PMDF popstore & MessageStore Manager's Guide

Order Number: N-5304-66-NN-N

September 2015

This manual describes the usage of V6.7 of the PMDF popstore and PMDF MessageStore. Documentation on the popstore API is also provided in this manual.

Revision/Update Information:	This manual supersedes the V6.6 *PMDF popstore & MessageStore Manager's Guide*
Software Version:	PMDF V6.7
Operating System and Version:	Solaris SPARC V2.6, V8 or later; (SunOS V5.6, V5.8 or later);
	Red Hat Enterprise Linux 4 update 8 or later on x86; (or other compatible Linux distribution)
	OpenVMS VAX V6.1 or later;
	OpenVMS Alpha V7.0 or later;
	OpenVMS I64 V8.2 or later;
	Windows 2003

Contents

Preface

Purpose of This Manual

This manual describes the structure, configuration, and use of PMDF's popstore and MessageStore. The intended audience is system managers who want to become familiar with how these two stores operate and are managed. It is assumed the reader is familiar with PMDF and the operating system on which PMDF is installed. Note also that the POP3, IMAP, HTTP, and poppassd servers used by the popstore and MessageStore are documented in the *PMDF System Manager's Guide* and not in this document. This is because, on UNIX and OpenVMS platforms, those servers also serve out message stores native to those operating systems.

This manual does not provide a description of the popstore or MessageStore suitable for end users. End users do not directly interact with these stores: instead they interact with a POP3, IMAP, or Web client which in turn, through a PMDF server, interacts with the store in question.

Note: Owing to time constraints, a fully-integrated PMDF MessageStore and popstore manual is not yet available. However, much of the documentation for the PMDF popstore applies to the PMDF MessageStore. A number of chapters in this manual will speak solely about the popstore. However, unless noted otherwise, the chapter likely applies to the MessageStore as well as the popstore. Chapter 2 provides information specific to the MessageStore as well as noting any significant differences with the popstore.

Overview of This Manual

This manual serves as both a configuration and usage guide for the popstore. It also provides technical information for programmers wanting to write code which interacts with the popstore via the popstore's API.

An overview of the popstore is provided in Chapter 1. Configuration instructions may be found in the *PMDF Installation Guide & Release Notes*. Directions on using the web-based management interface are provided in Chapter 4; see Chapters 6 (UNIX and NT) or 7 (OpenVMS) for directions on using the command line management utility.

Availability

PMDF software products are marketed directly to end users in North America, and either directly or through distributors in other parts of the world depending upon the location of the end user. Contact Process Software for ordering information, to include referral to an authorized distributor where applicable:

Process Software, LLC
959 Concord Street
Framingham, MA 01701 USA
+1 508 879 6994
+1 508 879 0042 (FAX)
sales@process.com

1 Overview

1.1 Introduction

The PMDF popstore is a message store streamlined for use with POP3 clients. It is distinct from the Berkeley and VMS MAIL mail box message stores traditionally used on UNIX and OpenVMS platforms. For a given message, a single copy is stored for all recipients. Moreover, the message is stored in a ready-to-download format; *i.e.*, the server can just map the file into memory and send it down the TCP connection without the need for any pre-processing of the message data as is the case with many stores such as Berkeley and VMS MAIL mail boxes.

The popstore is primarily designed for scalability. Central database files, a principal cause of bottlenecks in high volume settings, are avoided.[1] In a similar vein, the underlying message store itself can be spread across any number of disks.

The major components of the popstore are described below.

Legacy and popstore POP3 server

A dual-store, multi-threaded POP3 server is provided on UNIX and OpenVMS platforms which supports both the PMDF popstore and the legacy UNIX Berkeley and OpenVMS MAIL mailbox formats.

MessageStore and popstore POP3 server

A new dual-store, multi-threaded POP server is provided on all platforms which supports both the popstore and MessageStore. This server includes security and performance enhancements not possible while maintaining support for legacy mailbox formats.

Poppassd server

A multi-threaded poppassd server for users of Mulberry, Eudora, and other clients which support the *ad hoc* `poppassd` protocol for changing passwords.

Web-based user interface

A basic web-based user interface is provided. This interface allows users to use a web-client to change their password as well as see basic usage information about their popstore account. They can also read and delete messages stored for their account. For details, see Chapter 5.

Web-based management utility

A web-based management utility to manage the popstore. The utility presents itself as a multi-threaded CGI accessed through the PMDF HTTP server. Popstore users with management privileges can use this interface to monitor and manage the popstore. This utility is extremely reconfigurable; the entire interface can be changed around to suit a site's needs. See Chapter 4 for a description of this interface.

[1] Note that the popstore does use some databases. However, these databases are only used to expedite management operations such as listing accounts and maintaining information about management groups. They are not used for non-management operations and therefore do not impact the performance of the popstore.

Command line management utility

A command line oriented management utility. Users with operating system privileges as well as popstore users who have been granted popstore management privileges can use the utility. See Chapters 6 or 7 for information about this utility.

Migration utility

A utility is provided to migrate the mail inboxes for login accounts to the popstore. The utility can create a popstore account for each migrated user, migrate their mail inbox, and then establish mail forwarding from their login account's message store to the popstore. See Chapter 8 for details about this utility.

Forwarding database

A forwarding database which allows mail for popstore users, fictitious or otherwise, to be automatically redirected elsewhere. Consult Section 1.5 for further details.

Delivery channel

A master channel to deliver inbound messages to the popstore. Inbound messages for the popstore are queued to this channel by PMDF. The channel is then run by PMDF to deliver the messages to the popstore. See Section 10.1 for details.

Message bouncer

This is a job which runs periodically and either returns or deletes old stored messages which have “expired”. This channel is best likened to the PMDF RETURN job. This job is used to “time out” old messages which popstore users have not deleted. If an old message has never been read, it is returned as undelivered. If it has been read, it is deleted. Note that the popstore can be configured to never delete old messages and just keep messages around indefinitely. See Section 10.2 for further discussion.

Validating accounts

The immediate validation of accounts is turned on by default on the msgstore channel so that it can, when presented with a popstore address, immediately check to see if it is valid or not (*e.g.*, is it a valid recipient address, is the recipient allowed to receive new messages, *etc.*). This allows the various incoming mail streams to reject up front invalid messages for the popstore thereby obviating cases where the message is received only to then have to be bounced. Section 10.1.1 contains additional information on this account validation.

API

An API for sites who want to generate their own management, accounting, billing, logging, *etc.* facilities. In addition, agents which access the popstore or manipulate user accounts can be written using the API. See Chapter 12 for further details.

1.2 Licensing

Although installed as part of the base PMDF-MTA product, the PMDF popstore is licensed separately. The popstore can, however, be used without a license: sites without a PMDF-POPSTORE license can create up to ten popstore user accounts plus a `default` account. A PMDF-POPSTORE license enables a site to create more than ten user accounts.

From the interactive command line management utilities, the `show -count_users` command (`SHOW/COUNT_USERS` on OpenVMS) can be used to display the number of currently defined accounts as well as the limit allowed by your license. The web-based management utility can also display this information; see the "License limits" link in the menu at the top of the main page.

1.3 Accounts

Each user of the popstore has a popstore account. Accounts have several key attributes:

Attribute	Description
Name	The username used to identify the account.
Password	The secret used to access the messages stored for the account. Note that passwords are case sensitive. The popstore stores the passwords in an encrypted form.
Group	Management group to which the account belongs. Use of management groups is optional.
Quota	A storage quota which limits the total amount of messages which an account can store.
Usage flags	Flags used to control usage of the account. For example, whether or not the account can receive new messages.
Accounting information	Accounting information such as time of last connect, total connect time, current storage, *etc.*

Accounts are created with either the web-based or command line management utilities. Sites can also develop their own utilities using the API.

Of particular importance is the account name and password. The account name specifies the mailbox for the popstore account. That is, if the popstore has the domain name `sample.example.com` then a popstore account with the name `jdoe` would have the e-mail address

```
jdoe@sample.example.com
```

Also, the popstore supports the concept of subaddresses. If an address contains a plus sign, `+`, in the local part then the plus sign and any characters to the right of it up to the at sign are ignored. For instance, the user `jdoe` can want to identify mail they receive from the HBD mailing list by subscribing themselves to that list with the address

```
jdoe+hbd@sample.example.com
```

Mail coming in to the popstore with that address will then be delivered to the account `jdoe` and not `jdoe+hbd`.

POP users access the mail stored for their account by supplying to their POP3 client their popstore account name and password. The rules for account names and passwords are given in Section 1.3.1 and Section 1.3.2.

Note that there is a special account — the `default` account — which is created at the time that the popstore is configured. The settings for the `default` account serve as account defaults. When new accounts are created, their defaults are copied from the `default` account.

The complete list of account fields is given in Table 1–1. Their interpretation and usage are made clear throughout the remainder of this manual.

Table 1–1 popstore Account Fields

Field name	Description
`version`	Data structure version number indicating what revision of the popstore account data structure is used by the account.
`store_type`	Storage type: popstore or MessageStore.
`flags`	Account usage flags.
`ulen`	Length in bytes of the value stored in the `username` field.
`plen`	Length in bytes of the value stored in the `password` field. The value stored in the `plen` field is stored in an encrypted form.
`olen`	Length in bytes of the value stored in the `owner` field.
`slen`	Length in bytes of the value stored in the `private` field.
`username`	The account's name. Maximum length of this field is 32 bytes.
`password`	The account's password. Maximum length of this field is 32 bytes. The value stored in this field is encrypted.
`owner`	Information about the account's owner. Maximum length of this field is 40 bytes.
`private`	Site-defined data field. Maximum length of this field is 64 bytes. The contents of this field can be set through the API or either of the management interfaces.
`quota`	The account's primary storage quota measured in bytes. A value of zero indicates unlimited storage quota.
`return_after`	How long to retain messages in the popstore before returning them.
`overdraft`	The account's overdraft quota measured in bytes.
`last_billing`	Time when billing information for the account was last generated. When the account is created, this value is set to the creation time of the account.
`total_connections`	Total number of connections made to the account.
`last_connect`	Time when the user last connected to the account with a POP3 client.
`last_pwd_change`	Time when the user last changed their password.
`last_disconnect`	Time when the user last disconnected from the account with a POP3 client.
`total_connect`	Total time, in seconds, spent connected to the account.
`past_block_days`	Storage block days for previously stored and since deleted or returned messages. Does not include storage values for messages currently being held in the store.

Table 1–1 (Cont.) popstore Account Fields

Field name	Description
`past_block_days_remainder`	Roundoff from the `past_block_days` field. The roundoff is measured in units of byte minutes.
`message_count`	Count of messages presently stored for the account.
`quota_used`	Total size in bytes of the messages presently being stored for the account.
`received_messages`	Cumulative number of messages which have been stored for the account.
`received_bytes`	Cumulative number of message bytes which have been stored for the account.
`glen`	Length in bytes of the value stored in the `group` field.
`group`	Management group to which this account belongs. Maximum length of this field is 16 bytes.

1.3.1 Account Naming

The popstore supports three different naming schemes: two are case-insensitive while the third is case sensitive. Note that the character set used for account names is controlled with the `USERNAME_CHARSET` option discussed in Section 3.4. The choice of character set is only of relevance to `USERNAME_STYLE` `1` and `2`. And, for those two, is only used when doing case-insensitive comparisons of account names (*e.g.*, are the account names `Sue` and `sue` identical).

1.3.1.1 The Preferred Naming Scheme

The preferred naming scheme corresponds to an option setting of `USERNAME_STYLE=3` in the popstore option file. This option is required by the PMDF MessageStore and will be included in newly generated PMDF configurations.

With this setting,

- usernames are case-sensitive,
- can contain any printable `UTF-8` characters except for

  ```
  / @ % +
  ```

  and,
- must not begin with an underscore, _.[2]

[2] A leading underscore character in a recipient address is interpreted by the delivery agent as a request to ignore any forwarding for that recipient. For instance, a message for _jdoe@pop.example.com is delivered to the popstore account `jdoe` regardless of any popstore forwarding which might exist for that account. A plus sign and any characters to the right of it but before the at sign, @, are also ignored. For instance, mail to the address joe+hbd-list@example.com would be delivered to the popstore account `joe`.

However, for interoperability with Internet protocols it is safer to confine usernames to alphanumeric characters and

```
- _ .
```

If you want to have case-insensitive behavior, create all your accounts using lower-case names, and use

```
$\$U$_
```

in appropriate domain rewriting rules to convert usernames to lower case on message delivery. Also, set `TRANSLATE=ASCII-NOCASE` in your `security.cnf` file to convert login usernames to lower case.

1.3.1.2 The Default Naming Scheme

For backwards compatability, the default naming scheme corresponds to an option setting of `USERNAME_STYLE=2` in the popstore option file. That option file is described in Chapter 3.

In the default scheme, account names can be up to 32 characters long and can contain any of the characters from the set

```
0 1 2 3 4 5 6 7 8 9 . _ -
a b c d e f g h i j k l m n o p q r s t u v w x y z
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
```

Since account names are treated as being case insensitive, the names `Anne` and `anne` specify the same account.[3] Account names should not begin with an underscore character, "_".[2]

1.3.1.3 Another Obsolete Naming Scheme

A third, and largely obsolete naming scheme corresponds to the option setting of `USERNAME_STYLE=1` in the popstore option file. That option file is described in Chapter 3.

In this naming scheme, account names can contain any printable character in the DEC MCS or any one of the ISO Latin 1 through Latin 9 character sets (ISO 8559-1 through ISO 8859-9). That is, any character with decimal ordinal value in the ranges 33—126 or 161—255 (0x21—0x7E or 0xA1—0xFF in hexadecimal). On UNIX and NT platforms, the account names can be up to 32 characters long; on OpenVMS platforms up to 19 characters. Note that the names can contain spaces and punctuation characters. Again, the names are treated as being case insensitive. The account names should not begin with an underscore character, _, or contain plus signs, +.[2].

When a name is specified, it is converted to an account name using the following five steps:

[3] For storage purposes, account names are stored in lowercase.

1. All control characters are removed from the name. That is, all characters with decimal ordinal values in the ranges 0—31 or 127—159 are removed (0x00—0x1F or 0x7F—0x9F in hexadecimal).
2. All leading and trailing white space characters are removed. White space characters are the characters SPACE (0x20), TAB, (0x09), and non-breaking space (0xA0).
3. Each internal white space character is replaced with a `SPACE` character.
4. Consecutive `SPACE` characters are replaced with a single SPACE character.
5. All uppercase characters are converted to lowercase.

The choice of character set is specified with the `USERNAME_CHARSET` option as described in Chapter 3.

1.3.2 Account Passwords

Passwords are used to authenticate a would-be user of the popstore. That is, a user wanting to access the popstore must supply a valid popstore account username as well as the password for that account. The user-supplied password is checked against the password stored for the account: only if they match is access permitted.

1.3.2.1 Password Location

Each popstore account requires a password. A password can either be stored with the popstore account's profile or it can be stored externally outside of the popstore. Regardless of where the password is stored, it can be set and changed with the popstore management and user interfaces as well as with the PMDF poppassd server. PMDF's authentication API is used by the popstore both to authenticate as well as set or change passwords. By default, this means that when a password is authenticated, it will be checked against

1. the user's popstore profile provided that the `PWD_ELSEWHERE` usage flag is not set for that profile;
2. or the PMDF password database if an entry exists for the user;
3. or the operating system's password database if an entry exists for the user.

The first password entry found for the user is that used to compare against the supplied password. If the password supplied by the user matches the password in the entry found, then access is permitted. If the supplied password does not match, access is denied. Further password entries are not checked.

When a password is set or changed for a popstore user, it will by default be

1. added or changed in the user's popstore profile provided that the `PWD_ELSEWHERE` usage flag is not set for that profile;
2. and changed in the PMDF password database provided that a previous entry for the user already exists;

3. and changed in the operating system's password database provided that an entry for the user already exists.

The above actions are the default behavior—the behavior when PMDF's authentication facilities are using their default configuration. Those facilities can be reconfigured to act differently and even to use other repositories of password information (*e.g.*, an LDAP server or other authentication server). For more information on PMDF's authentication facilities, see the "Connection Authentication and Password Management" chapter of the *PMDF System Manager's Guide*. When using those facilities, the popstore uses a service name of `POP`.

When a password is stored with the account's profile, the plain text form of the password is encrypted and stored. Such passwords are

- case sensitive,
- can contain any bytes (0x00—0xff), and
- can be up to 32 bytes long.

Accounts with a zero length password are "public" accounts: any password will access such an account.

To store a password in an external source, mark the account with the `PWD_ELSEWHERE` usage flag. This tells the popstore that the password is stored externally. See also the information on password migration in the *PMDF System Manager's Guide*.

The POP3 server uses PMDF's authentication services and as such supports plain text as well as `APOP`, `CRAM-MD5`, and `DIGEST-MD5` password authentication. Moreover, the POP3 server supports SASL (RFC 2222).

POP users can change their account passwords with PMDF's poppassd server, as described in Section 11.3, or with the web-based user interfaces described in Chapter 5. The poppassd server implements an *ad hoc* password changing protocol employed by several popular POP3 clients such as Eudora.

1.3.2.2 Using the Operating System's Password Database

With the default configuration of PMDF's authentication services, using the operating system's password database is quite straightforward. Simply set the PWD_ELSEWHERE usage flag for each popstore account which is to use the operating system password database.[4] Once this has been done, authentication will be performed against

1. the PMDF password database if an entry exists for the user;
2. or the operating system's password database if an entry exists for the user.

