymaps

So far we have explained the ins and outs of the global map. Major modes customise af by providing their own key bindings in *local keymaps*. For example, minibuffer mode overrides the printable characters to bind them to the command `self-insert-command`, so that the characters insert themselves into the minibuffer at point.

A local keymap can locally redefine a key as a prefix key by defining it as a prefix keymap. If the key is also defined globally as a prefix, then its local and global definitions (both keymaps) effectively combine: both of them are used to look up the character that follows the prefix key. Thus, if the mode's local keymap defines `C-x` as another keymap, and that keymap defines `C-z` as a command, this provides a local meaning for `C-x C-z`. This does not affect other sequences that start with `C-x`; if those sequences don't have their own local bindings, their global bindings remain in effect.

Another way to think of this is that af handles a multi-event key sequence by looking in several keymaps, one by one, for a binding of the whole key sequence. First it checks the minor mode keymaps for minor modes that are enabled, then it checks the major mode's keymap, and then it checks the global keymap. This is not precisely how key lookup works, but it's good enough for understanding ordinary circumstances.

Designing af keymaps is made more complicated because several commands will only work in one or two of the major modes. You should be careful using global bindings; local bindings will usually achieve what you want, and are less confusing to work with.

21.6.4 Changing Key Bindings Interactively

The way to redefine an af key is to change its entry in a keymap. You can change the global keymap, in which case the change is effective in all major modes (except those that have their own overriding local definitions for the same key). Or you can change the current buffer's local map, which affects all buffers using the same major mode.

`M-x global-set-key` ⟨RET⟩ *key cmd* ⟨RET⟩
: Define *key* globally to run *cmd*.

`M-x global-unset-key` ⟨RET⟩ *key*
: Make *key* undefined in the global map.

`M-x local-set-key` ⟨RET⟩ *key cmd* ⟨RET⟩
: Define *key* locally (in the major mode now in effect) to run *cmd*.

`M-x local-unset-key` ⟨RET⟩ *key*
: Make *key* undefined locally (in the major mode now in effect).

`M-x typeout-set-key` ⟨RET⟩ *key cmd* ⟨RET⟩
: Define *key* to run *cmd* in the typeout keymap.

`M-x typeout-unset-key` ⟨RET⟩ *key*
: Make *key* undefined in the typeout map.

`M-x minibuffer-set-key` ⟨RET⟩ *key cmd* ⟨RET⟩
: Define *key* to run *cmd* in the minibuffer keymap.

`M-x minibuffer-unset-key` ⟨RET⟩ *key*
: Make *key* undefined in the minibuffer map.

`M-x make-keymap` ⟨RET⟩ *map* ⟨RET⟩
: Create a new keymap *map*, which you can then use to add your own prefix keys.

For example, suppose you like to execute commands in a subshell, instead of suspending af and executing commands in your login shell. Normally, *C-z* is bound to the function `suspend-af`, but you can change *C-z* to invoke a subshell by binding it to `shell` as follows:

```
M-x global-set-key ⟨RET⟩ C-z shell ⟨RET⟩
```

`global-set-key` reads the command name after the key. After you press the key, a message like this appears so that you can confirm that you are binding the key you want:

```
Set key C-z to command:
```

You can rebind a key that contains more than one character in the same way. Af keeps reading the key to rebind until it is a complete key (that is, not a prefix key). Thus, if you type *C-f* for *key*, that's the end; the minibuffer is entered immediately to read *cmd*. But if you type *C-x*, another character is read; if that is 4, another character is read, and so on. For example,

```
M-x global-set-key ⟨RET⟩ C-x 4 $ scroll-other-window ⟨RET⟩
```

redefines *C-x 4 $* to run the command scroll-other-window.

The commands `typeout-set-key` and `minibuffer-set-key` are a convenience; they bind keys in the typeout or minibuffer keymaps rather than the global or local maps.

You can remove the global definition of a key with `global-unset-key`. This makes the key *undefined*; if you type it, af will just beep. Similarly, `local-unset-key` makes a key undefined in the current major mode keymap, which makes the global definition (or lack of one) come back into effect in that major mode.

If you have redefined (or undefined) a key and you subsequently wish to retract the change, undefining the key will not do the job—you need to redefine the key with its standard definition. The documentation of keys in this manual also lists their command names.

If you want to define a new prefix key, you will need to create a new keymap to handle the prefix key. To do this, use the command *M-x make-keymap*, which will prompt you for the name of the keymap to create. If you specify a name that already has a definition an error message is printed and nothing is changed.

There are two variables which can be important when you are binding keys. The variable `meta-prefix-char` names the character which is used as a prefix when you type a meta character at the keyboard. If you change the binding of ⟨ESC⟩, then you should change the value of `meta-prefix-char` to reflect this. Similarly, the variable `quit-char` names the character which is used to quit from confirm prompts, and similar; so it should probably reflect the usual binding of `keyboard-quit` in the keymaps.

22 Afl

Afl is the af *extension language*, a simple programming language which allows you to write files containing af commands. Afl is similar to the Lisp programming language, so Lisp programmers should find afl easy to learn. Afl should be simple enough that it is easy for other programmers to learn.

Afl is designed for programmers to use, although once a programmer has written an afl program anyone should be able to use that program.

22.1 Basic Afl Concepts

Afl programs are made up of one or more afl *objects*; pieces of data used and manipulated by afl programs. For our purposes, a *type* or *data type* is a set of possible objects.

When you run an afl program, the *afl reader* is called to read the textual representations of the afl objects in the program, and convert them to actual afl objects. If you try to display an object, the *afl printer* is called to generate a *printed representation* of the object.

The *evaluation* of expressions in afl is performed by the *afl interpreter*; a program that receives an afl object as input and computes its *value as an expression*. How it does this depends on the data type of the object. An afl object that is intended for evaluation is called an *expression* or a *form*.

The most useful type of afl object to evaluate is lists. When you evaluate an afl list, the head of the list is evaluated as a function, and then the remaining items in the list are passed as arguments to the function. Arguments are always evaluated in left-to-right order, although there are a few functions, known as *special forms*, which may not evaluate all of their arguments. These functions' handling of arguments will be explained in the function's description.

22.2 Afl Data Types

Each data type in afl has a *read syntax* and a *printed representation*. The read syntax is the way the data type should be written in an afl program. The printed representation is the way that the data type will be displayed by the afl printer. These may be the same or different.

22.2.1 Comments

A *comment* is text that is written in a program only for the sake of humans that read the program, and that has no effect on the meaning of the program. In afl, a semicolon ('`;`') starts a comment if it is not within a string or character constant. The comment continues to the end of line. The afl reader discards comments; they do not become part of the afl objects which represent the program.

22.2.2 Special Constants

In afl, the symbol `nil` is overloaded with three meanings: it is a symbol with the name '`nil`'; it is the logical truth value *false*; and it is the empty list—the list of zero elements. When used as a variable, `nil` always has the value `nil`.

As far as the afl reader is concerned, '`()`' and '`nil`' are identical: they stand for the same object, the symbol `nil`. The different ways of writing the symbol are intended entirely for human readers. After the afl reader has read either '`()`' or '`nil`', there is no way to determine which representation was actually written by the programmer.

In contexts where a truth value is expected, any non-`nil` value is considered to be *true*. However, `t` is the preferred way to represent the truth value *true*. When you need to choose a value which represents *true*, and there is no other basis for choosing, use `t`. The symbol `t` always has value `t`.

For convenience, the constant `a` is defined to be the value '`ask`' or '`accept`' in af's tristate configuration variables. This is the preferred way to set, for example, the variable `copy-on-reply` to '`ask`'.

In afl, `nil`, `t` and `a`, are special symbols that always evaluate to themselves. This is so that you do not need to quote them to use them as constants in a program. An attempt to change their values results in an error.

22.2.3 Lists

A list of objects of any data type (including lists). A list is read or printed as a set of objects enclosed in parentheses; for example '`(a b c)`', or '`(a b (c d e) f)`'.

Lists are evaluated by evaluating the symbol at their head as a function. The remaining objects are evaluated (unless the function is a special form), and passed as arguments to the function.

22.2.4 Numbers

Afl only supports integer numbers. A number is read or printed as an optional sign, followed by one or more digits; for example '`42`', or '`-999`'.

There is a system-defined limit to the maximum value of a number, usually 32 bits. Larger values will be truncated to fit within these bounds, which will leave them with unexpected values.

Numbers always evaluate to themselves.

22.2.5 Characters

In afl, characters are interchangeable with numbers, and are always printed as a number. They can be read either as a number, or as a question mark followed by the character; for example '`?a`'. If the character is not a normal alphanumeric one, then it is often necessary to prefix it with a backslash '\' to make sure it is treated as a character. If the character is '\' then you need a second '\' to quote it; for example. '`?\\`'.

You can write the characters Control-g, backspace, tab, newline, vertical tab, formfeed, return, and escape as '`?\a`', '`?\b`', '`?\t`', '`?\n`', '`?\v`', '`?\f`', '`?\r`', and '`?\e`', respectively.

Rather than quoting a control character with '\', another read syntax may be used. If a character is a backslash followed by a caret and another non-control character, then it represents the control character. For example, '`?\^I`' is read syntax for the character *`C-i`*. You may also use '`C-`' rather than '`^`', so '`\C-i`' is also read syntax for *`C-i`*.