Since, by default, users do not have entries in the PMDF password database, authentication will be performed against the operating system's password database (*e.g.*, `/etc/passwd` on UNIX systems and `sysuaf.dat` on OpenVMS systems). Note, however, that this requires that there be an entry for the user in the operating system's

[4] You can also want to effect that setting for the `default` account so that new accounts automatically have that setting.

password database. Usually, this means that the user will also have a login account on the system.

1.3.2.3 Password Security

If a password is stored with the account's profile, the popstore provides several security features.

- The time that a password is changed is recorded. This information is displayed in the output of the command line management utility and the web-based management interface.
- The popstore has support for password expiration, minimum password length, and basic "reasonableness" checks. These features are enabled by using the options PASSWORD_LIFETIME, PASSWORD_MINIMUM_LENGTH, and PASSWORD_REASONABLENESS, respectively.
- Sites may also apply additional validation or "reasonableness" checks through the VALIDATE_PASSWORD subroutine (see Section 14.3).
- Accounts can be marked "pre-expired", which forces the user of the account to change the password immediately. Note that "pre-expired" is only meaningful if password expiration has been enabled. An account can be marked pre-expired, or marked not pre-expired, using either the command line or web-based management interfaces.
- A utility is available to go through all accounts and mark them as not pre-expired. To run the utility, for example on unix:

```
# /pmdf/bin/unpreexpire
```

Note: As soon as password expiration is enabled, all accounts created by versions of PMDF prior to V6.2-1 are automatically pre-expired.

1.3.3 Account Quotas

Account quotas can be used to control how much message storage a given account can have. When an account exceeds its storage quota, as measured in bytes of disk storage, the account can not receive new mail messages. The user must delete some of their stored messages in order to receive new mail messages.

Each account has two storage quotas: a primary storage quota and an overdraft quota. This two-quota scheme is used for efficiency purposes. With most message transfer protocols, the size of an incoming message is not known upfront and a message has to be received in its entirety in order to determine its size. Use of an overdraft quota allows PMDF to temporarily refuse to accept incoming messages for an over quota user. How so? If a user has just one quota limit, then odds are they will never quite attain it. For instance, if they have 100 spare bytes of storage they are not over quota and so the SMTP server will accept new messages for the user. However, most received messages will be larger than 100 bytes and will thus need to be bounced. This will continue until the user receives enough small messages to exactly use up those remaining 100 bytes. Once they have exactly consumed their quota, the SMTP server can now immediately refuse further messages without the need to first receive the message and see how large it is. But this is only possible once the user has exactly consumed their quota. An alternative is to have a "fudge" factor of some form. Once the user is somehow close to their quota, the server refuses additional messages. The popstore's overdraft quota is such a fudge factor. A user's quota is really a quota range with a lower limit given by their message storage quota and the upper limit given by the sum of their message storage quota and their overdraft quota.

All that having been said, note that by default PMDF will not reject incoming messages for an over quota user. By default, PMDF accepts the messages and holds onto them until either the user has available quota to receive the messages or the messages "times out" and are returned by PMDF's message bouncer. To have PMDF reject incoming messages for an over quota user, specify `REJECT_OVER_QUOTA=1` in the popstore option file as described in Section 3.4. When `REJECT_OVER_QUOTA=1` is specified and a user is over quota, the message copy for the over quota user will be rejected with a temporary error message.

If desired, accounts can be granted unlimited storage quota as denoted by a primary quota value of zero.

1.3.4 Management Groups

For management and accounting purposes, you can associate with each popstore account a group name. Use of group names is optional. You can ignore this feature for the time being and later, if a need should arise, then begin using it.

Group names are case-insensitive and can be zero to sixteen bytes long. Group names are shared across all user domains. When you create a new account and do not specify a group name for the account, the account is placed in the same group as the `default` account. By default, the `default` account is not in any group — it has a group name of zero length. This group with a zero length name is referred to as the `world` group.

There are two primary uses for management groups:

1. Accounting: For instance, listing or billing all accounts within the same management group.
2. Management: Allowing a privileged account to only manage a subset of the popstore accounts.

In regards to the latter usage, a privileged popstore account — that is a popstore account with the `MANAGE` flag set — can perform management functions on only those accounts within the same user domain and management group. A privileged popstore account which is in no group (and hence is in the `world` group) can manage all popstore accounts within the same user domain. However, a privileged popstore account which is in no group *and* is in the `default` user domain can manage *all* accounts within all user domains and groups.

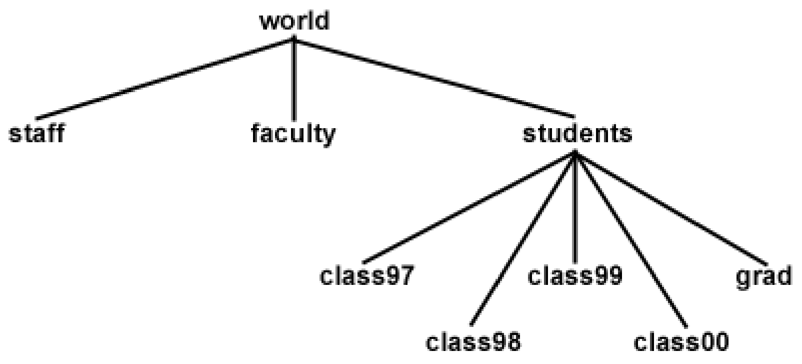
The actual details behind defining groups and assigning group names to accounts is documented in Chapters 4 — 7 as part of the discussions of the various account management commands. Note that only two classes of users can create, delete, or modify group definitions. These classes are (1) users with operating system privileges, and (2) popstore users with privileged account which themselves are in no group [and thus are in the `world` group]. The first class can use the interactive command line management utilities; the second class can use either the interactive command line or web-based management utilities.

Groups can be nested. That is, a group can contain subgroups and those subgroups can contain further subgroups. An account is contained in the group *G* if either (1) the account's group name is *G*, or (2) the account's group is a subgroup (nested arbitrarily deep) of the group *G*. The `world` group — the group with zero length group name — is a distinguished group: it implicitly contains all other groups.

You can visualize typical group hierarchies as an inverted tree. A hypothetical example with nine groups is shown in Figure 1–1. The root of the tree is the `world` group, which contains all other groups. The `staff` and `faculty` groups have no subgroups. The `students` group, however, has five subgroups.

Group hierarchical structure need not be a tree: in the language of graph theory, loops are allowed. For instance, a group can be a subgroup of more than one group. Modifying the example of Figure 1–1, it is possible for `grad` to also be a subgroup of `staff`.

Figure 1–1 Example management groups



The ability to nest groups is particularly useful in regards to account management. An account with management privileges can manage any account within the same group as the privileged account itself. Thus, if a privileged account has a zero length group name, then that account can manage any and all accounts. If, however, the privileged account is in a group with a non-zero length group name, then that privileged account can only manage accounts contained within the same group. For instance, a privileged account in the `students` group can manage any account in that group; *i.e.*, any account with group name `students`, `class97`, `class98`, `class99`, `class00`, or `grad`. A privileged account in the group class97 can only manage other accounts in the group class97. If you want to have an account which can manage both staff and faculty but not students, then just create a new group named, for instance, dean and make staff and faculty be subgroups of that new group. Then, make the group name for the privileged account be dean.

If you want to allow an account to manage all accounts yet be in a named group such as manager, then create a group named manager and make the world group be a subgroup of the manager group.

1.3.5 User Domains

By default, all popstore accounts are considered to be part of the same user domain called the default domain. This is true regardless of the e-mail address used to reach the account's mailbox. At some sites, however, it is useful to have distinct sets of user communities, each differentiated by a distinct Internet host name. For instance, one community can be the example.com community with mail addressed to user@example.com while another community can be the sample.com community with mail addressed to user@sample.com. In the popstore,[5] each community can be assigned a different Internet host name with an associated community name called a user domain name.

The popstore user user in the user domain host has the e-mail address user@host.[6] The delivery channel, which can deliver mail for several different user domains, determines which domain or domains a message should be delivered to by examining each envelope recipient address. The non-legacy POP server determines which user domain a client is in from either the username presented by the client or from the PORT_ACCESS mapping table. In regards to the former, the client must present a username of the form user%host or user@host.[7] In regards to the latter, consult the PORT_ACCESS mapping table documentation in the *PMDF System Manager's Guide*. Use of that table allows selection of the user domain to be based upon such information as the client's source IP address or the TCP port which the client has been configured to use for POP service.

When managing the popstore via the Web-based management utility, the user domain to manage is specified by including it in the URL; for example, to manage the example.org user domain, use the URL

```
http://host:7633/popstore/example.org/admin.html
```

When using user domains, there will be a special user domain referred to as the default domain. The official host name for the delivery channel will be mapped to the default domain. Moreover, a privileged management account in the default domain which is in no group can manage any account within the popstore regardless of user domain. To make this a little more clear, consider the channel definition

[5] This functionality is not yet supported by the PMDF MessageStore.

[6] Accounts in the default user domain would use the delivery channel's official host name rather than @default.

[7] Note that not all clients can handle a username of user@host: attempts to configure such a username are sometimes interpreted by the client as meaning, "The username is user and the POP server is host". For this reason, users can achieve better results by configuring their clients with user%host as their username.

```
popstore defragment holdexquota
example.com
example.org
```

In the above, the host example.com is the channel's official host name and therefore identified with the popstore's `default` user domain; the host example.org is identified with the `example.org` user domain.[8] Mail to `jdoe@example.com` is delivered to the account `jdoe` in the `example.com` user domain. Mail to `jdoe@example.org` is delivered to the separate account `jdoe` in the user domain `example.org`. Accounts associated with the `example.com` host are managed by managing the `default` user domain; accounts for the sample.com host are managed via the `example.org` user domain.

See Sections 6.13 and 7.13 for detailed information on creating a user domain and managing accounts within it.

1.3.6 Account Usage Flags

Through “usage flags”, account access to the popstore can be controlled:

DISMAIL
The `DISMAIL` flag is used to prevent an account from receiving new mail messages. When this flag is set for an account, new messages are rejected and returned to their sender. The account owner can, however, read any existing messages they might have unless the account is also flagged with the `DISUSER` flag.

DISUSER
The `DISUSER` flag is used to deny access to an account. The account can, however, continue to receive new messages unless it is either over quota or also flagged with the `DISMAIL` flag. When the user attempts to access their account, they will be met with an “account disabled” error message.

LOCKPWD
The `LOCKPWD` flag prevents users from changing their account's password. The password can only be changed by a user with the management privilege or operating system privileges.

MANAGE
Accounts with this flag can use the web-based interface to manage the popstore. In addition, users with unprivileged login accounts to the platform running the popstore can manage the popstore using the command line interface when their popstore account has the `MANAGE` flag set. This is accomplished through the command line interface's `LOGIN` command. Section 1.3.7 for further details.

MIGRATED
This is a flag used by the PMDF's migration utilities to track whether or not migration from an external source to the popstore has been completed for a given popstore user account.

[8] Note that if the `USER_DOMAINS` option is set to its default value of zero in the popstore option file, then all domains associated with the popstore channel will be identified with the popstore's `default` user domain.

PWD_ELSEWHERE
This flag tells the popstore that the user's password information is stored externally, outside of the popstore. The popstore's authentication mechanisms use this flag when determining how to authenticate a user password. See Section 1.3.2 for further details.

With the exception of the `MANAGE` and `NOMANAGE` flags, these flags can be set or cleared with either the web-based or command line management interfaces. As a security precaution, the `MANAGE` and `NOMANAGE` flags can only be manipulated through the command-line interface.

Note that PMDF itself has a variety of other access control mechanisms such as the `SEND_ACCESS` mapping table to control access at the envelope address level and the `PORT_ACCESS` mapping tables to control access at an IP level. See the *PMDF System Manager's Guide* for details on these and other access mapping mechanisms provided by PMDF.

1.3.7 Privileged Accounts

The popstore has the concept of "privileged" popstore accounts. These are popstore accounts which have the `MANAGE` usage flag set. Only accounts with the `MANAGE` flag set can use the web-based management interface. As a security precaution, this flag can not be set or cleared through the web-based interface. That means that the command line interface must be used to grant an account management privileges. That, in turn, requires a priviliged login account to the operating system running the popstore. From that login account, one or more popstore accounts can be granted management privileges. Those accounts can then be used to manage the popstore either through the web interface or, as described in Sections 6.14 and 7.14, the command line utility.

Privileged accounts can only manage other accounts within the same management group and user domain. If a privileged account is in the `world` group (*i.e.*, is not in any group or is in a group which explicitly contains as a subgroup the `world` group), then it can manage any popstore account within the same user domain. However, a privileged popstore account which is in the `world` group *and* is in the `default` user domain can manage *all* accounts within all user domains and groups.

1.3.8 Bulk Loading

Accounts can be created *en masse* using the command line management utility. This is done by creating a file of commands, one command per line, and then directing the utility to process the commands from that file. For further details, see Sections 6.7 (UNIX and NT) or 7.7 (OpenVMS).

1.3.9 Account Storage

The information associated with each user account is stored in a “profile” file on disk. One file per user account is used. An account’s settings, usage information, and list of currently stored messages is stored in its profile file. On UNIX systems, the location of these files are determined via the `PMDF_POPSTORE_PROFILES` option in the PMDF tailor file; on NT systems, the `PMDF_POPSTORE_PROFILES` registry entry is used, and on OpenVMS systems, in the `PMDF_POPSTORE_PROFILES:` directory tree.

Read and write access to these files is controlled using private locks. As such, they should only be accessed using the popstore API routines documented in Chapter 12.

Each profile file has a base size of 256 bytes plus 40 bytes per message stored for the user. Presently, the profile files are interchangeable amongst the different platforms on which the popstore runs. That is, for instance, it is presently possible to move the popstore from one platform to another without the need to reformat the profile files.[9] See Section 8.1 for complete details on migrating the popstore to another platform.

Note: The profile files must be stored on a disk with a reliable file system. The profile files must not be accessed via NFS: *NFS, even with a lockd daemon, does not provide adequate file locking, integrity, or performance and is not supported.*

1.4 Messages

When PMDF receives a message for the popstore, the message is queued to the popstore’s inbound delivery channel. That channel then processes each queued message and stores the resulting messages in the popstore’s message store. A single message copy is stored for all recipients of a message. Once a message has been deleted by each recipient, it is deleted from the store itself.

Message storage quotas are set on a per-user basis via the quota and overdraft quota account settings. See Section 1.3.3 for further details.

Users access their stored mail messages via their POP3 client and the popstore’s POP3 server. Ideally, users download their messages and delete them from the store itself. By default, if a message is not deleted within a fixed number of days, the message will be deleted silently. Should one or more of the recipients not have read the message, a non-delivery notification is sent to the message’s originator prior to deleting it.

Popstore administrators can delete message files using either of the two management utilities. Optionally, when a message file is deleted, a non-delivery notification can be sent to the originator of the message. The non-delivery notification will state which recipients had not read the message.

[9] While Process Software hopes to maintain this level of portability in the future, it can not be possible, at which point a conversion utility will be provided.

The actual message files are binary files stored in a ready-to-download format. On UNIX systems, these files are kept in the directory tree specified by the `PMDF_POPSTORE_MESSAGES` option in the PMDF tailor file; on NT systems, the `PMDF_POPSTORE_MESSAGES` registry entry is used; and, on OpenVMS systems, in the `PMDF_POPSTORE_MESSAGES:` directory tree. Read and write access to these files is controlled using private locks. The files should only be accessed using the popstore API routines as documented in Chapter 12.

The size of each message file varies depending upon the amount of envelope information and message content which must be stored. Presently, the message files are interchangeable amongst the different platforms on which the popstore runs. That is, for instance, it is presently possible to move the popstore from one platform to another without the need to reformat the message files. While Process Software hopes to maintain this level of portability in the future, it can not be possible, at which point a conversion utility will be provided.

Note: The message files must be stored on a disk with a file system which supports byte range file locking. The message files must not be accessed via NFS: *NFS, even with a lockd daemon, does not provide adequate file locking, integrity, or performance and is not supported.*

1.5 Forwarding Mail

The popstore includes a forwarding database which can be used to re-route mail for the popstore to other addresses. The addresses can be either internal or external to the popstore. Moreover, forwardings need not correspond to actual popstore accounts. For instance, if mail for staff@example.com is to be forwarded to a PMDF mailing list, then there need not be a `staff` account in the popstore. Note that when a forwarding is established for a popstore account, the popstore account itself will not receive copies of its forwarded messages.

Forwardings are recursive. That is, if a popstore address has a forwarding which in turn points to another popstore address, then that new address will also be checked for a forwarding. In addition, a forwarding can point to more than one address. That is, a forwarding can forward to multiple addresses and, moreover, some of those addresses can themselves be forwarded.

Forwardings are established, examined, and removed through either of the management interfaces. The forwardings are stored in an ordinary PMDF CRDB database — the `PMDF_POPSTORE_FORWARD_DATABASE`. If desired, the database can be manipulated using ordinary PMDF tools as well as through the PMDF API.

Messages destined to the popstore can explicitly defeat any forwarding by prefixing the address with an underscore character, _. For instance, if the account “`jdoe`” has mail forwarded elsewhere, then a message sent to the address `_jdoe@example.com` will bypass that forwarding and be delivered to the “`jdoe`” account. To explicitly forbid such bypassing, the `DISMAIL` usage flag can be set for the account in which case mail to `_jdoe@example.com` will be returned as undeliverable.

Finally, note that it is the popstore delivery channel which actually effects mail forwardings. When processing message files, it checks each recipient to see if a forwarding exists. If it does, it then uses the forwarding address instead. If the forwarding address points back to the popstore, it then checks that address for a forwarding. This process is iterated up to ten times; an address which is forwarded internally more than ten times is deemed to be a forwarding loop and is rejected. If the forwarding address is external to the popstore, then a new message is enqueued with the forwarding address used as its envelope `To:` address.