A *meta character* is one that has the top bit set, and can be typed with a meta key on some keyboards. Read syntax for meta characters is '`?\M-`' followed by the character. This may be combined with control character read sequence, hence '`?\C-\M-i`' is read syntax for *C-M-i*.

Finally, the most general read syntax for characters is a question mark followed by a backslash and the numeric value of the character in octal (up to three digits are allowed). So '`?\001`' is read syntax for the character `C-a`.

An invalid control or meta character sequence (such as '`?\C-\^a`' or '`?\C-1`') will be flagged as an error by the afl reader.

As with numbers, there is a system-defined limit to the maximum value of a character, usually 8 bits. Larger values will be truncated to fit within these bounds, which will leave them with unexpected values.

Characters always evaluate to themselves.

22.2.6 Strings

The read syntax and printed representation of afl strings is the string enclosed in double quotes; for example '`"A string"`'. All the special ways of writing characters can be used within strings. In addition, a backslash followed by a newline is ignored within a string, so strings can be broken to fit on the screen better; for example:

```
"This string is broken here \
by a \\\\n sequence"
```

would be read as '`This string is broken here by a \\n sequence`'.

Strings always evaluate to themselves.

22.2.7 Symbols

A symbol name can contain any characters you require. Any use of the characters '`'"();?.,\ \t\n`' should be quoted with a '`\`' character. If the symbol name looks like a number, then it can be prefixed with a '`\`' to force it to be treated as a symbol. Note that unlike its use in strings, the '`\`' simply quotes the next character in the symbol name.

Evaluating a symbol works differently according to the context in which the symbol is being evaluated. An afl symbol can store a variable or a function, or both. The afl evaluator will usually look up the value of the symbol as a variable, but will use the value as a function if the symbol is at the head of a list.

22.3 Afl Primitive Functions

Afl has several *primitive functions*, which are predefined in af, and can be called from your afl programs. In addition, an afl program can call any af command as a function; af commands are simply functions that have made provision for interactive use.

The appearance of the keyword `&optional` in the parameter list of a function indicates that the arguments for subsequent parameters may be omitted (omitted parameters default to `nil`). Do not write `&optional` when you call the function.

The keyword `&rest` (which will always be followed by a single parameter) indicates that any number of arguments can follow. The value of the single following parameter will be a list of all these arguments. Do not write `&rest` when you call the function.

22.3.1 Quoting Forms

`(quote` *form*`)`
: Quote *form*, protecting it from evaluation.

Sometimes, you may not want a form to be evaluated, for example the name of a symbol to assign a value to with `set`. You can prevent a form from being evaluated by using the special form `quote`. So `(quote list-functions)` expands to the symbol `list-functions`.

Because `quote` is used so often, there is a special notation for it. Writing `'`*form* is equivalent to writing `(quote` *form*`)`. So `'list-functions` is equivalent to the example above.

22.3.2 Controlling Program Flow

cindex comparing afl objects

`(if` *condition then* `&optional` *else*`)`
: Execute *then* if *condition* evaluates to non-nil, *else* otherwise.

`(progn &rest` *forms*`)`
: Evaluates *forms* from left to right, returning the value of the last form evaluated.

`(equal` *object1 object2*`)`
: Returns `t` if *object1* and *object2* have the same afl type and contents, `nil` otherwise.

`(and &rest` *conditions*`)`
: Evaluates *conditions* until one returns `nil`. Returns the value of the last condition evaluated.

`(or &rest` *conditions*`)`
: Evaluates *conditions* until one returns non-nil. Returns the value of the last condition evaluated.

`(not` *condition*`)`
: Returns `t` if *condition* evaluates to `nil`, `nil` otherwise.

To evaluate code only if a condition is true, use `(if` *condition true false*`)`. This evaluates *condition*, and then evaluates *true* if *condition* returned non-nil, or *false* (if specified) otherwise.

You can use `progn` to group a set of forms, for example to use in an `if`. `progn` evaluates its arguments from left-to-right, and returns the value returned by the last argument.

You can compare the value of two afl objects with `equal`. This will return `t` if the two objects have the same afl type and contents, or `nil` otherwise.

To combine one or more logical expressions, you can use `and`, `or`, and `not`. The special form `and` evaluates its arguments from left to right until one of them returns `nil`, and then returns `nil`; the remaining arguments are never evaluated. If no argument returns `nil`, then the value returned by the last argument evaluated is returned.

The special form `or` is more-or-less the opposite of `and`; it evaluates its arguments from left to right until one of them returns non-nil, and then returns the value the argument returned; the remaining forms are never evaluated. If no argument evaluates to non-nil, then `or` returns `nil`.

To reverse a test use `not`. This returns `t` if the argument evaluates to `nil`, or `nil` otherwise.

22.3.3 Configuring Af From Afl

`(set` *symbol value*`)`
: Set *symbol* to contain *value*.

`(setq` *symbol value*`)`
: Equivalent to `(set '`*symbol value*`)`.

`(define-key` *keymap* `"`*key*`"` *value*`)`
: Bind key *key* in keymap *keymap* to *value*.

`(define-kbd-macro` *macro* `"`*keys*`")`
: Define a keyboard macro *macro* to execute *keys*.

To set an af configuration variable, use the function `set`. This sets the variable to the value supplied. The value will be checked to make sure that it is a valid value for the variable.

Since `set` is a function you will need to quote the symbol to be set. For convenience, the special form `setq` is equivalent to `set`, but does not evaluate the object which is to be set. So `(setq ask-cc t)` is equivalent to `(set 'ask-cc t)`.

To bind a key in a keymap use `(define-key` *keymap* `"`*key*`"` *object*`)`. This binds *key* in keymap *keymap* to *object*, which must be a command, a keymap, a keyboard macro, or `nil`. To find the name of the keymap you want to bind the key in use the `list-keymaps` command. Since `define-key` is a function you will need to quote *keymap* and *object*.

To define a new keyboard macro, or redefine an existing one, use the function `(define-kbd-macro` *macro* `"`*keys*`")`. This defines *macro* to execute *keys*. Since `define-kbd-macro` is a function you will need to quote *macro*.

22.3.4 Interacting With the Environment

`(message &rest` *forms*`)`
: Display the print format of *forms* in the echo area.

`(error &rest` *forms*`)`
: Beep and display *forms* in the echo area.

`(getenv "`*name*`")`
: Return the value of the environment variable *name*, or `nil` if name is not set in the environment.

The functions `message` and `error` display messages or error messages in the echo area. While `message` returns the forms that it printed, `error` returns an error condition, which will halt the] execution of the afl program.

You can extract values from the environment by using `(getenv "`*variable*`")`, which looks up and returns the definition of *variable* in the environment. This can be very useful for terminal-dependent customisation of af.

22.4 Running Afl Programs

M-x load-file ⟨RET⟩ *file* ⟨RET⟩
: Load *file* as an afl program.

M-x load-library ⟨RET⟩ *lib* ⟨RET⟩
: Search for and load a file '*lib*' or '*lib*`.afl`'.

Once you have created a file containing af commands (known as an *afl program* or *afl library*, you can *load* it. When you load an afl program, the file is opened, and its contents are passed to the afl reader. Once all the file has been read and evaluated, then the file is closed. If any form in the file is invalid, or produces an error, then the afl reader will abort, and the file will be closed.

The simplest way to load an afl program is to use the the command `load-file`, which loads a named file. You must specify the full name of the file.

The `load-library` command will search for an afl program in a set of directories defined by the `load-path` variable. It checks for the file name with '`.afl`' appended, as well as the filename alone.

If you want to modify where af looks for library files, there are two ways to do so. You can set the environment variable `AFLOADPATH` to a colon-separated list of directories to search; or you can set the configuration variable `load-path` to the same value. For example:

```
(setq load-path '("/usr/lib/af" "/usr/local/lib/af"))
```

would make af look for afl libraries in the directories '`/usr/lib/af`' and '`/usr/local/lib/af`'.

22.5 Examples of Afl Code

Here are a few examples of afl code, with brief explanations.

Load the library '`ansi.afl`' if you are using an ANSI terminal (under one of several common names):

```
(if (or (equal (getenv "TERM") "ansi")
(equal (getenv "TERM") "xterm")
(equal (getenv "TERM") "vt100"))
    (load-library "ansi"))
```

Set the `viewable-charsets` variable to allow displaying the `iso-8859-1` character set if running on an xterm:

```
(if (equal (getenv "TERM") "xterm")
    (setq viewable-charsets '("iso-8859-1" "us-ascii")))
```

Complain if the user isn't running on an xterm:

```
(if (not (equal (getenv "TERM") "xterm"))
    (error "Must be running on an xterm"))
```

Define a keyboard macro to move to the next message from typeout:

```
(define-kbd-macro 'typeout-next-message "\C-g\C-n\r")
```

Bind the above macro to the key *M-n* in the typeout maps:

```
(define-key 'typeout-prefix-command "n" 'typeout-next-message)
```

Glossary

Afl
: Af's extension language; a small programming language which allows you to write commands for af.

Attachment
: A body part which has been marked as an attachment with the '`Content-Disposition`' header. Af requires confirmation before it will display an attachment.

Bind
: To bind a key sequence means to give it a binding (q.v.). See Section 21.6 [Key Bindings], page 87.