2 The PMDF MessageStore

Owing to time constraints, a fully-integrated PMDF MessageStore and popstore manual is not yet available. However, much of the documentation for the PMDF popstore applies to the PMDF MessageStore. This chapter supplements the PMDF popstore documentation with information specific to the PMDF MessageStore.

2.1 Introduction

As with the popstore, end users do not directly interact with the MessageStore: instead they interact with an IMAP or POP3 client which in turn, through the PMDF MessageStore IMAP or POP3 server, interacts with the MessageStore.

Whereas, the popstore is a message store streamlined for use with POP3 clients, the PMDF MessageStore is a message store streamlined for use with IMAP clients, and, incidentally, also supports POP3 clients.

The MessageStore is primarily designed for IMAP scalability and manageability. It supports public folders so that a single mailing list subscription can be shared by a large community of users. Responses to common IMAP operations are pre-computed at delivery time, and the messages are stored in a ready-to-download format for IMAP without the need for any pre-processing of the message data. In order to simplify backup and restore of a single user, each user's mail is stored in a single directory subtree.

The major components of the MessageStore are described below.

MessageStore and popstore POP3 server

A new dual-store, multi-threaded POP server is provided on all platforms which supports both the popstore and MessageStore. This server includes security and performance enhancements not possible while maintaining support for legacy mailbox formats.

MessageStore IMAP server

A new multi-threaded IMAP server is provided to access the MessageStore. This server supports the `IMAP4 ACL` and `QUOTA` extensions so that existing clients can be used to directly manage shared folders and quotas through the `IMAP` protocol.

Poppassd server

A multi-threaded poppassd server for users of Eudora, Mulberry, and other clients which support the *ad hoc* poppassd protocol for changing passwords.

Web-based user interface

A basic web-based user interface is provided. This interface allows users to use a web-client to change their password as well as see basic usage information about their account. Unlike the user interface for the popstore, this interface does not allow MessageStore users to access mail stored for their account. For details, see Chapter 5.

Web-based management utility

A web-based management utility to manage the MessageStore. The utility presents itself as a multi-threaded CGI accessed through the PMDF HTTP server. MessageStore users with management privileges can use this interface to monitor and manage the MessageStore. This utility is extremely reconfigurable; the entire interface can be changed around to suit a site's needs. See Chapter 4 for a description of this interface.

Command line management utility

A command line oriented management utility. Users with operating system privileges as well as MessageStore users who have been granted MessageStore management privileges can use the utility. See Chapters 6 or 7 for information about this utility.

Migration utility

A utility is provided to migrate the mail inboxes for native login and popstore accounts to the MessageStore. The utility can create a MessageStore account for each migrated user, migrate their mail inbox, and then establish mail forwarding from their prior account's message store to the MessageStore. See Chapter 8 for details about this utility.

Forwarding database

A forwarding database which allows mail for MessageStore users, fictitious or otherwise, to be automatically redirected elsewhere. Consult Section 1.5 for further details.

Delivery channel

A master channel to deliver inbound messages to the MessageStore. Inbound messages for the MessageStore are queued to this channel by PMDF. The channel is then run by PMDF to deliver the messages to the MessageStore. See Section 10.1 for details.

Validating accounts

The immediate validation of accounts is turned on by default on the msgstore channel so that it can, when presented with a popstore address, immediately check to see if it is valid or not (*e.g.*, is it a valid recipient address, is the recipient allowed to receive new messages, *etc.*). This allows the various incoming mail streams to reject up front invalid messages for the popstore thereby obviating cases where the message is received only to then have to be bounced. Section 10.1.1 contains additional information on this account validation.

API

The popstore API, while including full support for the popstore, provides some limited support for creating and deleting MessageStore accounts. See Section 2.6 below and Chapter 12 for further details.

Reconstruct utility

A utility is provided to reconstruct MessageStore index and mailbox list files in the event they are corrupted. In addition, if mailboxes are restored from backup, this utility can re-integrate them into the MessageStore without the need to stop the servers. For further information on this utility, see Section 2.5.4.

2.2 Licensing

Although installed as part of the base PMDF-MTA product, the PMDF MessageStore is licensed separately. Sites without a license can create up to ten MessageStore user accounts in addition to the `default` account. A PMDF-MSGSTORE license is required to create more than ten user accounts for the MessageStore.

2.3 Accounts

MessageStore accounts have the same basic profile information as popstore accounts. Indeed, the same underlying data structure is used to describe both MessageStore accounts and popstore accounts. That structure is shown in Table 1–1. However, for MessageStore accounts the `message_count` field of that structure is not used and usually should have a value of zero.

Note in particular that the password security features described in Section 1.3.2.3 apply to both popstore and MessageStore accounts.

Note that in the MessageStore, the account name `post` is reserved. It is the mailbox name associated with the store's collection of public folders. For instance, a message for post+staff@host is delivered to the public folder named "staff".

2.3.1 Account Naming

The MessageStore supports only one account naming scheme: `USERNAME_STYLE=3`. When configuring the MessageStore, the PMDF configuration utilities will select that naming scheme automatically. See Section 1.3.1.1 for information on this naming scheme.

2.4 Messages

The MessageStore stores a copy of each message in each folder to which it is delivered.[1]

Delivery can be directed to a user's `INBOX` or to any folder in the MessageStore if the folder's IMAP ACL permits posting. A message directed to

```
user+folder@host
```

will be filed into the specified folder named `folder` if the ACL permits; otherwise, it will be delivered to the user's `INBOX`.

[1] On UNIX platforms, hard links are used to minimize disk storage when possible.

A reserved name is used to deliver to public folders: by default the name `post` is used. A message directed to

```
post+folder@host
```

will be delivered to the public folder named `folder` if public posting is enabled; otherwise, the message will be returned to its originator as undeliverable.

A user's folders and messages are stored in a subdirectory of the directory containing the user's profile file as described in Section 1.3.9. Public folders are stored in a different directory tree. On UNIX systems, public folders are kept in the directory tree specified by the `PMDF_MSGSTORE_MESSAGES` option in the PMDF tailor file; on NT systems by the `PMDF_MSGSTORE_MESSAGES` registry entry; and, on OpenVMS systems in the `PMDF_MSGSTORE_MESSAGES`: directory tree as specified by the system-wide logical, `PMDF_MSGSTORE_MESSAGES`.

The `IMAP` protocol is used to manipulate folders and messages in the MessageStore. A number of IMAP clients provide rich management facilities including `ACL` and `QUOTA` support. User management is available through a command line interface, a web-based interface, and a subset of the popstore API as described in Section 2.6.

2.5 Management

The MessageStore comes with a web-based management utility, a command-line management facility, and a reconstruct utility for managing the MessageStore. In addition, a number of third party IMAP clients and utilities are available which take advantage of the management capabilities provided by the IMAP protocol and the `IMAP ACL` and `QUOTA` extensions.

2.5.1 Web-based Management Utility

The MessageStore's web-based management utility is very similar to the popstore's web-based management utility described in Chapter 4 with the following differences:

1. The URL for the web-based MessageStore Management utility is

   ```
   http://host:7633/msgstore/admin.html
   ```

2. The template files used by the utility are located in the directory `/pmdf/www/msgstore` on UNIX and NT platforms and `pmdf_root:[www.msgstore]` on OpenVMS platforms.
3. The MessageStore does not have a command to rename user accounts, and the web-based interface does not provide access to messages in the MessageStore. In particular, the following commands are not supported with MessageStore accounts: `add_user`, `delete_message`, `delete_messages`, `rename_user`, and `show_message`. In addition, the `message_count` command parameter is not supported. Finally, the following format strings are not supported with MessageStore accounts: `message_count`, `msg_*`, and `msgr_*`.

See Chapter 4 for detailed information on the use of the web-based management utility.

2.5.2 Command Line Management Utility

The MessageStore's command-line management utility is very similar to the popstore's command-line management utility described in Chapters 6 and 7, with the following differences:

1. The command to invoke the utility is

   ```
   # pmdf msgstore
   ```

2. The template files used by the utility are located in the directory `/pmdf/www/msgstore` on UNIX and NT platforms and `pmdf_root:[www.msgstore]` on OpenVMS platforms.
3. The MessageStore does not have a command to rename user accounts, and access to stored messages is not provided via the management utilities. In particular, the following command switches—qualifiers on OpenVMS—are not supported for use with MessageStore accounts: `-message_count` and `-messages`.

See Chapters 6 and 7 for detailed information.

2.5.3 IMAP-based Management Utilities

The `IMAP` protocol provides commands to create, delete, and rename folders. Thus you can use almost any IMAP client to manage public folders in the MessageStore, simply by logging in as a user with top-level management privileges, and creating folders under the top-level "Public Folders" node.

A good use of public folders is to subscribe a public folder to a mailing list so that individual users will not have to. First, verify that the public folder exists and grants "post" rights to `anyone`. The simplest way is to make sure the `DEFAULT_ACL` option includes the `p` right after the `anyone` identifier before you create the folder. Next use your IMAP client to create the desired folder under the "Public Folders" tree. To verify that delivery to the new public folder is working, send a test message to the folder's address. Its address will be

```
post+folder@host
```

where `post` is replaced with the value of the `POST_USER` option, `folder` is the name of the new public folder, and `host` is the domain name you configured for the MessageStore. Note that the default value of the `POST_USER` option is `post`.

Once this test succeeds, set your mail client's identity so that the `From:` address is the folder's posting address. Then simply follow the instructions for mailing list subscription. This often involves sending a message to the e-mail address `listname-request@listdomain` with a subject or body of "SUBSCRIBE".

The PMDF MessageStore supports the standard IMAP Access Control List ACL and QUOTA extensions documented in RFCs 2086 and 2087. (See Table 3–1 for a list of the ACL rights letters and what they mean.) This means that advanced IMAP clients such as Mulberry and Execmail will be able to manage the access control lists on your MessageStore directly.

2.5.4 Reconstruct Utility

The MessageStore comes with a “reconstruct” utility to be used in the event of a disk corruption. The reconstruct utility rebuilds the index and cache files the MessageStore uses to support the IMAP protocol. In addition, if an individual user is restored from backup, the reconstruct utility can be used to re-integrate that user’s mailboxes into the MessageStore.

Note: There must be no IMAP clients connected to a mailbox while it is being reconstructed.

The `reconstruct` utility is available as a command within the MessageStore command-line management utility. There are three ways to use it:

1. Reconstruct index and cache files for one or more individual mailboxes. This is necessary if a mailbox got corrupted (perhaps due to a power failure in the middle of a delivery or expunge). It is also necessary to reconstruct a mailbox after restore from backup, since the `backup` utility could also snapshot the mailbox in the middle of a delivery or expunge operation. For this usage, no switches are provided; the mailboxes are simply listed:

```
msgstore> reconstruct mailbox-name-1 [mailbox-name-2 ...]
```

   Note that the MessageStore does not support messages with NUL bytes. In normal operation, PMDF will strip or downconvert such bytes prior to final delivery. However, in the event such a message is detected during the reconstruct process, BAD will be appended to the name of the message file and the message will be ignored.

2. Reconstruct index and cache files for one or more subtrees of mailboxes. This works much like (1), except it reconstructs the mailbox and all subfolders listed in the mailbox list. The `-recurse` or `-r` switch is used to specify this behavior. If no mailbox names are provided, then all mailboxes in the mailbox list will be reconstructed.

```
msgstore> reconstruct -r [mailbox-name-1 [mailbox-name-2...]]
```

3. Reconstruct the mailbox list. The mailbox list is used by the MessageStore to locate mailboxes and to provide fast responses to IMAP LIST commands. This is necessary if the mailbox list got corrupted (perhaps due to a disk error or software defect), or to re-integrate a restored mailbox which was previously deleted. Note that while the reconstruct mailbox list command is running, users will be unable to create, delete or rename mailboxes through IMAP. The `-mailbox_list` or `-m` switch is used to specify this behavior.

```
msgstore> reconstruct -m
```

   Note that this command uses the user database to determine the list of users to reconstruct, so in the event that database is corrupted, the `x-build-user-db` command must be used first.

The name of a user's INBOX in the MessageStore is

```
Other Users/user
```

and their personal folders are one level below that. Thus if the user `joe` creates the folder `stuff`, the full name for that folder would be

```
Other Users/joe/stuff
```

Since these names contain spaces, it will be necessary to enclose the mailbox names in double quotes. As a convenience, the `reconstruct` utility will prepend "Other Users/" to any mailbox name argument which does not already begin with "Other Users/" or "Public Folders/".

2.6 Application Program Interface (API)

Support for the MessageStore was added to the popstore API after the API was designed. Therefore all API routine names begin with "POPSTORE_" even if they work with the MessageStore. The routines which operate on popstore message files are not supported for use with the MessageStore. The following routines are supported for use with the MessageStore:

Table 2–1 popstore API routines which support the MessageStore

Routine Name	Comments
POPSTORE_command_d	Include `store=imap` in the command to operate on a MessageStore user. Use the `copy_user` command with the `default` account as the source to create new MessageStore accounts.
POPSTORE_end	
POPSTORE_error_to_text	
POPSTORE_format_profiles_stype	
POPSTORE_init	
POPSTORE_manage	
POPSTORE_user_begin_d	
POPSTORE_user_billing_d	
POPSTORE_user_delete_d	
POPSTORE_user_end	
POPSTORE_user_exists_d	Returns success if a matching popstore or MessageStore account exists.
POPSTORE_user_pw_change_d	
POPSTORE_user_pw_check	
POPSTORE_user_update	POPSTORE_SET_MESSAGE_COUNT is not supported for use with MessageStore accounts.

3 Options

The popstore and MessageStore have a few options controlled through the use of option files. The popstore option file contains popstore specific options as well as options shared by both the popstore and MessageStore. Use of this option file is optional for the popstore but mandatory for the MessageStore. The MessageStore option file contains options specific to the MessageStore. Both of these option files are described in the following sections.

Initial option files are created by the `configuration` utility described in the *PMDF Installation Guide* manual.

3.1 Location of the Option Files

On UNIX and NT platforms, the popstore and MessageStore option files are, respectively, the files

```
/pmdf/table/popstore_option
/pmdf/table/msgstore_option
```

On OpenVMS platforms, they are the files

```
PMDF_TABLE:popstore_option.
PMDF_TABLE:msgstore_option.
```

The option files are ordinary text files and can be created and edited with a normal text editor. Like many PMDF option files, the option files must be world readable.

Note: The POP and IMAP servers also have option files which alter their behavior. See the *PMDF System Manager's Guide* for details for information on those option files.

3.2 Option File Formats

Option files consist of several lines. Each line contains the setting for one option. An option setting has the form:

```
option-name=option-value
```

where *option-value* can be either a string or an integer, depending on the option's requirements. If the option accepts an integer value, *option-value*, a base can be specified using notation of the form *b*%*v*, where *b* is the base expressed in base 10 and *v* is the actual value expressed in base *b*.

Comments are allowed in option files. Any line that begins with an exclamation point is considered to be a comment and is ignored. Blank lines are also ignored in any option file.

3.3 Options for Both the popstore and the MessageStore

The following options, while specified in the popstore option file, affect both the popstore and MessageStore:

AVOID_LOGIN_NAMES (0, 1, or 2)

Under some circumstances, sites using a password database shared between login and popstore or MessageStore accounts may want to ensure that no popstore or MessageStore accounts are created which have the same name as a login account. By setting `AVOID_LOGIN_NAMES=1`, popstore and MessageStore accounts cannot be created which have the same name as a login account. When `AVOID_LOGIN_NAMES` is set to the value `2`, popstore and MessageStore accounts cannot be created which have the same name as a privileged login account. On UNIX platforms, a privileged login account is considered to be any account with a UID of `0` or which is in the root group. On OpenVMS platforms, a privileged account is deemed to be any account with a group UIC number less than or equal to the `SYSGEN MAXSYSGROUP` parameter or which has any default or authorized privileges other than `TMPMBX` and `NETMBX`. On NT platforms, `AVOID_LOGIN_NAMES` only prevents an account named Administrator from being created when a value of either `1` or `2` is used.

The default value of `AVOID_LOGIN_NAMES` is `0` which allows accounts of any name to be created.

Note that the `AVOID_LOGIN_NAMES` option only influences accounts created through the management interfaces and the `POPSTORE_command` and `POPSTORE_command_d` subroutines. The option does not influence accounts created through other popstore API routines such as `POPSTORE_user_create`.

COMPUTE_CONNECT (UNIX or NT file specification; OpenVMS exec-mode logical name)

This option is used to supply the name of a site-developed, shared library which the popstore will then use to compute elapsed connect time for client connections. The shared library must contain an entry point for a subroutine named `compute_connect` as described in Section 14.1. On UNIX and NT systems, the value of the option must be a full file path — the path to the file containing the shared library. On OpenVMS systems, the value of the option must be the name of a system-wide, executive mode logical whose translation value is the full file specification for the shared library. Any logical referenced by the logical must also be a system-wide, executive mode logical. Moreover, on OpenVMS systems, the shared library must be installed as a known image with the OpenVMS `INSTALL` utility.

DEBUG (bit mask)

When set to a value of `-1`, enables debugging in the popstore. When debugging is enabled, the output is appended to the file `/pmdf/log/popstore.log` (UNIX and NT) or `PMDF_LOG:popstore.log` (OpenVMS). The default value is 0 which disables all debug output.

Note that use of this option will impact performance of the popstore. Moreover, the format of the debug output is intentionally not documented as it is subject to change. Sites wanting to generate similar output should provide their own logging subroutine; see Chapter 13 for details.