Binding
: A key sequence gets its meaning in af by having a binding, which is a command (q.v.) that is run when the user types that sequence. See Section 3.2 [Commands], page 24. Customisation often involves rebinding a character to a different command. The bindings of all key sequences are recorded in the keymaps (q.v.). See Section 21.6.1 [Keymaps], page 88.

Blind Copy
: A copy of an outgoing message, sent so someone in such a way that the other people receiving the message won't know that the person received a copy. See Chapter 8 [Sending Mail], page 41.

Body
: The body of a mail message is the actual text of the message, excluding the headers. See Section 1.3 [Message Body], page 16.

Bouncing
: Bouncing a message, means to re-send the message "as is" to someone, while making it appear that the new message is still from the person who sent the original message. See Section 8.1 [Sending Commands], page 41.

Buffer
: The buffer is the basic mail-reading unit; one buffer corresponds to one mail folder being read. You can have several buffers, but at any time you are processing only one, the 'selected' buffer, though several can be visible when you are using multiple windows (q.v.). Most buffers are visiting (q.v.) some folder. See Chapter 19 [Buffers], page 78.

C-
: *C-* in the name of a character is an abbreviation for Control. See Chapter 3 [User Input], page 23.

C-M-
: *C-M-* in the name of a character is an abbreviation for Control-Meta. See Chapter 3 [User Input], page 23.

Carbon Copies
: A copy of an outgoing message to someone, with an inference that the person should not regard the message as being addressed to them.

Command
: A command is an af function which can be bound to a key. When you type a key sequence (q.v.), its binding (q.v.) is looked up in the relevant keymaps (q.v.) to find the command to run. Alternatively, any command can be invoked by name using *M-x* See Section 3.2 [Commands], page 24.

Complete Key
: A complete key is a key sequence which fully specifies one action to be performed by af. For example, `RET` and `C-n` and `M-m` are complete keys. Complete keys derive their meanings from being bound (q.v.) to commands (q.v.). Thus, `RET` is by default bound to a command to open and read the current message; `M-m` is by default bound to a command to begin composing a mail message. See Section 3.1 [Keys], page 23.

Completion
: Completion is what af does when it automatically fills out an abbreviation for a name into the entire name. Completion is done for minibuffer (q.v.) arguments when the set of possible valid inputs is known; for example, on command names, buffer names, and file names. Completion occurs when ⟨TAB⟩, ⟨SPC⟩ or ⟨RET⟩ is typed. See Section 7.6 [Completion], page 38.

Control Character
: ASCII characters with octal codes 0 through 037, and also code 0177, do not have graphic images assigned to them. These are the control characters. To type a control character, hold down the ⟨CTRL⟩ key and type the corresponding non-Control character. ⟨RET⟩, ⟨TAB⟩, ⟨ESC⟩, ⟨LFD⟩ and ⟨DEL⟩ are all control characters. See Chapter 3 [User Input], page 23.

Copyleft
: A copyleft is a notice giving the public legal permission to redistribute a program or other work of art. Copylefts are used by left-wing programmers to give people equal rights, just as copyrights are used by right-wing programmers to gain power over other people.

  The particular form of copyleft used by af is that used by the Free Software Foundation's project GNU, and is called the GNU General Public License. See [Copying], page 3.

Current Buffer
: The current buffer in af is the buffer on which most commands operate. You can select any af mail buffer as the current one. See Chapter 19 [Buffers], page 78.

Current Directory
: When you specify a file name that does not start with a '`/`' or a special prefix character such as '`~`', it is interpreted relative to the current directory. See Section 18.1 [File Names], page 72.

Current Message
: The message point is before. Many af commands operate on the current message. See Section 2.1 [Point], page 20.

Cursor
: The cursor is the pointer on the screen which indicates the position called point (q.v.) at which insertion and deletion takes place. It may be represented by an arrow pointer or by the terminal's cursor. The cursor is on the message (or character in the minibuffer) that follows point. Often people speak of 'the cursor' when, strictly speaking, they mean 'point'. See Chapter 5 [Basics], page 31.

Customisation
: Customisation is making minor changes in the way af works. It is often done by setting variables (see Section 21.4 [Variables], page 85) or by rebinding key sequences (see Section 21.6 [Key Bindings], page 87).

Default Argument
: The default for an argument is the value that will be assumed if you do not specify one. When the minibuffer is used to read an argument, the default argument is used if you just type ⟨RET⟩. See Chapter 7 [Minibuffer], page 36.

Default Response
: A default response for an argument is a value that is initially inserted into the minibuffer for you to edit. See Chapter 7 [Minibuffer], page 36.

⟨DEL⟩
: ⟨DEL⟩ is a character that runs the command to delete one character of text. See Chapter 5 [Basics], page 31.