HTTP_REALM (string)

Use the `HTTP_REALM` option to override the name of the HTTP authentication realm used by the popstore and MessageStore HTTP CGIs.

LOG_ACTIVITY (UNIX or NT file specification; OpenVMS exec-mode logical name)

This option is used to supply the name of a site-developed, shared library which the popstore will then use to log activity. The shared library must contain an entry point for a subroutine named `log_activity` as described in Section 13.2. On UNIX and NT systems, the value of the option must be a full file path — the path to the file containing the shared library. On OpenVMS systems, the value of the option must be the name of a system-wide, executive mode logical whose translation value is the full file specification for the shared library. Any logical referenced by the logical must also be a system-wide, executive mode logical. Moreover, on OpenVMS systems, the shared library must be installed as a known image with the OpenVMS `INSTALL` utility.

LOG_ACTIVITY_MASK (bit mask)

This option can be used in conjunction with the `LOG_ACTIVITY` option to control for which events the site supplied logging routine is called. By default, the logging routine is called for all events. See Section 13.2.2 for further details.

MAP_PROFILE_FILENAME (UNIX or NT file specification; OpenVMS exec-mode logical name)

This option is used to supply the name of a site-developed, shared library which the popstore will then use to map message filenames to disk devices and directory trees. The shared library must contain an entry point for a subroutine named `map_profile_filename` as described in Chapter 14. On UNIX and NT systems, the value of the option must be a full file path — the path to the file containing the shared library. On OpenVMS systems, the value of the option must be the name of a system-wide, executive mode logical whose translation value is the full file specification for the shared library. Any logical referenced by the logical must also be a system-wide, executive mode logical. Moreover, on OpenVMS systems, the shared library must be installed as a known image with the OpenVMS `INSTALL` utility.

MESSAGE_PROFILE_VERSION (non-negative integer)

The value of this option is passed to the site supplied `map_profile_filename` subroutine; see Section 14.2 for details. When not specified in the option file, this option assumes the default value of `0`.

PASSWORD_LIFETIME (non-negative integer)

Non-negative integer value specifying in units of days, how long a password's lifetime should be before it expires and the user must change their password. To disable password expiration, specify a value of zero, `PASSWORD_LIFETIME=0`, meaning that passwords never expire.

When no `PASSWORD_LIFETIME` value is specified, password expiration is disabled.

Units other than days can be selected by immediately following the numeric value with a single character unit specifier chosen from the table below:

M, m	Units of minutes
H, h	Units of hours
D, d	Units of days
W, w	Units of weeks

For instance, a value of 14 days might be specified as

```
PASSWORD_LIFETIME=2w
```

where the value `2w` indicates two weeks.

PASSWORD_MINIMUM_LENGTH (0 <= integer <= 32)

This option is used to specify a minimum acceptable length for a password. If the password specified is shorter than `PASSWORD_MINIMUM_LENGTH`, then it is rejected. If a value of 0 is specified, or this option is not specified at all, there is no minimum length required (this is the default).

PASSWORD_REASONABLENESS (0, 1, or 2)

This option is used to specify what level of "reasonableness" checks that a password must go through to be acceptable. If the password specified does not pass the reasonableness checks, it is rejected.

A value of 0 (the default) causes no reasonableness checks to be made.

A value of 1 causes PMDF to check and make sure that the password is not the same as the username.

A value of 2 causes PMDF to check and make sure both that the password is not the same as the username, and it is not a substring of the "Owner" field (usually used to contain the real name of the user).

REJECT_OVER_QUOTA (0 or 1)

By default, PMDF will accept incoming messages for users who have exceeded their storage quota. This corresponds to the option setting `REJECT_OVER_QUOTA=0`. To have PMDF reject incoming messages for over quota users, specify `REJECT_OVER_QUOTA=1`. The rejections will take the form of a temporary error (*e.g.*, an SMTP 4yz response). Note that when this option is specified, local users can experience difficulty sending mail to over quota users. For instance, a local POP user when sending mail to an over quota user will be met with a temporary error and their client will be unable to send the message. This option is, in general, only useful when all incoming messages are coming from remote store-and-forward mail systems and not from local user agents.

USERNAME_STYLE (1, 2, or 3)

Selects the name space used for usernames. A value of `3` selects the preferred name space described in Section 1.3.1.1. A value of `2` selects the default name space described in Section 1.3.1.2. Finally, a value of `1` selects the name space described in Section 1.3.1.3.

Note that the MessageStore requires that `USERNAME_STYLE=3` be used.

Note: After creating popstore accounts, you cannot simply change the value of the `USER-NAME_STYLE` option. The value of this option influences the file names used to store profile files. Should you change this option after creating accounts, accounts created before the change can become inaccessible. Contact Process Software for assistance in changing this option with an existing popstore installation. As such, sites which have been using the popstore prior to PMDF V6.0 and now want to use the MessageStore should contact Process Software for assistance.

VALIDATE_PASSWORD (UNIX or NT file specification; OpenVMS exec-mode logical name)

This option is used to supply the name of a site-developed, shared library which PMDF will call to perform additional validation (reasonableness) checks when a user's password is changed. The shared library must contain an entry point for a subroutine named validate_password as described in Section 14.3. On UNIX and NT systems, the value of the option must be a full file path — the path to the file containing the shared library. On OpenVMS systems, the value of the option must be the name of a system-wide, executive mode logical whose translation value is the full file specification for the shared library. Any logical referenced by the logical must also be a system-wide, executive mode logical. Moreover, on OpenVMS systems, the shared library must be installed as a known image with the OpenVMS INSTALL utility.

3.4 popstore Specific Options

The following options are specified in the popstore option file and only affect the behavior of the popstore.

COMPUTE_BLOCK_DAYS (UNIX or NT file specification; OpenVMS exec-mode logical name)

This option is used to supply the name of a site-developed, shared library which the popstore will then use to compute elapsed block days of storage for message files. The shared library must contain an entry point for a subroutine named compute_block_days as described in Section 14.1. On UNIX and NT systems, the value of the option must be a full file path — the path to the file containing the shared library. On OpenVMS systems, the value of the option must be the name of a system-wide, executive mode logical whose translation value is the full file specification for the shared library. Any logical referenced by the logical must also be a system-wide, executive mode logical. Moreover, on OpenVMS systems, the shared library must be installed as a known image with the OpenVMS INSTALL utility.

MAP_MESSAGE_FILENAME (UNIX or NT file specification; OpenVMS exec-mode logical name)

This option is used to supply the name of a site-developed, shared library which the popstore will then use to map message filenames to disk devices and directory trees. The shared library must contain an entry point for a subroutine named map_message_filename as described in Chapter 14. On UNIX and NT systems, the value of the option must be a full file path — the path to the file containing the shared library. On OpenVMS systems, the value of the option must be the name of a system-wide, executive mode logical whose translation value is the full file specification for the shared library. Any logical referenced by the logical must also be a system-wide, executive mode logical. Moreover, on OpenVMS systems, the shared library must be installed as a known image with the OpenVMS INSTALL utility.

MESSAGE_FILENAME_VERSION (non-negative integer)

This option is intended for use with a site-supplied map_message_filename subroutine as described in Section 14.2. The value of this option modulo 36 is used to generate the last character appearing in the names of the files used to store messages. The value of the option is also passed to the site supplied map_message_filename subroutine. When not specified in the option file, this option assumes the default value of 0.

QUOTA_WARNING (0 <= integer <= 100)

The POP3 server can optionally generate quota warnings when a user's current storage exceeds a given percentage of their primary storage quota. The warning takes the effect of a virtual e-mail message. That is, the user appears to have a new e-mail message in their inbox which warns about their quota usage. However, this message does not actually exist: the POP server dynamically generates it.

By default, these quota warnings are not generated. This corresponds to the option setting `QUOTA_WARNING=0`. To warn users when they have reached 90% of their allowed primary quota, specify `QUOTA_WARNING=90`.

Note that the POP server `OVER_QUOTA_MSG_FILE` option may be used to specify a file containing customized warning message text. This option is specified in the `PMDF_POP3_CONFIG_FILE` configuration file for the legacy POP server and in the `PMDF_IMAPPOP_CONFIG_FILE` configuration file for the MessageStore POP server. See the *PMDF System Manager's Guide* for details.

RETURN_AFTER (non-negative integer)

Non-negative integer value specifying in units of days, how long to retain messages in the store. Any message exceeding the specified age will be silently deleted from the message store. A non-delivery notice is sent back to the message's originator, if one or more of the message's recipients had not read the message. To retain messages indefinitely, specify a value of zero, `RETURN_AFTER=0`.

When no `RETURN_AFTER` value is specified, a maximum message age of 14 days is assumed, `RETURN_AFTER=14`.

Units other than days can be selected by immediately following the numeric value with a single character unit specifier chosen from the table below:

M, m	Units of minutes
H, h	Units of hours
D, d	Units of days
W, w	Units of weeks

For instance, the default value of 14 days might be specified as

```
RETURN_AFTER=2w
```

where the value `2w` indicates two weeks.

USER_DOMAINS (0 or 1)

By default, all popstore accounts are considered to be part of the same user domain called the `default` domain. This is true regardless of the e-mail address used to reach the account's mailbox. This default behavior corresponds to the setting USER_DOMAINS=0.

By setting `USER_DOMAINS=1`, the use of user domains is enabled. This allows distinct domains of users to be established, each domain distinguished by the host name associated with the e-mail address used for the accounts in that domain.

Note that if you are using `USER_DOMAINS=0` and have more than one host listed in the delivery channel's definition,

```
popstore defragment holdexquota
official-host-name
second-host-name
third-host-name
fourth-host-name
...
```

then setting USER_DOMAIN=1 will cause the mailboxes for the second, third, fourth, *etc.* hosts to appear to disappear. Specifically, with the above channel definition and USER_DOMAINS=0, all the mailboxes for all those hosts are stored in the same user domain: the default user domain. For example, the e-mail addresses sue@official-host-name and bob@third-host-name correspond, respectively, the the accounts sue and bob in the default user domain. When USER_DOMAINS=1 is set, only the mailboxes for the first host, official-host-name, are associated with the default domain. Mail for bob@third-host-name will be for the account bob in the user domain named third-host-name. That mail will bounce until such time that the bob account is moved from the default user domain to the third-host-name domain, or a new bob account is added to the third-host-name domain.

Note: You must not use the filter channel keyword on the popstore or MessageStore delivery channel if USER_DOMAINS=1 is set. So doing will cause the wrong filters to be used for users in domains other than the default domain.

Note: The legacy POP3 server does not support user domains.

USERNAME_CHARSET (0-9)

The USERNAME_CHARSET option specifies the character set used for popstore account usernames and is only used when USERNAME_STYLE=1 is set.[1] This character set information is in turn used by the popstore to perform upper case to lower case conversions on usernames presented to the popstore. The valid values for this option are

Value	Character Set	Value	Character Set
0	ASCII, DEC-MCS	5	Latin 5 (ISO-8859-5)
1	Latin 1 (ISO-8859-1)	6	Latin 6 (ISO-8859-6)
2	Latin 2 (ISO-8859-2)	7	Latin 7 (ISO-8859-7)
3	Latin 3 (ISO-8859-3)	8	Latin 8 (ISO-8859-8)
4	Latin 4 (ISO-8859-4)	9	Latin 9 (ISO-8859-9)

The default character set, when no USERNAME_CHARSET option is specified, is Latin 1 and corresponds to USERNAME_CHARSET=1.

3.5 MessageStore Specific Options

The following options are specific to the MessageStore and are specified in the MesageStore option file.

DEFAULT_ACL (Access Control List)

This option sets the default Access Control List ACL used when creating top-level, public folders. An ACL is a list of pairs, each pair containing an identifier and a set of rights represented as single letters. The identifier can be anyone to represent all users, group:*groupname* to represent all users in a management group, or a user name to represent just that user. The rights are shown in Table 3–1.

[1] The character sets used for stored messages is controlled through a PMDF CHARSET-CONVERSION mapping table. Consult the *PMDF System Manager's Guide* for details.

Table 3–1 MessageStore ACL Rights

Letter	Rights	Usage
`a`	Administrative	Permits ACLs to be changed
`c`	Create	Permit the creation of sub-folders
`d`	Delete	Permit the deletion of messages and folders
`i`	Insert	Permit messages to be directly appended to a folder
`l`	Lookup	Permit the folder to be seen in listings
`p`	Post	Permit sending e-mail messages to the folder (currently only implemented for `anyone`)
`r`	Read	Permit reading of messages in the folder
`s`	Seen	Permit saving seen information between sessions
`w`	Write	Permit shared IMAP flags to be set

The default setting is

```
DEFAULT_ACL=anyone lrs
```

The value

```
DEFAULT_ACL=anyone lrsp
```

will simplify administration as it allows messages to be delivered directly to all newly created top-level, public folders. You can also want to use a value like

```
DEFAULT_ACL=anyone lrsp manager lrspicdwa
```

to allow the user `manager` to manage all folders. See RFC 2086 for technical details about IMAP access control lists.

FILE_DEBUG (bit encoded integer value)

A value of `-1` sets full debugging on the low-level file operations used by the Message-Store. The debugging output goes to the server thread log file with the IMAP, POP and HTTP servers, to the `channel log` file for the `msgstore` channel, and to the display with the `command-line management` utility.

POST_USER (string)

This option sets the reserved account name which will be used for delivery to public folders. By default, mail addressed to

```
post+folder@host
```

will be delivered to the public folder named `folder` if the `ACL` on the folder has the `p` right for the `anyone` identifier.

4 Web-based Management Interface

This chapter provides a description of both the popstore's web-based management interface as well as the HTTP CGI supporting that interface. The description of the interface itself, Section 4.1, is intentionally brief as the interface is intended to be self-documenting; use the HELP buttons provided in the interface to obtain operating instructions.

Sites which are familiar with the Hypertext Markup Language (HTML), can customize the interface or redesign it entirely. This is where an understanding of the HTTP CGI used by the popstore is needed. See Section 4.3 for details.

4.1 Using the Management Interface

Before attempting to use the web-based management interface, you must first configure the PMDF Dispatcher and HTTP server. This is accomplished by running the Dispatcher configuration utility as described in the *PMDF Installation Guide*. If you have not configured the Dispatcher, then do so now.

In order to use the management interface, you will need a web client which supports HTML tables. The management interface is accessed via the URL

```
http://host:7633/popstore/admin.html
```

In place of `host`, use the actual IP host name of the system running the PMDF HTTP server. If you chose to run the PMDF HTTP server on a port other than port 7633, then specify that port number in place of `7633` in the above URL.

To manage a user domain other than the default user domain, instead specify the URL

```
http://host:7633/popstore/domain/admin.html
```

where `domain` is the name of the user domain to manage.

If you are met with an "Access Forbidden" message when you attempt to access the above URL, then you need to check:

1. that you have configured the PMDF Dispatcher and HTTP server;
2. that you have an HTTP_ACCESS mapping table in your PMDF mapping file;
3. that if you created or changed the HTTP_ACCESS mapping table that you have recompiled your configuration if using a compiled configuration; and,
4. that you have started the PMDF Dispatcher and that it is running.

Note further that if you change the HTTP_ACCESS mapping table, then you will need to restart the HTTP server in order for those changes to take effect.

When you issue the first command which requires access to the popstore itself, your web client will prompt you for your popstore account name and password. You must supply the correct name and password of a popstore account which has popstore management privileges.[1] Management privileges can only be granted to popstore accounts using the command line management utility. See Chapters 6 and 7 for directions on the use of that utility.

To obtain help for a given popstore management function, click on the HELP button associated with that function. A form's RESET button resets the form's entries to their default values.

4.1.1 Restricting Access

Would-be managers of the popstore must be able to supply the correct name and password for a popstore account which has management privileges. You can impose further restrictions through the use of an HTTP_ACCESS mapping table. For instance, you can deny access to the management interface from all but a few IP addresses. See the description of the HTTP_ACCESS mapping table in the *PMDF System Manager's Guide* for directions on how to use the HTTP_ACCESS mapping table.

The web management interface only allows managers to access and manage accounts contained within their own management group as described in Section 1.3.4. Moreover, only a manager with management privileges for the `world` group can create new management groups or modify the structure of existing groups.

4.1.2 Location of the Management Interface

The GIF, HTML, and formatting files which comprise the web-based management interface are located in the `/pmdf/www/popstore/` directory tree on UNIX and NT systems and `PMDF_ROOT:[WWW.POPSTORE]` directory on OpenVMS systems. Sites wanting to customize the interface or develop their own interface can use these files as a starting point. However, do not make changes to the supplied files themselves: *any changes will be lost when you upgrade PMDF.* Instead, make copies of the files and access them via the appropriate URL; *e.g.*,

```
http://host:7633/popstore/x-admin.html
```

Details on developing your own formatting files are given in Section 4.3.

[1] Presently, the popstore CGI uses the HTTP Basic Authentication scheme. This is currently the only standardized HTTP authentication scheme. Unfortunately, with this authentication scheme, the account name and plain text password are transmitted in the clear from the client to the server.

4.2 Using the Password Change Interface

The web-based user interface password change page (see Chapter 5) can also be used by managers to change a user's password. This is accomplished by specifying for authentication the username and password of either the `pmdf` account, or the system account (`root` on Unix, `SYSTEM` on OpenVMS, or `Administrator` on Windows).

Note that if you are modifying the password on behalf of a user, and the username and password are stored in multiple locations (for example, in the PMDF password database as well as in a system account), and there is a failure while changing the password, that the passwords could be left in an inconsistent state (i.e. some locations have the old password and some have the new password). The way to fix this is to keep attempting to change the password until it succeeds.