Deletion of Messages
: Deleting a message means flagging it to be eliminated from your mail file. Until you save (q.v.) the buffer, you can still undelete the messages you have deleted. See Section 5.4 [Deleting Messages], page 32.

Deletion of Windows
: Deleting a window means eliminating it from the screen. Other windows expand to use up the space. The deleted window can never come back, but no actual text is thereby lost. See Chapter 20 [Windows], page 80.

Directory
: File directories are named collections in the file system, within which you can place individual files or subdirectories. See Section 18.1 [File Names], page 72.

E-Mail
: See 'Electronic Mail'.

Electronic Mail
: Electronic mail means messages sent from one user to another through the computer system, to be read at the recipient's convenience. Af has commands for composing and sending mail, and for reading and editing the mail you have received.

Echo Area
: The echo area is the bottom line of the screen, used for echoing the arguments to commands, for asking questions, and printing brief messages (including error messages). See Section 2.2 [Echo Area], page 21.

Echoing
: Echoing is acknowledging the receipt of commands by displaying them (in the echo area). Af never echoes single-character key sequences; longer key sequences echo only if you pause while typing them. See Section 2.2 [Echo Area], page 21.

Error
: An error occurs when an af command cannot execute in the current circumstances. When an error occurs, execution of the command will normally stop, and af reports the error by printing an error message (q.v.). Then af is ready to read another command.

Error Message
: An error message is a single line of output displayed by af when the user asks for something impossible to do (such as, killing a message when point is at the end of the buffer). They appear in the echo area, accompanied by a beep.

Envelope
: The information used to route a mail message to the correct people. The information is usually stored in the message's headers. See Chapter 1 [E-Mail], page 14.

⟨ESC⟩
: ⟨ESC⟩ is a character used as a prefix for typing Meta characters on keyboards lacking a ⟨META⟩ key. Unlike the ⟨META⟩ key (which, like the ⟨SHIFT⟩ key, is held down while another character is typed), the ⟨ESC⟩ key is pressed once and applies to the next character typed.

File Name
: A file name is a name that refers to a file. File names may be relative or absolute; the meaning of a relative file name depends on the current directory, but an absolute file name refers to the same file regardless of which directory is current. On Unix systems, an absolute file name starts with a slash (the root directory) or with '`~/`' or '`~`*user*`/`' (a home directory).

File Name Component
: A file name component names a file directly within a particular directory. On Unix systems, a file name is a sequence of file name components, separated by slashes. For example, '`foo/bar`' is a file name containing two components, '`foo`' and '`bar`'; it refers to the file named '`bar`' in the directory named '`foo`' in the current directory.

Folder
: A file which contains mail messages. See Chapter 18 [Folders], page 72.

Folder Directory
: The directory where folders are stored. See Chapter 18 [Folders], page 72.

Forwarding
: Sending an annotated copy of a mail message to someone other than the original sender of the message. See Section 8.1 [Sending Commands], page 41.

Global Keymap
: The global keymap (q.v.) contains key bindings that are in effect except when overridden by local key bindings in a major mode's local keymap (q.v.). See Section 21.6.1 [Keymaps], page 88.

Header Line
: A single line on the screen which shows details of a mail message, and represents that message in the buffer. See Chapter 2 [Screen], page 20.

Headers
: The part of a message which contains information about the message itself, rather than the content of the message. The headers appear at the start of the message, and are separated from the body (q.v.) by a blank line.

Incoming Mailbox
: The folder where the system delivers you incoming mail. See Chapter 18 [Folders], page 72.

Input Event
: An input event represents, within af, one action taken by the user on the terminal. The only Input Events currently accepted by af are those for typing characters. See Chapter 3 [User Input], page 23.

Insertion
: Insertion means copying messages or text into a buffer, either from the keyboard or with some other af command.

Keyboard Macro
: Keyboard macros are a way of defining new af commands from sequences of existing ones. See Section 21.5 [Keyboard Macros], page 86.

Key Sequence
: A key sequence (key, for short) is a sequence of input events (q.v.) that are meaningful as a single unit. If the key sequence is enough to specify one action, it is a complete key (q.v.); if it is not enough, it is a prefix key (q.v.). See Section 3.1 [Keys], page 23.

Keymap
: The keymap is the data structure that records the bindings (q.v.) of key sequences to the commands that they run. For example, the global keymap binds the character *C-n* to the command function `next-line`. See Section 21.6.1 [Keymaps], page 88.

Kill Ring
: The kill ring is where all messages you have killed recently are saved. You can reinsert any of the killed messages still in the ring; this is called yanking (q.v.). See Section 14.2 [Yanking], page 62.

Killing
: Killing means erasing messages (or text in the minibuffer), and saving it on the kill ring so it can be yanked (q.v.) later. Some other systems call this "cutting". See Section 14.1 [Killing], page 62.

Local Keymap
: A local keymap is used in a particular major mode; the key bindings (q.v.) in the current local keymap override global bindings of the same key sequences. See Section 21.6.1 [Keymaps], page 88.

M-
: *M-* in the name of a character is an abbreviation for ⟨META⟩, one of the modifier keys that can accompany any character. See Chapter 3 [User Input], page 23.

M-C-
: *M-C-* in the name of a character is an abbreviation for Control-Meta; it means the same thing as *C-M-*. If your terminal lacks a real ⟨META⟩ key, you type a Control-Meta character by typing ⟨ESC⟩ and then typing the corresponding Control character. See Chapter 3 [User Input], page 23.

M-x
: *M-x* is the key sequence which is used to call an af command by name. This is how you run commands that are not bound to key sequences. See Chapter 9 [M-x], page 50.

Mail
: See 'electronic mail'.

Mail Aliases
: A short mnemonic name which stands for one or more mail addresses. See Section 8.7 [Aliases], page 48.

Mailbox
: See 'incoming mailbox'.

Major Mode
: The Af major modes are a mutually exclusive set of options, each of which handles different functions within af. See Section 3.3 [Major Modes], page 25.

Mark
: The mark points to a position in the text. It specifies one end of the region (q.v.), point being the other end. Many commands operate on all the text from point to the mark. Each buffer has its own mark. See Chapter 12 [Mark and Region], page 56.

Message
: A single electronic mail message. See Chapter 1 [E-Mail], page 14.

Meta
: Meta is the name of a modifier bit which a command character may have. It is present in a character if the character is typed with the ⟨META⟩ key held down. Such characters are given names that start with `Meta-` (usually written `M-` for short). For example, `M-<` is typed by holding down ⟨META⟩ and at the same time typing `<` (which itself is done, on most terminals, by holding down ⟨SHIFT⟩ and typing `,`). See Chapter 3 [User Input], page 23.

Meta Character
: A Meta character is one whose character code includes the Meta bit.

MIME
: A standard for transferring messages that contain information other then plain ASCII text. See Section 1.3 [Message Body], page 16.

Minibuffer
: The minibuffer is the window that appears when necessary inside the echo area (q.v.), used for reading arguments to commands. See Chapter 7 [Minibuffer], page 36.

Minibuffer History
: The minibuffer history records the text you have specified in the past for minibuffer arguments, so you can conveniently use the same text again. See Section 7.7 [Minibuffer History], page 40.

Minor Mode
: A minor mode is an optional feature of af which can be switched on or off independently of all other features. Some minor modes have a command to turn it on or off. See Section 3.4 [Minor Modes], page 25.

Mode Line
: The mode line is the line at the bottom of each window (q.v.), giving status information on the buffer displayed in that window. See Section 2.4 [Mode Line], page 22.

Modified Buffer
: A buffer (q.v.) is modified if its contents have been changed since the last time the buffer was saved (or since when it was created, if it has never been saved). See Section 18.6 [Saving], page 76.

Moving Messages
: Moving messages means erasing them from one place and inserting them in another. The usual way to move messages is by killing (q.v.) and then yanking (q.v.). See Section 14.1 [Killing], page 62.

Narrowing
: Narrowing means creating a restriction (q.v.) that limits the display of message in the current buffer to only some of the messages in the buffer. Messages which are not displayed are inaccessible to the user until the buffer is widened again, but it is still there, and saving the folder saves it all. See Chapter 17 [Narrowing], page 71.

Numeric Argument
: A numeric argument is a number, specified before a command, to change the effect of the command. Often the numeric argument serves as a repeat count. See Section 5.8 [Arguments], page 33.

Point
: Point is the place in the buffer at which most commands take effect. Point is considered to be between two messages, not at one message. A cursor (q.v.) indicates the location of point. See Chapter 5 [Basics], page 31.

Prefix Argument
: See 'numeric argument'.

Prefix Key
: A prefix key is a key sequence (q.v.) whose sole function is to introduce a set of longer key sequences. `C-x` is an example of prefix key; any two-character sequence starting with `C-x` is therefore a legitimate key sequence. See Section 3.1 [Keys], page 23.

Prompt
: A prompt is text printed to ask the user for input. Displaying a prompt is called prompting. Af prompts always appear in the echo area (q.v.). One kind of prompting happens when the minibuffer is used to read an argument (see Chapter 7 [Minibuffer], page 36); the echoing which happens when you pause in the middle of typing a multi-character key sequence is also a kind of prompting (see Section 2.2 [Echo Area], page 21).

Quitting
: Quitting means cancelling a partially typed command or a running command, using `C-g`. See Section 5.6 [Quitting], page 33.