4.3 The Management CGI

In order to customize the management interface, it is first necessary to understand how the management CGI processes HTTP requests and formulates HTTP responses. This is described in Section 4.3.1 and Section 4.3.2. Following those descriptions, Section 4.3.4 describes the individual commands which can be embedded in those requests.

4.3.1 Processing HTTP Requests

The management CGI only processes HTTP POST requests. GET requests are relayed to a generic processor used by PMDF and can only be used to retrieve entities (*e.g.*, HTML, GIF, *etc.* files) from the /pmdf/www/popstore/ directory tree (UNIX and NT) or `PMDF_ROOT:[WWW.POPSTORE]` directory (OpenVMS).

The CGI interface responds to HTTP POST requests by parsing the request for a management command and generating the appropriate response. The content of the POST request is the management command and takes the general form

```
command=command-name&parameter-name-1=parameter-value-1&
   ...&parameter-name-N=parameter-value-N
```

(The above line has been wrapped for typographic reasons.) The ellipses, `...`, in the above indicate additional `parameter-name=parameter-value` pairs which might appear in the command. The allowed command names and associated parameters are described in Section 4.3.4. The commands are case-insensitive: the command and parameter names can be specified in either upper, lower, or mixed case.

If the command cannot be extracted from the request, an HTTP 5yz error response is sent back to the client. If the command can be extracted but cannot be parsed or successfully executed, a successful HTTP 200 response is sent back; the content of the HTTP response will be formatted as per the error formatting directions specified in the management command. If those directions could not be extracted from the command, then an HTTP 500 error response is returned. See Section 4.3.2 for further details.

4.3.2 Generating HTTP Responses

After processing an HTTP request from a client, the result of processing the command is sent back as an HTTP response to the client. The format of the response is governed by formatting files specified in the command from the client. That is, the request from the client includes the names of formatting files on the CGI server system that are to be used to format the response sent back to the client. On OpenVMS systems, these files must reside in the `PMDF_ROOT:[WWW.POPSTORE]` directory; on UNIX and NT systems, these files must reside in the `/pmdf/www/popstore/` directory tree.

The formatting files can contain text to be copied verbatim into the HTTP response as well as directives to substitute in values associated with the results of a particular command. There are three basic types of formatting files: success, error, and command-specific files.

After the CGI parses a request and executes it, the results of the operation are sent back to the HTTP client using the following formatting steps:

1. The command-specific formatting files are used to format the data collected by the command. The files can be consulted multiple times such as when generating a list or table of information (*e.g.*, account listings).
2. If the preceeding step is successful, the content of the success formatting file is used as the response to the client. Each line of the file is copied to the content of the HTTP response. Any line beginning with `%s` is replaced with the formatted data generated in the first step. The HTTP response sent back to the client will have an HTTP 200 status code.
3. If the first step failed, the content of the error formatting file is sent as the response to the client. Each line of the file is copied to the content of the HTTP response. Any line beginning with `%s` is replaced with the output, if any, of the first step as well as any error messages. The HTTP response sent back to the client will have an HTTP 200 status code.

In the command-specific formatting files, command-specific substitution strings can appear. These strings all begin with the percent character, `%`. When such a string is encountered, the value it references is substituted into the HTTP response. For instance, the formatting file

```
%none{No users found matching your search criterion}
%first{<TABLE><TR><TH>Username<TH>Messages}
<TR><TD>%username<TD>%message_count
%last{</TABLE>}
```

might be used in conjunction with the `list_users` command to produce an HTML table of the popstore users and how many messages each have stored; *e.g.*,

```
<TABLE><TR><TH>Username<TH>Messages
<TR><TD>asmith<TD>11
<TR><TD>jdoe<TD>7
<TR><TD>mfreed<TD>8
<TR><TD>wrobinson<TD>12
<TR><TD>zsmith<TD>0
</TABLE>
```

In the tables describing each substitution string, the type of data associated with the substitution string is stated. These types are:

Type	Description
int	Signed integer
float	Single precision floating point number
string	ASCII text string
uint	Unsigned integer

In addition, the default formatting string used to format the data is also shown in the tables. The formatting strings follow the C programming language convention for formatting strings passed to the sprintf() C run-time library subroutine. Alternate formatting strings can be used by enclosing them in braces, {}, and appending them to the substitution string. For instance,

```
%quota_used{%.1f} Kbyte%s
```

might be used to limit the `%quota_used` to a single digit of precision after the decimal point.

Five substitution strings, `%first`, `%last`, `%!first`, `%!last`, and `%none`, deserve special attention. These first four strings substitute into the output specific text when formatting, respectively, the first, the last, not the first, or not the last instance of the data to be formatted. The text to be substituted in must be enclosed in braces, {}, following the substitution string.[3] For example, suppose that a list of popstore users is to be formatted using the following formatting file:

```
%first{<TABLE>}
<TD>%username
%last{</TABLE>}
```

In the above, when information for the first popstore account is formatted, the text `<TABLE>` will be output followed by the username (`%username`). When information for the last account is formatted, the text `</TABLE>` will be output following the username.

The `%none` substitution string supplies text to output when there are no instances of the collected data to display. For example, when there are no messages to list for a given user account.

[3] At present, substitution strings appearing within the text to be substituted are ignored and treated as literal text.

4.3.3 An Example

Interested readers should consult the file `admin_cmd.html` which can be found in the /pmdf/www/popstore/ directory on UNIX and NT systems, and on OpenVMS systems in the `PMDF_ROOT:[WWW.POPSTORE]` directory. That file is the main management page for the popstore's web-based management interface. As such, the file shows most all of the commands described in Section 4.3.4 in use. Familiarity with HTML, especially the HTML <FORM> tag and its related tags, is helpful in understanding the contents of the `admin.html` file.

4.3.4 Management Commands

As described in Section 4.3.1, management commands take the general form

```
command=command-name&parameter-name-1=parameter-value-1&
  ...&parameter-name-N=parameter-value-N
```

In the above, *command-name* gives the name of the command to execute. It is then followed by two or more parameters which provide supplemental information relevant to the operation to be performed. When parameter names are duplicated in the command, only the right most instance of the parameter=value pair is honored. Two exceptions to this are the `username` and `flag` parameters which for many commands can be repeated with significance.

For those parameters whose values are file specifications, the file specifications must be relative file paths specifying files in the /pmdf/www/popstore/ directory tree on UNIX and NT systems, or the directory `PMDF_ROOT:[WWW.POPSTORE]` on OpenVMS systems.

The valid command names are listed in the table below and described in the following sections.

Command name	Section	Description
`add_group`	4.3.4.1	Add a new management group
`add_user`	4.3.4.2	Add a new user account
`copy_user`	4.3.4.3	Use an existing user account as the basis for a new user account
`delete`	4.3.4.5	Synonym for `delete_message`
`delete_group`	4.3.4.4	Delete a management group
`delete_message`	4.3.4.5	Delete one or more messages for a user account
`delete_messages`	4.3.4.6	Delete all stored messages for a user account
`delete_user`	4.3.4.7	Remove a user account
`forward`	4.3.4.8	Establish a forwarding address
`list_forward`	4.3.4.9	List forwarding addresses
`list_groups`	4.3.4.10	List management groups
`list_users`	4.3.4.11	List user accounts

Command name	Section	Description
`modify_group`	4.3.4.12	Modify a management group definition
`modify_user`	4.3.4.13	Modify an existing user account
`rename_user`	4.3.4.14	Change the username associated with a user account
`show`	4.3.4.16	Synonym for `show_message`
`show_counters`	4.3.4.15	Show channel counters for the popstore or other PMDF channels
`show_message`	4.3.4.16	Show a stored message
`show_user`	4.3.4.17	Show settings for a user account
`unforward`	4.3.4.18	Remove a forwarding address

4.3.4.1 `add_group` Command: add a new management group

The `add_group` command adds a new management group to the popstore. Parameter names and associated values accepted by the command are listed in the Table 4–1.

Table 4–1 `add_group` command parameters

`parameter=value`		Description
`group=name`	Required	Name of the management group, `name`, to create. The name can be one to sixteen bytes long.
`group_blist=blist`	Optional	Byte delimited list of subgroups to be contained within the new group. The first byte in `blist` is the delimiter which is then used to delimit each group name in the string; *e.g.,* `|name-1|name-2|name-3`. The length of the string `blist` can not exceed 236 bytes.
`group_list=list`	Optional	Comma delimited list of subgroups to be contained within the new group. For instance, `name-1,name-2,name-3`. Leading and trailing white space around each group name is ignored. The length of the string `blist` can not exceed 236 bytes.
`log=bvalue`	Optional	Boolean value, `0` or `1`. When `bvalue` is `1`, then a status message will be output indicating a successful operation. The default is `bvalue=0`. Note that errors are always reported.
`on_error=file-spec`	Required	Name of the formatting file to use to format the results when the command fails.
`on_success=file-spec`	Required	Name of the formatting file to use to format the results when the command succeeds.

When creating a new management group, at most one of `group_list` or `group_blist` should be specified. If neither are specified, then the new group will contain no subgroups. If the specified group name conflicts with an existing group name, then an error will be issued and the pre-existing group definition left unchanged.

4.3.4.2 `add_user` Command: add a new user account

The `add_user` command adds one or more user accounts to the popstore. Parameter names and associated values accepted by the command are listed in the Table 4–2.

Table 4–2 `add_user` command parameters

`parameter=value`		Description

`flag=fvalue`
`flag0=fvalue`
`flag1=fvalue`
`flag2=fvalue`
`flag3=fvalue`
`flag4=fvalue`
	Optional	Usage flags for the account. `fvalue` must be one of `dismail`, `nodismail`, `disuser`, `nodisuser`, `lockpwd`, or `nolockpwd`. The `flag0`, `flag1`, ... parameters are provided for use in instances in HTML where the same parameter name cannot be used more than once in the same form.

`group_name=name`	Optional	Name of the management group, `name`, to place the new account in. The account must be placed into a group to which the manager creating the account has access to. When a group name is not specified for the new account, a zero length group name is assumed thus placing the new account into the `world` group.
`log=bvalue`	Optional	Boolean value, `0` or `1`. When `bvalue` is `1`, then a status message will be output indicating a successful operation. The default is `bvalue=0`. Note that errors are always reported.
`on_error=file-spec`	Required	Name of the formatting file to use to format the results when the command fails.
`on_success=file-spec`	Required	Name of the formatting file to use to format the results when the command succeeds.
`overdraft=qvalue`	Optional	Amount of storage overdraft to allow the new account. `qvalue` is a non-negative, floating point value.
`overdraft_units=qunits`	Optional	Storage units in which the overdraft quota `qvalue` is expressed. Must be one of `b` (bytes), `kb` (kilobytes), `mb` (megabytes), or `gb` (gigabytes). If this parameter is not specified, storage units of `kb` are assumed.
`owner=string`	Optional	Name of the owner of the account. This string has a maximum length of 40 bytes.
`password=string`	Required	The account password. This string has a maximum length of 32 bytes. A zero-length password allows anyone to access the account.
`private=string`	Optional	Data to store in the account's site-defined private data area. This string has a maximum length of 64 bytes.
`quota=qvalue`	Optional	Primary message storage quota for the account. A `qvalue` of zero grants the account unlimited storage quota. If the `quota_units` parameter is not specified, then units of kilobytes, `kb`, are assumed. `qvalue` is a non-negative, floating point value.
`quota_units=qunits`	Optional	Storage units in which the quota `qvalue` is expressed. Must be one of `b` (bytes), `kb` (kilobytes), `mb` (megabytes), or `gb` (gigabytes).

Table 4–2 (Cont.) `add_user` **command parameters**

`parameter=value`		**Description**
`username=string`	Required	Username to associate with the account. This string has a maximum length of 32 bytes and can not contain wild card characters.

When creating a new account, values for omitted, optional parameters will be set to "zero": numeric values will be set to zero and string values will have a zero length.

If the `username` parameter is repeated, then multiple accounts are created, one with each `username` value. Each account so created will have the same password, flags, primary and overdraft quotas, *etc*.

As an example, consider the command

```
command=add_user&username=jdoe&password=SeCrEt&owner=Jane Doe&
  success_format=add_success.txt&error_format=add_error.txt
```

which creates the new account `jdoe` with password `SeCrEt` and owner field `Jane Doe`.

To set more than one usage flag, repeat the `flag` parameter. For instance,

```
command=add_user&username=jdoe&password=SeCrEt&owner=Jane Doe&
  flag=nodismail&flag=nodisuser&flag=lockpwd&
  success_format=add_success.txt&error_format=add_error.txt
```

which sets the account's usage flags to `nodismail`, `nodisuser`, and `lockpwd`.

4.3.4.3 `copy_user` Command: copy an existing user account

The `copy_user` command creates a new account whose settings duplicate those of an existing account. Note that accounting information and stored messages are not copied.

The parameter names and associated values accepted by the command are listed in Table 4–3.

Table 4–3 `copy_user` **command parameters**

`parameter=value`		**Description**

`flag=fvalue`
`flag0=fvalue`
`flag1=fvalue`
`flag2=fvalue`
`flag3=fvalue`
`flag4=fvalue`
	Optional	Usage flags for the new account. `fvalue` must be one of `dismail`, `nodismail`, `disuser`, `nodisuser`, `lockpwd`, or `nolockpwd`. The `flag0`, `flag1`, ... parameters are provided for use in instances in HTML where the same parameter name cannot be used more than once in the same form.

`log=bvalue`	Optional	Boolean value, `0` or `1`. When `bvalue` is `1`, then a status message will be output indicating a successful operation. The default is `bvalue=0`. Note that errors are always reported.

Table 4–3 (Cont.) `copy_user` **command parameters**

parameter=value		Description
group_name=name	Optional	Name of the management group, `name`, to place the new account in. The account must be placed into a group to which the manager creating the account has access to. When a group name is not specified for the new account, the group name of the account being copied is assumed.
on_error=file-spec	Required	Name of the formatting file to use to format the results when the command fails.
on_success=file-spec	Required	Name of the formatting file to use to format the results when the command succeeds.
overdraft=qvalue	Optional	Amount of storage overdraft to allow the new account. `qvalue` is a non-negative, floating point number.
overdraft_units=qunits	Optional	Storage units in which the overdraft quota `qvalue` is expressed. Must be one of `b` (bytes), `kb` (kilobytes), `mb` (megabytes), or `gb` (gigabytes). If this parameter is not specified, storage units of `kb` are assumed.
owner=string	Optional	Name of the owner of the new account. This string has a maximum length of 40 bytes.
password=string	Optional	The password for the new account. This string has a maximum length of 32 bytes. A zero-length password allows anyone to access the account.
private=string	Optional	Data to store in the new account's site-defined private data area. This string has a maximum length of 64 bytes.
quota=qvalue	Optional	Primary message storage quota for the new account. A `qvalue` of zero grants the account unlimited storage quota. If the `quota_units` parameter is not specified, then units of kilobytes, `kb`, are assumed. `qvalue` is a non-negative, floating point number.
quota_units=qunits	Optional	Storage units in which the quota `qvalue` is expressed. Must be one of `b` (bytes), `kb` (kilobytes), `mb` (megabytes), or `gb` (gigabytes).
username=string	Required	First instance of this parameter gives the name of the existing account to copy. Subsequent instances give the names of the new accounts to create. These strings each have a maximum length of 32 bytes and can not contain wild card characters.

The `username` parameter must appear at least twice. The first instance, when reading the command from left to right, gives the name of the account to copy. The second and subsequent instances give the names of the accounts to create. Values for account fields not specified in the command will be drawn from the account being copied.

As an example, consider the command

```
command=copy_user&username=default&username=jdoe&
  password=SeCrEt&owner=Jane Doe&success_format=copy_success.txt&
  error_format=copy_error.txt
```

which creates a new account `jdoe` which is a copy of the account `default`. The new account is given the password `SeCrEt` and owner field `Jane Doe`. All other fields duplicate those of the `default` account.

As an aside, note that the `add_user` command is actually implemented as a `copy_user` command with the `default` account being the account which is copied.

4.3.4.4 `delete_group` Command: delete a management group

Use the `delete_group` command to delete a management group definition. Subgroups contained within the group are only deleted when `recur=1` is explicitly specified. Note that any accounts contained in the group are not deleted; only the group definitions are deleted. Parameter names and associated values accepted by the command are listed in the Table 4–4.

Table 4–4 `delete_group` **command parameters**

`parameter=value`		**Description**
`group=name`	Required	Name of the management group, `name`, to delete. The name can be one to sixteen bytes long.
`log=bvalue`	Optional	Boolean value, `0` or `1`. When `bvalue` is `1`, then a status message will be output indicating a successful operation. The default is `bvalue=0`. Note that errors are always reported.
`on_error=file-spec`	Required	Name of the formatting file to use to format the results when the command fails.
`on_success=file-spec`	Required	Name of the formatting file to use to format the results when the command succeeds.
`recur=bvalue`	Optional	Boolean value, `0` or `1`. When `bvalue` is `1`, the definitions of all subgroups contained within the group are also deleted. When `bvalue=0`, the default, only the group definition of `name` is deleted.

4.3.4.5 `delete_message` Command: delete a user's message

A specific message stored for a user account can be deleted with the `delete_message` command. The message to be deleted is identified by means of a UIDL as returned by the `%msgr_uidl` substitution string of the `show_users` command.

If a `return_messages` parameter with a `bvalue` of `1` is specified, then a non-delivery notice will be sent to the message's originator if the message is unread. Note that if the message has multiple recipients, those recipient's copies of the message are not affected by this command.

Parameter names and associated values accepted by the command are listed in Table 4–5.

Table 4–5 `delete_message` **command parameters**

`parameter=value`		**Description**
`on_error=file-spec`	Required	Name of the formatting file to use to format the results when the command fails.
`on_success=file-spec`	Required	Name of the formatting file to use to format the results when the command succeeds.
`log=bvalue`	Optional	Boolean value, `0` or `1`. When `bvalue` is `1`, then a status message will be output indicating a successful operation. The default is `bvalue=0`. Note that errors are always reported.
`return_messages=bvalue`	Optional	Boolean value, `0` or `1`, indicating whether (`1`) or not (`0`) a non-delivery notice should be sent if the message has not been read by the user. By default, a non-delivery notice is not sent; this corresponds to a `bvalue` of `0`. When `bvalue` is `1`, a non-delivery notice is sent should the message prove to be unread.
`uidl=string`	Required	UIDL for this user's instance of the message to be deleted.
`username=string`	Required	Username of the account for which to delete the stored messages. This string has a maximum length of 32 bytes and can not contain wild card characters.

Only a single instance of the `username` and `uidl` parameters can appear.

As an example, consider the following command which deletes the message with UIDL `!!!!01234` for the account `sue`:

```
command=delete_message&username=sue&uidl=!!!!01234&
  return_messages=1&on_success=delmsg_success.txt&
  on_error=delmsg_error.txt
```

As `return_messages=1` is specified, a non-delivery notice is sent for each unread message which is deleted.

4.3.4.6 `delete_messages` Command: delete a user's messages

Messages stored for a popstore account can be deleted with the `delete_messages` command. If a `return_messages` parameter with a `bvalue` of `1` is specified, then a non-delivery notice will be sent for unread messages. Parameter names and associated values accepted by the command are listed in Table 4–6. Note that if the messages have other recipients, those other recipients' copies of the messages are not effected by this command.

Table 4–6 `delete_messages` **command parameters**

`parameter=value`		**Description**
`group=name`	Optional	Delete the messages stored for the matching accounts contained within the specified management group and subgroups thereof. The name of the group, `name`, can not contain wild card characters. This parameter should only be used when then value specified with the `username` parameter contains wild cards.
`log=bvalue`	Optional	Boolean value, `0` or `1`. When `bvalue` is `1`, then a status message will be output indicating a successful operation. The default is `bvalue=0`. Note that errors are always reported.
`on_error=file-spec`	Required	Name of the formatting file to use to format the results when the command fails.
`on_success=file-spec`	Required	Name of the formatting file to use to format the results when the command succeeds.
`return_messages=bvalue`	Optional	Boolean value, `0` or `1`, indicating whether (`1`) or not (`0`) non-delivery notices should be sent for unread messages. By default, non-delivery notices are not sent; this corresponds to a `bvalue` of `0`. When `bvalue` is `1`, non-delivery notices are sent for unread messages.
`username=string`	Required	Username of the account for which to delete the stored messages. This string has a maximum length of 32 bytes and can contain wild card characters.

When multiple instances of the `username` parameter are provided, messages for each of the specified accounts are deleted. When the value specified with a `username` parameter contains wild card characters, the operation will be confined to any matching user name within the management group and subgroups of the manager requesting the operation. If a group name is also specified, then the operation will be further confined to that group and subgroups.

As an example, consider the following command which deletes the messages for the accounts `sue` and `tom`:

```
command=delete_messages&username=sue&username=tom&return_messages=1&
  on_success=delmsgs_success.txt&on_error=delmsgs_error.txt
```

As `return=1` is specified, non-delivery notices are sent for each unread message which is deleted.

4.3.4.7 `delete_user` Command: delete a user account

Popstore accounts can be deleted with the `delete_user` command. Parameter names and associated values accepted by the command are listed in Table 4–7.

Table 4–7 `delete_user` **command parameters**

`parameter=value`		**Description**
`group=name`	Optional	Delete the matching accounts contained within the specified management group and subgroups thereof. The name of the group, `name`, can not contain wild card characters. This parameter should only be used when then value specified with the `username` parameter contains wild cards.
`log=bvalue`	Optional	Boolean value, `0` or `1`. When `bvalue` is `1`, then a status message will be output indicating a successful operation. The default is `bvalue=0`. Note that errors are always reported.
`on_error=file-spec`	Required	Name of the formatting file to use to format the results when the command fails.
`on_success=file-spec`	Required	Name of the formatting file to use to format the results when the command succeeds.
`return_messages=bvalue`	Optional	Boolean value, `0` or `1`, indicating whether (`1`) or not (`0`) unread messages stored for the account should be returned to their originator. By default, unread messages are not returned; this corresponds to a `bvalue` of `0`. A `bvalue` of `1` requests that unread messages be returned.
`username=string`	Required	Username associated with the account to delete. This string has a maximum length of 32 bytes and can contain wild card characters.

When multiple instances of the `username` parameter are provided, each of the associated accounts are deleted. When the value specified with a `username` parameter contains wild card characters, the operation will be confined to any matching user names within the management group and subgroups of the manager requesting the operation. If a group name is also specified, then the operation will be further confined to that group and subgroups.

As an example, consider the following command which deletes the accounts `sue` and `tom`:

```
command=delete_user&username=sue&username=tom&
  on_success=delusr_success.txt&on_error=delusr_error.txt
```

Since the `return_messages` parameter is not specified, any unread messages will be deleted.

4.3.4.8 `forward` Command: establish a forwarding address

The `forward` command establishes a forwarding address for a given name in the popstore's addressing name space. The name need not correspond to an existing popstore account. Parameter names and associated values accepted by the command are listed in Table 4–8.

Table 4–8 `forward` **command parameters**

`parameter=value`		**Description**
`fwd_from=username`	Required	Username for which to establish the forwarding. Maximum string length is 32 bytes.
`fwd_override=bvalue`	Optional	Boolean value, `0` or `1`, indicating whether or not to establish a forwarding for an existing popstore account. If `bvalue` is `0`, then the forwarding will be disallowed if the `fwd_from` username corresponds to an existing account. Otherwise, the forwarding is permitted. When this parameter is omitted, a `bvalue` of `1` is assumed.
`fwd_to=address`	Required	Single RFC822 address to which to forward messages. The address can be either another popstore address or an address external to the popstore. Maximum length of the string is 252 bytes.
`on_error=file-spec`	Required	Name of the formatting file to use to format the results when the command fails.
`on_success=file-spec`	Required	Name of the formatting file to use to format the results when the command succeeds.

As an example, consider establishing a forwarding address for the existing user account `jdoe` to the RFC822 address `Jane.Doe@example.com`:

```
command=forward&fwd_from=jdoe&fwd_to=jane.doe@example.com&
  on_success=fwd_success.txt&on_error=fwd_error.txt
```

4.3.4.9 `list_forward` Command: list forwarding addresses

The `list_forward` command is used to list popstore forwardings. The parameter names and associated values accepted by the command are listed in Table 4–9.

Table 4–9 `list_forward` **command parameters**

`parameter=value`		**Description**
`on_error=file-spec`	Required	Name of the formatting file to use to format the results when the command fails.
`on_success=file-spec`	Required	Name of the formatting file to use to format the results when the command succeeds.
`pformat=file-spec`	Required	Name of the formatting file to use to format each forwarding address. The recognized substitution strings for this formatting file are listed in Tables 4–10 and 4–11.

Table 4–9 (Cont.) `list_forward` **command parameters**

parameter=value		Description
`rooted=bvalue`	Optional	Boolean value, `0` or `1`, indicating whether or not an exact lookup is performed. When `bvalue` is `0`, only the entry, if any, exactly matching the username string is returned. When `bvalue` is `1`, then the username string is treated as a prefix and any forwarding beginning with the supplied username string will be returned. When not specified, a `bvalue` of `0` is assumed.
`username=string`	Optional	Username for which to list the forwarding. If omitted, all forwardings are listed. Maximum length of `string` is 32 bytes. The string can not contain wild cards.

The `username` parameter can appear in the command at most once. If the `username` parameter is omitted and `rooted=1` is specified, then all forwardings will be returned. Note that username lookups are done in a case-insensitive manner.

An example command to list all forwardings starting with the letter `d` is shown below:

```
command=list_forward&username=d&rooted=1&pformat=lfwd.txt&
  on_success=lfwd_success.txt&on_error=lfwd_error.txt
```

A sample formatting file is shown in Example 4–1. This is the formatting file used by the interactive, command line management utility.

Example 4–1 `list_forward` **formatting file**

```
%none{No forwardings exist}
%first{Username                          Forwarding address}
%first{---------------------------------------------------}
%fwd_from{%-32s}  %fwd_to
```

Table 4–10 General substitution strings

Substitution string	Type	Format	Description
`%last`	string	`%s`	Text string to display if this is the last formatting pass.
`%none`	string	`%s`	Text string to display if there is no information to format.
`%first`	string	`%s`	Text string to display if this is the first formatting pass.
`%!first`	string	`%s`	Text string to display if this is not the first formatting pass.
`%!last`	string	`%s`	Text string to display if this is not the last formatting pass.

Table 4–10 (Cont.) General substitution strings

Substitution string	Type	Format	Description
`%S`	string	`S`	Output an uppercase letter "S" if the previously displayed numeric value had a non-singular value. See the text for a description of how to output different strings depending upon whether or not the last numeric value was non-singular.
`%s`	string	`s`	Output a lowercase letter "s" if the previously displayed numeric value had a non-singular value. See the text for a description of how to output different strings depending upon whether or not the last numeric value was non-singular.
`%host`	string	`%s`	Display the TCP/IP domain name of the CGI server host on which the popstore information was collected.
`%image_ident`	string	`%s`	PMDF version number as recorded in the PMDF shared library.
`%image_link_date`	string	`%s`	Date and time the PMDF shared library was linked.
`%time`	string	`%s`	Display the date and time at which the data was collected.

The formatting strings for the `%s` and `%S` substitution strings work differently from most substitution strings. The formatting strings specify the text to display when the previous numeric value was non-singular. If the formatting text contains a vertical bar, |, then the text to the left of the vertical bar is displayed when the previous numeric value is singular while the text to the right of the vertical bar is displayed when the previous numeric value is non-singular. For instance, the formatting instructions

```
Repl%s{y|ies}
```

generate the output `Reply` when the previously displayed numeric value is singular and the `Replies` otherwise.

Table 4–11 `list_forward` command substitution strings

Substitution string	Type	Format	Description
`%fwd_from`	string	`%s`	Username being forwarded.
`%fwd_to`	string	`%s`	Address to which the username is being forwarded.

4.3.4.10 `list_groups` Command: list management groups

Listings of management group definitions are produced with the `list_groups` command. Parameter names and associated values accepted by the command are listed in the Table 4–12.

Table 4–12 `list_groups` command parameters

parameter=value		Description
`group=name`	Optional	Name of the management groups, `name`, to list. The name can be one to sixteen bytes long and can contain wild card characters.
`on_error=file-spec`	Required	Name of the formatting file to use to format the results when the command fails.
`on_success=file-spec`	Required	Name of the formatting file to use to format the results when the command succeeds.
`pformat=file-spec`	Required	Name of the formatting file to use to format each group definition. The recognized substitution strings for this formatting file are listed in Tables 4–10 and 4–13.

Table 4–13 `list_groups` command substitution strings

Substitution string	Type	Format	Description
`%group`	string	`%s`	Group name.
`%group_blist`	string	`%s`	Byte delimited representation of the group's subgroups, if any. If the group contains no subgroups, then a zero length string is substituted.
`%group_delimiter`	string	`%s`	Delimiter character used in the byte delimited representation.
`%group_list`	string	`%s`	Comma delimited representation of the group's subgroups, if any. If the group contains no subgroups, then a zero length string is substituted.

4.3.4.11 `list_users` Command: list user accounts

The `list_users` command is used to list accounts. Note that only accounts contained within the manager's management group will be listed. The parameter names and associated values accepted by the command are listed in Table 4–14.

Table 4–14 `list_users` command parameters

parameter=value		Description
`group=name`	Optional	Restrict the listing to only accounts contained in the specified group, `name`. When not specified, the group associated with the manager generating the listing is assumed. Wild cards are not permitted.

Table 4–14 (Cont.) `list_users` **command parameters**

`parameter=value`		**Description**
`on_error=file-spec`	Required	Name of the formatting file to use to format the results when the command fails.
`on_success=file-spec`	Required	Name of the formatting file to use to format the results when the command succeeds.
`pformat=file-spec`	Required	Name of the formatting file to use to format each account listing. The recognized substitution strings for this formatting file are listed in Tables 4–10 and 4–15.
`rooted=bvalue`	Optional	Boolean value, `0` or `1`, indicating whether or not an exact lookup is performed. When `bvalue` is `0`, only the entry, if any, exactly matching the username string is returned. When `bvalue` is `1`, then the username string is treated as a prefix and any forwarding beginning with the supplied username string will be returned. When not specified, a `bvalue` of `0` is assumed.
`username=string`	Optional	List accounts whose username match the pattern specified by `string`. `string` can contain wild cards. If this parameter is omitted, then `*` is assumed and a listing of all accounts is generated.

The `username` parameter can appear in the command at most once. If the `username` parameter is omitted and `rooted=1` is specified, then all accounts will be returned. Note that username lookups are done in a case-insensitive manner.

An example command to list all accounts starting with the letter `d` is shown below:

```
command=list_users&username=d&rooted=1&pformat=lusr.txt&
  on_success=lusr_success.txt&on_error=lusr_error.txt
```

An example formatting file is shown in Example 4–2. This is the formatting file used by the interactive, command line management interface.

Example 4–2 `list_users` **formatting file**

```
%first{                                          Quota  Message  Quota used}
%first{ Username                              (kbytes)    Count    (kbytes)}
%first{ --------------------------------------------------------------------}
%flags_manage{ |*}%username{%-32s} %quota_k{%10.2f}  %message_count{%7u}    %quota_used_k{%8.2f}
%last{ --------------------------------------------------------------------}
%last{*Note: privileged users are flagged with an asterisk}
```

Table 4–15 `list_users` **command substitution strings**

Substitution string	**Type**	**Format**	**Description**
`%filename`	string	`%-s`	Full file specification for the profile file representing the account.

Table 4–15 (Cont.) `list_users` **command substitution strings**

Substitution string	Type	Format	Description
`%flags`	string	`%s`	Comma separated string representation of the account's usage flags. Built from the words `DELETE`, `DISUSER`, `DISMAIL`, `LOCKPWD`, `MANAGE`, `MIGRATED`, and `PWD_ELSEWHERE`.
`%flags_delete`	string	`Disabled|Enabled`	See text.
`%flags_dismail`	string	`Disabled|Enabled`	See text.
`%flags_disuser`	string	`Disabled|Enabled`	See text.
`%flags_lockpasswd`	string	`Disabled|Enabled`	See text.
`%flags_manage`	string	`Disabled|Enabled`	See text.
`%flags_migrated`	string	`Disabled|Enabled`	See text.
`%flags_pwd_elsewhere`	string	`Disabled|Enabled`	See text.
`%glen`	uint	`%u`	Length in bytes of the contents of the group_name field.
`%group_name`	uint	`%s`	Contents of the group_name field.
`%last_billing`	string	`%s`	Date and time when the account was last billed. This field is initialized to the creation date and time for the acount after which it is subsequently set by site-supplied account procedures.
`%last_connect`	string	`%s`	Date and time of last connect (*i.e.,* date and time when the user last connected to the POP3 server with their POP3 client).
`%last_disconnect`	string	`%s`	Date and time of last disconnect (*i.e.,* date and time when the user last disconnected from the POP3 server with their POP3 client).
`%last_pwd_change`	string	`%s`	Date and time that this user's password was last changed.
`%message_count`	uint	`%u`	Count of stored messages.
`%olen`	uint	`%u`	Length in bytes of the contents of the owner field.
`%overdraft_b`	uint	`%u`	Message overdraft quota in units of bytes.
`%overdraft_k` `%overdraft_m` `%overdraft_g`	float	`%.2f`	Message overdraft quota in units of, respectively, kbytes (`_k`), mbytes (`_m`), or gbytes (`_g`).
`%owner`	string	`%s`	Contents of the owner field.
`%past_block_days`	uint	`%u`	Accumulated message storage for past (*i.e.,* deleted) messages as measured in units of block days.

Table 4–15 (Cont.) `list_users` **command substitution strings**

Substitution string	Type	Format	Description
`%past_block_days_remainder`	uint	`%u`	Accumulated round off of the past_ block_days field as measured in units of byte minutes.
`%private`	string	`%s`	Contents of the site-specific private data storage field.
`%quota_b`	uint	`%u`	Primary message storage quota in units of bytes.
`%quota_k` `%quota_m` `%quota_g`	float	`%.2f`	Primary message storage quota in units of, respectively, kbytes (`_k`), mbytes (`_m`), or gbytes (`_g`).
`%quota_used_b`	uint	`%u`	Storage in bytes consumed by messages currently stored for the account.
`%quota_used_g` `%quota_used_k` `%quota_used_m`	float	`%.2f`	Storage in kbytes (`_k`), mbytes (`_m`), or gbytes (`_g`) consumed by messages currently stored for the account.
`%received_bytes`	uint	`%u`	Cumulative count of message bytes stored for the account.
`%received_messages`	uint	`%u`	Cumulative count of messages stored for the account.
`%slen`	uint	`%u`	Length in bytes of the contents of the private field.
`%store`	uint	`popstore|MessageStore|native`	See text.
`%total_connect_s`	uint	`%u`	Total elapsed connect time expressed in units of seconds.
`%total_connect_m` `%total_connect_h` `%total_connect_d`	float	`%.2f`	Total elapsed connect time expressed, respectively, in units of minutes (`_m`), hours (`_h`), or days (`_d`).
`%total_connect_dhms`	string	`%s`	Total, elapsed connect time expressed in a format showing days, hours, minutes, and seconds: dd hh:mm:ss.
`%total_connections`	uint	`%u`	Total number of connections made to the account.
`%ulen`	uint	`%u`	Length in bytes of the contents of the username field.
`%username`	string	`%s`	Username field.
`%version`	uint	`%u`	Profile file version format field.