Quoting
: Quoting means depriving a character of its usual special significance. In af this is usually done with `C-q`. What constitutes special significance depends on the context and on convention. For example, in the minibuffer, an "ordinary" character as an af command inserts itself; so in this context, a special character is any character that does not normally insert itself (such as ⟨DEL⟩, for example), and quoting it makes it insert itself as if it were not special. Not all contexts allow quoting. See Chapter 7 [Minibuffer], page 36.

Quoting a Message
: When replying to a message, quoting is copying the text of the original message into the text of the new message, prefixed with some characters to mark it as the text to which you are replying. See Chapter 8 [Sending Mail], page 41.

Read-Only Buffer
: A read-only buffer is one whose contents you are not allowed to change. Normally af makes buffers read-only when the folder they are visiting is write-protected. See Chapter 19 [Buffers], page 78.

Redisplay
: Redisplay is the process of correcting the image on the screen to correspond to changes that have been made in the text being edited. See Chapter 2 [Screen], page 20.

Regex
: See 'regular expression'.

Region
: The region is the text between point (q.v.) and the mark (q.v.). Many commands operate on the region. See Chapter 12 [Mark and Region], page 56.

Regular Expression
: A regular expression is a pattern that can match various text strings; for example, '`l[0-9]+`' matches '`l`' followed by one or more digits. See Section 15.4 [Regex Syntax], page 66.

Repeat Count
: See 'numeric argument'.

⟨RET⟩
: ⟨RET⟩ is a character that is used to terminate most arguments read in the minibuffer (q.v.). See Chapter 3 [User Input], page 23.

Saving
: Saving a buffer means copying its contents back into the file that was visited (q.v.) in that buffer. This is the way in which folders actually get changed by af. See Section 18.6 [Saving], page 76.

Scrolling
: Scrolling means shifting the messages (or text in typeout) in the window so as to see a different part of the buffer. See Section 5.3 [Scrolling], page 32.

Searching
: Searching means moving point to the next match for a specified regular expression. See Chapter 15 [Searching], page 65.

Search Path
: A search path is a list of directory names, to be used for searching for files for certain purposes. For example, the variable `load-path` holds a search path for finding afl library files. See Chapter 22 [Afl], page 91.

Selecting
: Selecting a buffer means making it the current (q.v.) buffer. See Chapter 19 [Buffers], page 78.

Self-Documentation
: Self-documentation is the feature of af which can tell you what any command does, or give you a list of all commands related to a topic you specify. You ask for self-documentation with the *C-h*. See Chapter 10 [Help], page 51.

Signature File
: A file which is automatically appended to the text of any messages you send. See Section 1.4 [Signatures], page 19.

System Tags
: Tags (q.v.) which are automatically set by af to show status information about a message. See Chapter 13 [Tags], page 58.

Tags
: Tags are markers which are set on a message either by af (See 'system tags') or a user. Af allows you to perform commands on messages which meet specified criteria for tags. See Chapter 13 [Tags], page 58.

Tagset
: The set of messages which match a tag expression. See Section 13.1.4 [Tag Expressions], page 59.

Top Level
: Top level is the normal state of af, in which you are looking at the contents of the folder you have visited. You are at top level whenever you are not in typeout (q.v.) or the minibuffer (q.v.), and not in the middle of a command. You can get back to top level by quitting (q.v.). See Section 5.6 [Quitting], page 33.

Typeout
: Typeout is the mode used for paging text or messages inside af. The typeout window pops up when it is needed, and disappears when typeout is complete. See Chapter 6 [Typeout], page 35.

Variable
: A variable is an object that can store an arbitrary value. Af has some variables (called "configuration variables) that exist just so that you can set their values to control the behaviour of af. See Section 21.4 [Variables], page 85.

Visiting
: Visiting a folder means reading its contents into a buffer (q.v.) where they can be processed. See Section 18.3 [Visiting], page 73.

Whitespace
: Whitespace is any run of consecutive formatting characters (space, tab, newline, and backspace).

Widening
: Widening is removing any restriction (q.v.) on the current buffer; it is the opposite of narrowing (q.v.). See Chapter 17 [Narrowing], page 71.

Window
: Af divides the screen into one or more windows, each of which can display the contents of one buffer (q.v.) at any time. See Chapter 2 [Screen], page 20, for basic information on how af uses the screen. See Chapter 20 [Windows], page 80, for commands to control the use of windows.

Yanking
: Yanking means reinserting messages or text previously killed. It can be used to undo a mistaken kill, or for copying or moving messages. Some other systems call this "pasting". See Section 14.2 [Yanking], page 62.

Key (Character) Index

Command and Function Index

L

M

N

O

P

Q

R

S

T

U

V

W

Y

Variable Index

Concept Index

P

Q

R

S

T

U

V

W

Y

Short Contents

Table of Contents