The formatting fields for the `%store` and `%flags_` substitution strings work differently than other substitution strings. These fields are interpreted, respectively, as three and two strings separated by a vertical bar, `|`. In the case of the `%store` substitution string, the first string is substituted when the account is a popstore account, the second when it is a MessageStore account, and the third when it is a profile file marked as being native. In the case of the `%flags_` substitution strings, the first string is that substituted when the associated flag is not set and the second string that when the field is set. For instance, the formatting instructions

```
          Management privileges: %flags_manage{Disabled|Enabled}
```

produce the output

```
          Management privileges: Disabled
```

for an account which does not have the MANAGE usage flag set. For an account with the MANAGE usage flag set, the output would instead be

```
          Management privileges: Enabled
```

4.3.4.12 `modify_group` Command: modify a management group definition

The `modify_group` command modifies a management group definition. The only reason to modify a management group definition is to add, change, or remove the list of subgroups contained by the group.

Parameter names and associated values accepted by the command are listed in the Table 4–16.

Table 4–16 `modify_group` **command parameters**

`parameter=value`		**Description**
`group=name`	Required	Name of the management group, `name`, to modify. The name can be one to sixteen bytes long.
`group_blist=blist`	Optional	Byte delimited list of subgroups to be contained within the group. The first byte in `blist` is the delimiter which is then used to delimit each group name in the string; *e.g.,* `|name-1|name-2|name-3`. The length of the string `blist` can not exceed 236 bytes.
`group_list=list`	Optional	Comma delimited list of subgroups to be contained within the group. For instance, `name-1,name-2,name-3`. Leading and trailing white space around each group name is ignored. The length of the string `blist` can not exceed 236 bytes.
`log=bvalue`	Optional	Boolean value, `0` or `1`. When `bvalue` is `1`, then a status message will be output indicating a successful operation. The default is `bvalue=0`. Note that errors are always reported.
`on_error=file-spec`	Required	Name of the formatting file to use to format the results when the command fails.
`on_success=file-spec`	Required	Name of the formatting file to use to format the results when the command succeeds.

When modifying a management group, at most one of `group_list` or `group_blist` should be specified. If neither are specified, then the group is modified such that it contains no subgroups.

4.3.4.13 modify_user Command: modify a user account

The modify_user command changes settings for one or more user accounts. Parameter names and associated values accepted by the command are listed in Table 4–17.

Table 4–17 modify_user command parameters

parameter=value		Description

flag=fvalue
flag0=fvalue
flag1=fvalue
flag2=fvalue
flag3=fvalue
flag4=fvalue
flag5=fvalue
	Optional	Usage flags for the account. fvalue must be one of pop, imap, dismail, nodismail, disuser, nodisuser, lockpwd, nolockpwd, migrated, nomigrated, pwd_elsewhere, or nopwd_elsewhere. The flag0, flag1, ... parameters are provided for use in instances in HTML where the same parameter name cannot be used more than once in the same form.

group=name	Optional	Restrict the modifications to accounts contained within the specified management group and subgroups thereof. The group name, name, can not contain wild card characters. This parameter should only be used when the value supplied with the username parameter contains wild cards.
group_name=name	Optional	Name of the management group, name, to move the account to. The account can only be moved to a group to which the manager modifying the account has access to.
last_connect=0	Optional	Resets the recorded last connect time to indicate that no connection has yet been made.
last_disconnect=0	Optional	Resets the recorded last connect time to indicate that no connection has yet been made.
log=bvalue	Optional	Boolean value, 0 or 1. When bvalue is 1, then a status message will be output indicating a successful operation. The default is bvalue=0. Note that errors are always reported.
message_count=count	Optional	Set the account's stored message count to the specified value, count. If the account's current stored message count, c_count, exceeds this value, then the oldest c_count - count messages are deleted.
on_error=file-spec	Required	Name of the formatting file to use to format the results when the command fails.
on_success=file-spec	Required	Name of the formatting file to use to format the results when the command succeeds.
overdraft=qvalue	Optional	Amount of storage overdraft quota to allow the account. qvalue is a non-negative, floating point value.
overdraft_orig=string	Optional	See text.

Table 4–17 (Cont.) `modify_user` **command parameters**

parameter=value		Description
overdraft_units=qunits	Optional	Storage units in which the overdraft quota qvalue is expressed. Must be one of b (bytes), kb (kilobytes), mb (megabytes), or gb (gigabytes). If this parameter is not specified, storage units of kb are assumed.
owner=string	Optional	Name of the owner of the account. string has a maximum length of 40 bytes.
password=string	Optional	The account password. string has a maximum length of 32 bytes. A zero-length password allows anyone to access the account.
past_block_days=value	Optional	Set the past block days to the specified value, value, and clear the past block days remainder field. value is a non-negative integer.
pre_expire=pevalue	Optional	Used to mark the account as pre-expired or not. If the account is marked as pre-expired, the last_pwd_change value is set to "No time recorded". If the account is marked as not pre-expired the last_pwd_changefield is set to the current time. pevalue must be one of dont-change, pre-expire, or un-pre-expire.
private=string	Optional	Data to store in the account's site-defined private data area. string has a maximum length of 64 bytes.
quota=qvalue	Optional	Primary message storage quota for the account. A qvalue of zero grants the account unlimited storage quota. If the quota_units parameter is not specified, then units of kilobytes are assumed. qvalue is a non-negative, floating point number.
quota_orig=string	Optional	See text.
quota_units=qunits	Optional	Storage units in which the quota qvalue is expressed. Must be one of b (bytes), kb (kilobytes), mb (megabytes), or gb (gigabytes).
received_bytes=rvalue	Optional	Cumulative count of received message bytes. If the received_bytes parameter is not specified, then units of kilobytes are assumed. rvalue is a non-negative, floating point number.
received_bytes_orig=string	Optional	See text.
received_byte_units=runits	Optional	Storage units in which the received bytes rvalue is expressed. Must be one of b (bytes), kb (kilobytes), mb (megabytes), or gb (gigabytes).
received_messages=mvalue	Optional	Cumulative count of received messages. mvalue is a non-negative, integer.

Table 4–17 (Cont.) `modify_user` **command parameters**

`parameter=value`		**Description**
`total_connect=tvalue`	Optional	Set the total connect time to the specified value, `tvalue`. If units are not specified with the `total_connect_units` parameter, units of seconds are assumed. `tvalue` is a non-negative, floating point value.
`total_connect_orig=string`	Optional	See text.
`total_connect_units=tunits`	Optional	Time units in which the total connect `tvalue` is expressed. Must be one of `s` (seconds), `m` (minutes), `h` (hours), or `d` (days).
`total_connections=value`	Optional	Set the cumulative count of connections to the value, `value`.
`username=string`	Required	Username of the account to modify. This string has a maximum length of 32 bytes and can contain wild card characters.

When multiple instances of the `username` parameter are provided, each of the associated accounts are modified. When the value specified with a `username` parameter contains wild card characters, the operation will be confined to any matching user names within the management group and subgroups of the manager requesting the operation. If a group name is also specified, then the operation will be further confined to that group and subgroups.

One important usage model for the `modify_user` command involves presenting a manager with a form whose input fields display an account's current settings. The manager is then free to change the values in only a few of the fields and then post the form to the management CGI via a submit button. The posted form will, however, carry values for every field: both those values which were and were not changed by the manager. Thus, the CGI will set every account field. This is acceptable in those cases where the displayed value reflects exactly the actual value stored for the account. However, this is not the case for those values which are displayed as fixed precision, floating point numbers. Specifically, the quota, overdraft quota, received bytes, and total connect time fields.

Hence the parameters `overdraft_orig`, `quota_orig`, `received_bytes_orig`, and `total_connect_orig`. The string values for these parameters should be the original string representation of the value presented in the form along with a unit specifier. For instance, suppose an account has a quota of 500,000 bytes and this is displayed in a form as 0.5 megabytes (524,288 bytes). If the manager doesn't change the quota field, then the CGI will see a request to change the quota to 0.5 megabytes and thus to increase the account's quota by 24,288 bytes. To avoid this mistake, the quota_orig parameter should be supplied:

```
quota_orig=0.5m
```

When the management CGI receives a `quota_orig` parameter, it compares the parameter value to that specified with the `quota` and `quota_units` parameters. If the two values are identical, then the account's quota field is not changed.

Thus, the values of the overdraft_orig, quota_orig, received_bytes_orig, and total_connect_orig parameters should be, respectively, representations of the overdraft, quota, received_bytes, and total_connect values initially presented in a form. The values should immediately be followed by a unit specifier: b (bytes), k (kilobytes), m (megabytes), g (gigabytes), or s (seconds), m (minutes), h (hours), d (days). For example, a HTML form might appear as

```
<FORM ACTION="/popstore/" METHOD="POST">
<INPUT TYPE="hidden" NAME="quota_orig" VALUE="0.5m">
<INPUT TYPE="text" NAME="quota" VALUE="0.5">
<SELECT NAME="quota_units" SIZE=1>
  <OPTION>bytes<OPTION>kbytes<OPTION SELECTED>mbytes
</SELECT>
<ENDFORM>
```

4.3.4.14 rename_user Command: rename a user account

The rename_user command changes the username associated with an account. No other attribute of the account is changed. Parameter names and associated values accepted by the command are listed in Table 4–18.

Table 4–18 rename_user **command parameters**

parameter=value		Description
log=bvalue	Optional	Boolean value, 0 or 1. When bvalue is 1, then a status message will be output indicating a successful operation. The default is bvalue=0. Note that errors are always reported.
on_error=file-spec	Required	Name of the formatting file to use to format the results when the command fails.
on_success=file-spec	Required	Name of the formatting file to use to format the results when the command succeeds.
username=string	Required	The first instance of this parameter gives the old username and the second instance gives the new username. string has a maximum length of 32 bytes.

The username parameter must appear twice in the command. The first instance, when reading the command from left to right, gives the name of the account to rename. The second instance gives the new name to assign to the account.

As an example, consider the command

```
command=rename_user&username=jdoe&username=jane.doe&
  success_format=rename_success.txt&error_format=rename_error.txt
```

which renames the account jdoe to jane.doe.

4.3.4.15 `show_counters` Command: show channel counters

The `show_counters` command is used to display channel counters for PMDF channels such as the popstore channel. The parameter names and associated values accepted by the command are listed in Table 4–19.

Table 4–19 `show_counters` command parameters

parameter=value		Description
`channel=name`	Optional	Name of the channel for which to display information If omitted, then information on all channels is displayed. `name` can contain wild card characters.
`mformat=file-spec`	Optional	Name of the formatting file to use to format each set of channel counters. The recognized substitution strings for this formatting file are listed in Tables 4–10 and 4–20. At least one of `mformat` or `pformat` must be specified.
`on_error=file-spec`	Required	Name of the formatting file to use to format the results when the command fails.
`on_success=file-spec`	Required	Name of the formatting file to use to format the results when the command succeeds.
`pformat=file-spec`	Optional	Name of the formatting file to use to format each set of channel counters. The recognized substitution strings for this formatting file are listed in Tables 4–10 and 4–20. At least one of `mformat` or `pformat` must be specified.
`rooted=bvalue`	Optional	Boolean value, `0` or `1`, indicating whether or not an exact channel name match is performed. When `bvalue` is `0`, only the channel, if any, exactly matching the channel name string is returned. When `bvalue` is `1`, then the channel name string is treated as a prefix and any channels with names beginning with the supplied name will be returned. When not specified, a `bvalue` of `0` is assumed.

At least one of `mformat` and `pformat` must appear in the command. Both can appear in which case the output generated by both formatting files will be included in the response.[4]

An example command to show channel counters for the popstore channel is shown below:

```
command=show_counters&channel=popstore&pformat=counters.txt&
  on_success=counters_success.txt&on_error=counters_error.txt
```

A sample formatting file is shown in Example 4–3.

[4] The utility of specifying both `mformat` and `pformat` lies in having one format normal channel counter information and the other format popstore message and profile counts. That then allows a command to optionally display just channel counters or both channel counters and popstore message and profile counts. Note that counting the number of popstore messages can be slow which is why a manager can not want to always view that information.

Example 4–3 `show_counters` **formatting file**

```
%none{No matching channels found}
%first{Channel                                    Stored messages}
%first{-----------------------------------------------------------}
%counter_channel{%-40s}  %counter_stored_messages
```

Table 4–20 `show_counters` **command substitution strings**

Substitution string	Type	Format	Description
`%counter_channel`	string	`%s`	The channel's name.
`%counter_delivered_messages`	int	`%d`	Total cumulative count of messages processed (dequeued) by the channel.
`%counter_delivered_recipients`	int	`%d`	Total cumulative count of message recipients processed (dequeued) by the channel.
`%counter_delivered_volume_b` `%counter_delivered_volume_k` `%counter_delivered_volume_m` `%counter_delivered_volume_g`	float	`%.2f`	Total cumulative count of message volume processed (dequeued) by the channel as measured in bytes (`_b`), kbytes (`_k`), mbytes (`_m`), and gbytes (`_g`).
`%counter_received_messages`	int	`%d`	Total cumulative count of messages sent to the channel (messages enqueued to the channel).
`%counter_received_recipients`	int	`%d`	Total cumulative count of message recipients sent to the channel (recipients enqueued to the channel).
`%counter_received_volume_b` `%counter_received_volume_k` `%counter_received_volume_m` `%counter_received_volume_g`	float	`%.2f`	Total cumulative count of message volume sent to the channel (enqueued to the channel) as measured in bytes (`_b`), kbytes (`_k`), mbytes (`_m`), and gbytes (`_g`).
`%counter_stored_message_files`	uint	`%u`	Count of message files currently stored in the popstore.
`%counter_stored_messages`	int	`%d`	Count of messages currently enqueued (sent) to the channel.
`%counter_stored_profile_files`	uint	`%u`	Count of popstore user accounts (profile files).
`%counter_stored_recipients`	int	`%d`	Count of recipients currently enqueued to the channel.
`%counter_submitted_messages`	int	`%d`	Total cumulative count of messages sent by the channel (enqueued by the channel).
`%counter_submitted_recipients`	int	`%d`	Total cumulative count of message recipients sent to by the channel (enqueued to by the channel).
`%counter_submitted_volume_b` `%counter_submitted_volume_k` `%counter_submitted_volume_m` `%counter_submitted_volume_g`	float	`%.2f`	Total cumulative count of message volume sent by the channel (enqueued by the channel) as measured in bytes (`_b`), kbytes (`_k`), mbytes (`_m`), and gbytes (`_g`).

4.3.4.16 `show_message` Command: show a user’s message

The `show_message` command is used to display a message stored in the popstore. The message to display is identified by its file name as given by the `%msg_filename` substitution string of the `show_user` command. Note that the message to be displayed is first checked to ensure that at least on of its recipients are in the manager’s management group.

The parameter names and associated values accepted by the command are listed in Table 4–21.

Table 4–21 `show_message` **command parameters**

parameter=value		Description
`filename=file-spec`	Required	File name for the message file to display. Note that this is not a full file path but rather a path relative to the popstore message file directory tree.
`mformat=file-spec`	Required	Name of the formatting file to use to format the message. The recognized substitution strings for this formatting file are listed in Table 4–10 and Table 4–22.
`on_error=file-spec`	Required	Name of the formatting file to use to format the results when the command fails.
`on_success=file-spec`	Required	Name of the formatting file to use to format the results when the command succeeds.

An example command to show the message file `A0123450` is shown below:

```
command=show_message&filename=A0123450&mformat=msg.txt&
  on_success=msg_success.txt&on_error=msg_error.txt
```

A sample formatting file is shown in Example 4–4.

Example 4–4 `show_message` **formatting file**

```
Recipient count: %msg_rcpt_count
Reference count: %msg_ref_count

From: %msg_env_from
To:   %msg_env_to

%msg_content
```

Table 4–22 `show_message` **command substitution strings**

Substitution string	Type	Format	Description
`%msg_channel`	string	`%s`	Name of the channel which delivered the message to the popstore.

Table 4–22 (Cont.) `show_message` **command substitution strings**

Substitution string	Type	Format	Description
`%msg_content`	string	`%s`	Content of the message (both RFC822 message header and body). Each line of the message content will be formatted using the formatting string and terminated with a CRLF pair.
`%msg_created`	string	`%s`	Date and time at which the message file was created.
`%msg_env_from`	string	`%s`	Message's envelope From: address.
`%msg_env_id`	string	`%s`	Message's envelope identifier.
`%msg_env_to`	string	`%s`	Message's envelope To: address. When there is more than one envelope To: address, each address will listed with comma separators.
`%msg_filename`	string	`%-s`	Name of the file containing this message.
`%msg_header`	string	`%s`	The message's RFC822 header. Each line of the message header will be formatted using the formatting string and terminated with a CRLF pair.
`%msg_id`	string	`%s`	Content of the message's RFC822 Message-id: header field.
`%msg_rcpt_count`	uint	`%u`	Number of popstore users which were recipients of this message.
`%msg_ref_count`	uint	`%u`	Number of popstore users which currently have references to this message. This is the number of popstore users who have not deleted their copy of this message.
`%msg_size_b`	uint	`%u`	Size of the message content as measured in bytes.
`%msg_size_k` `%msg_size_m` `%msg_size_b`	float	`%.2f`	Size of the message content as measured, respectively, in kbytes (`_k`), mbytes (`_m`), or gbytes (`_g`).
`%msg_version`	uint	`%u`	Value of the message file's format version field.

4.3.4.17 `show_user` Command: show a user account

The `show_user` command is used to display information about a given user. The parameter names and associated values accepted by the command are listed in Table 4–23.

Table 4–23 `show_user` **command parameters**

`parameter=value`		Description
`mformat=file-spec`	Optional	Name of the formatting file to use to format each message reference in the account's list of stored messages. The recognized substitution strings for this formatting file are listed in Tables 4–10 and 4–24.
`on_error=file-spec`	Required	Name of the formatting file to use to format the results when the command fails.

Table 4–23 (Cont.) `show_user` **command parameters**

parameter=value		Description
`on_success=file-spec`	Required	Name of the formatting file to use to format the results when the command succeeds.
`pformat=file-spec`	Required	Name of the formatting file to use to format each account listing. The recognized substitution strings for this formatting file are listed in Table 4–10 and 4–15.
`username=string`	Required	Name of the account to display. `string` has a maximum length of 32 bytes. Wild cards are not permitted.

The `username` parameter can appear more than once in which case information for each account will be displayed.

An example command to display the `jdoe` account is shown below:

```
command=show_user&username=jdoe&pformat=susr.txt&mformat=smsg.txt
  on_success=susr_success.txt&on_error=susr_error.txt
```

Sample formatting files are shown in Example 4–5 and Example 4–6. These are the formatting files used by the interactive command line management utility.

Example 4–5 `show_users` **account formatting file**

```
Username:           %username
Owner:              %owner
Group:              %group_name
Store Type:         %store
Usage flags:        %flags
Site-defined:       %private

Last pwd change:    %last_pwd_change
Last connect:       %last_connect
Last disconnect:    %last_disconnect
Total connect time: %total_connect_dhms
Total connections:  %total_connections
Past block days:    %past_block_days
Last billing:       %last_billing

Message count:      %message_count{%10u} (%received_messages total message%s received
Quota used:         %quota_used_k{%10.2f} Kbyte%s
Primary quota:      %quota_k{%10.2f} Kbyte%s
Overdraft quota:    %overdraft_k{%10.2f} Kbyte%s
```

Example 4–6 `show_users` **message list formatting file**

```
%none{User has no stored messages}
%msgr_id{%4u}. Filename: %msgr_filename
      Received: %msgr_created
      Size:     %msgr_size_k K
      Flags:    %msgr_flags
```

The formatting field for the `%msgr_flags_read` substitution string works differently from other substitution strings. The formatting field is interpreted as two strings separated by a vertical bar, |. The first string is that substituted when the message read flag is not set and the second string that when the field is set. For instance, the formatting instructions

```
Message read: %msgr_flags_read{No|Yes}
```

produces the output

```
Message read: No
```

for a message not marked as read. For a message which is marked as having been read, the output would instead be

```
Message read: Yes
```

Table 4–24 `show_user` **mformat command substitution strings**

Substitution string	Type	Format	Description
`%msgr_created`	string	`%s`	Creation date and time for the referenced message file.
`%msgr_filename`	string	`%-s`	Name of the message file containing the referenced message.
`%msgr_flags`	string	`%s`	Comma separated string representation of flags associated with the state of the message. Presently the only flag is `READ` which appears when the message is marked as having been read.
`%msgr_flags_read`	string	`Unread|Read`	See text.
`%msgr_id`	uint32	`%u`	Index of the message in the list of stored messages. The first message has an index of `1`, the second an index of `2`, *etc.*
`%msgr_size_b`	uint32	`%u`	Size in bytes of the message content.
`%msgr_size_k` `%msgr_size_m` `%msgr_size_g`	float	`%.2f`	Size of the message content as measured, respectively, in kbytes (`_k`), mbytes (`_m`), or gbytes (`_g`).
`%msgr_uidl`	string	`%-s`	UIDL for this instance of the message.

4.3.4.18 unforward Command: remove a forwarding address

Previously established forwarding addresses can be undone with the unforward command. Parameter names and associated values accepted by the command are listed in Table 4–25.

Table 4–25 unforward command parameters

parameter=value		Description
on_error=file-spec	Required	Name of the formatting file to use to format the results when the command fails.
on_success=file-spec	Required	Name of the formatting file to use to format the results when the command succeeds.
username=string	Required	Username for which to remove the forwarding. string has a maximum length of 32 bytes and can not contain wild cards.

When multiple instances of the username parameter are provided, the forwardings for each of the specified usernames are removed.

As an example, consider the command which removes the forwardings for the two usernames sue and tom:

```
command=unforward&username=sue&username=tom&
  on_success=unforward_success.txt&on_error=unforward_error.txt
```

5 Web-based User Interface

The popstore and MessageStore provides a web-based user interface. This interface is intended to provide a simple, safe mechanism with which popstore and MessageStore users can change their password and see basic usage information about their account. Primitive viewing of stored messages is also permitted (for popstore accounts only).

So as to maintain a solid brick wall between the web-based management interface and this web-based user interface, separate HTTP CGIs are used for each. Indeed, the web-based user interface takes the form of two distinct CGIs: one for showing account information and stored messages and another for changing passwords. This simplifies using the `HTTP_ACCESS` mapping table to restrict access: different sets of access controls can be placed on the management, user password, and user account information CGIs. Despite being separate CGIs, however, the HTTP commands used for the three interfaces are identical in the sense that the user interface commands are a small, tightly controlled subset of the management interface commands.

This chapter provides a description of both the web-based user interface as well as the HTTP CGIs supporting that interface.

Sites which are familiar with the Hypertext Markup Language (HTML), can customize the interface or redesign it entirely. This is where an understanding of the HTTP CGI used by the popstore and MessageStore is needed. See Section 5.2 for details.

5.1 Using the User Interface

Before attempting to use the web-based user interface, you must first configure the PMDF Dispatcher and HTTP server. This is accomplished by running the Dispatcher configuration utility as described in the *PMDF Installation Guide*. If you have not configured the Dispatcher, do so now.

In order to use the user interface, you will need a web client which supports HTML tables. The user CGIs are accessed via the URLs

```
http://host:7633/msps_user/
```

and

```
http://host:7633/chng_pwd/
```

The first URL accesses the user information CGI and the second the password changing CGI.

In place of `host` above, use the actual IP host name of the system running the PMDF HTTP server. If you chose to run the PMDF HTTP server on a port other than port 7633, then specify that port number in place of `7633` in the above URL.

If you are met with an "Access Forbidden" message when you attempt to access the above URL, then you need to check:

1. that you have configured the PMDF Dispatcher and HTTP server;
2. that you have an HTTP_ACCESS mapping table in your PMDF mapping file;
3. that if you created or changed the HTTP_ACCESS mapping table that you have recompiled your configuration if using a compiled configuration; and,
4. that you have started the PMDF Dispatcher and that it is running.

Note further that if you change the HTTP_ACCESS mapping table, then you will need to restart the HTTP server in order for those changes to take effect.

When a user attempts to access either URL, their web client will prompt them for their popstore/MessageStore account name and password. The user must supply the correct name and password of a popstore or MessageStore account. [1]

Note that the password change page can be used by system managers to change a password on behalf of a user, as well as used by users to change their own passwords. When users are changing their own passwords, the user must specify their own username and existing password when prompted. System managers must specify the username and password of either the `pmdf` account, or the system account (`root` on Unix, `SYSTEM` on OpenVMS, or `Administrator` on Windows).

5.1.1 Restricting Access

Users of the popstore/MessageStore must be able to supply the correct name and password for a popstore/MessageStore account. You can impose further restrictions through the use of an HTTP_ACCESS mapping table. For instance, you can deny access to the user interfaces from all external IP addresses. See the description of the HTTP_ACCESS mapping table in the *PMDF System Manager's Guide* for directions on how to use the HTTP_ACCESS mapping table.

5.1.2 Location of the User Interface

The GIF, HTML, and formatting files which comprise the web-based management interface are located in the /pmdf/www/msps_user/ and /pmdf/www/chng_pwd/ directory trees on UNIX and NT systems, and the PMDF_ROOT:[WWW.MSPS_USER] and PMDF_ROOT:[WWW.CHNG_PWD] directories on OpenVMS systems. Sites wanting to customize the interface or develop their own interface can use these files as a starting point. However, do not make changes to the supplied files themselves: *any changes will*

[1] Presently, the popstore/MessageStore CGIs use the HTTP Basic Authentication scheme. This is currently the only standardized HTTP authentication scheme. Unfortunately, with this authentication scheme, the account name and plain text password are transmitted in the clear from the client to the server.

be lost when you upgrade PMDF. Instead, make copies of the files and access them via the appropriate URL; *e.g.*,

```
http://host:7633/msps_user/x-main-page.html
```

Details on developing your own formatting files are given in Section 5.2.

5.2 The User CGIs

In order to customize the management interface, it is first necessary to understand how the CGIs process HTTP requests and formulate HTTP responses. This is described in Section 4.3.1 and Section 4.3.2. Section 5.2.1 describes the individual commands which can be embedded in those requests.

5.2.1 User Interface Commands

When a command is sent to the user interface, the command is applied to the popstore/MessageStore account whose name is given with the HTTP authentication. That is, the HTTP command carries with it a popstore/MessageStore username and password. If the username and password are valid, then the command is applied to the popstore/MessageStore account denoted by the username.

As described in Section 4.3.1, commands take the general form

```
command=command-name&parameter-name-1=parameter-value-1&
  ...&parameter-name-N=parameter-value-N
```

In the above, `command-name` gives the name of the command to execute. It is then followed by two or more parameters which provide supplemental information relevant to the operation to be performed. When parameter names are duplicated in the command, only the right most instance of the parameter=value pair is honored.

For those parameters whose values are file specifications, the file specifications must be relative file paths specifying files in the /pmdf/www/msps_user/ or /pmdf/www/chng_pwd/ directory trees on UNIX and NT systems, or the directories `PMDF_ROOT:[WWW.MSPS_USER]` and `PMDF_ROOT:[WWW.CHNG_PWD]` on OpenVMS systems.

The valid command names are listed in the table below and described in the following sections.

Command name	Section	Description
`delete`	5.2.1.1	Delete a message stored for the user (popstore only)
`set_pwd`	5.2.1.3	Change the password for the user
`show`	5.2.1.2	Show a message stored for the user (popstore only)

Command name	Section	Description
`show_user`	5.2.1.4	Show settings for the user account

5.2.1.1 `delete` command: delete a stored message (popstore only)

A specific message stored for a popstore user account can be deleted with the `delete` command. The message to be deleted is identified by means of a UIDL as returned by the `%msgr_uidl` substitution string of the `show_user` command.

This command must be presented to the user information CGI via the URL

```
http://host:7633/msps_user/
```

Parameter names and associated values accepted by the command are listed in Table 5–1.

Table 5–1 `delete` **Command Parameters**

`parameter=value`		Description
`on_error=file-spec`	Required	Name of the formatting file to use to format the results when the command fails.
`on_success=file-spec`	Required	Name of the formatting file to use to format the results when the command succeeds.
`log=bvalue`	Optional	Boolean value, `0` or `1`. When `bvalue` is `1`, then a status message will be output indicating a successful operation. The default is `bvalue=0`. Note that errors are always reported.
`uidl=string`	Required	UIDL for this user's instance of the message to be deleted.

As an example, consider the following command which deletes the message with UIDL `!!!!01234`:

```
command=delete&uidl=!!!!01234&
  on_success=delmsg_success.txt&
  on_error=delmsg_error.txt
```

5.2.1.2 `show` command: show a stored message (popstore only)

The `show` command is used to display a message stored in the popstore. The message to display is identified by its file name as given by the `%msg_filename` substitution string of the `show_user` command.

This command must be presented to the user information CGI which uses the URL

```
http://host:7633/msps_user/
```

The parameter names and associated values accepted by the command are listed in Table 5–2.

Table 5–2 `show` **Command Parameters**

`parameter=value`		**Description**
`filename=file-spec`	Required	File name for the message file to display. Note that this is not a full file path but rather a path relative to the popstore message file directory tree. A user can only display a message file referenced in his or her profile's list of stored messages. As such, user's cannot try displaying random file names in the hopes of seeing another user's mail.
`trim=bvalue`	Optional	Boolean value, `0` or `1`. Only honored when `part` is also specified. When `bvalue` is `1` header trimming will be applied. When `bvalue` is `0`, no header trimming is applied. The default is `bvalue=1`.
`mformat=file-spec`	Required	Name of the formatting file to use to format the message. The recognized substitution strings for this formatting file are listed in Table 4–10 and Table 4–22.
`on_error=file-spec`	Required	Name of the formatting file to use to format the results when the command fails.
`on_success=file-spec`	Required	Name of the formatting file to use to format the results when the command succeeds.
`part=bvalue`	Optional	Non-negative integer value indicating which message part to display. To display the entire message, specify `part=0`. The first message part is `part=1`, the second `part=2`, and so on.

An example command to show the message file `A0123450` is shown below:

```
command=show&filename=A0123450&mformat=msg.txt&
  on_success=msg_success.txt&on_error=msg_error.txt
```

A sample formatting file is shown in Example 5–1.

Example 5–1 `show_message` **Formatting File**

```
%none{Message could not be accessed or no longer exists}

<PRE>
%msg_content
</PRE>
```

When the `part` parameter is specified, the message will be interpreted as a MIME-formatted message and displayed all at once, but divided into parts. Binary parts will be represented by hypertext links which can be clicked on to download the associated data. Header trimming will be applied to the message's outer header as well as any internal headers. To suppress this trimming, specify `trim=0` in the command. Header trimming uses a default header trimming option file. Sites can supply their own header trimming option file if they want. See Section 5.2.2 for further details.

5.2.1.3 `set_pwd` command: change the user's password

The `set_pwd` command is used to change a user's password. This command must be presented to the password changing CGI which uses the URL

```
http://host:7633/chng_pwd/
```

The parameter names and associated values accepted by the command are listed in Table 5–3.

Table 5–3 `set_pwd` Command Parameters

parameter=value		Description
`newp=string1`	Required	New password to use for the account. Length of this string can not exceed a length of 32 bytes.
`on_error=file-spec`	Required	Name of the formatting file to use to format the results when the command fails.
`on_success=file-spec`	Required	Name of the formatting file to use to format the results when the command succeeds.
`username=string3`	Required	The name of the account whose password to change. Length of this string can not exceed a length of 32 bytes.
`verp=string2`	Required	Verification copy of the new password to use for the account. The value of `string2` must be identical to the value of `string1` given by the `newp` parameter. Length of this string can not exceed a length of 32 bytes.

An example command is shown below:

```
command=set_pwd&username=myacct&newp=SeCReT&verp=SeCReT&
  on_success=pwd_success.txt&on_error=pwd_error.txt
```

5.2.1.4 `show_user` command: show information about the user's account

The `show_user` command is used to display information about a given user. The parameter names and associated values accepted by the command are listed in Table 5–4.

This command must be presented to the user information CGI via the URL

```
http://host:7633/msps_user/
```

Table 5–4 `show_user` Command Parameters

parameter=value		Description
`mformat=file-spec`	Optional	(popstore only) Name of the formatting file to use to format each message reference in the account's list of stored messages. The recognized substitution strings for this formatting file are listed in Tables 4–10 and 4–24.
`on_error=file-spec`	Required	Name of the formatting file to use to format the results when the command fails.

Table 5–4 (Cont.) `show_user` **Command Parameters**

parameter=value		Description
`on_success=file-spec`	Required	Name of the formatting file to use to format the results when the command succeeds.
`pformat=file-spec`	Required	Name of the formatting file to use to format each account listing. The recognized substitution strings for this formatting file are listed in Table 4–10 and 4–15.

An example command is shown below:

```
command=show_user&pformat=susr.txt&mformat=smsg.txt
  on_success=susr_success.txt&on_error=susr_error.txt
```

Sample formatting files are shown in Example 5–2 and Example 5–3. These are the formatting files used by the interactive command line management utility.

Example 5–2 `show_user` **Account Formatting File**

```
Username:            %username
Owner:               %owner
Group:               %group_name
Store Type:          %store
Usage flags:         %flags
Site-defined:        %private

Last pwd change:     %last_pwd_change
Last connect:        %last_connect
Last disconnect:     %last_disconnect
Total connect time:  %total_connect_dhms
Total connections:   %total_connections
Past block days:     %past_block_days
Last billing:        %last_billing

Message count:       %message_count{%10u} (popstore only)
Quota used:          %quota_used_k{%10.2f} Kbyte%s
Primary quota:       %quota_k{%10.2f} Kbyte%s
Overdraft quota:     %overdraft_k{%10.2f} Kbyte%s
```

Example 5–3 `show_user` **Message List Formatting File (popstore only)**

```
%none{User has no stored messages}
%msgr_id{%4u}. Filename: %msgr_filename
      Received: %msgr_created
      Size:     %msgr_size_k K
      Flags:    %msgr_flags
```

5.2.2 Header trimming of displayed messages

Header trimming is applied to the displayed message's header when the `part` parameter is supplied to the `show` command. The user CGI uses a default header trimming option file, `popstore_cgi_headers.opt`, file located in the PMDF table directory. To use different options, create a new header trimming option file named `popstore_site_cgi_headers.opt`. Place that file in the PMDF_table directory. (On UNIX systems, ensure that the file is owned by the PMDF account.) When that file is detected, it will be used instead of the Process Software-supplied file. The Process Software-supplied file will be ignored and only those options specified in the site-supplied file used.

See the *PMDF System Manager's Guide* for information on the use and format of header trimming option files.

6 UNIX & Windows Command Line Management Utility

The command line management utility is an interactive, command oriented interface for managing popstore and MessageStore accounts. Users with operating system privileges as well as users with privileged popstore or MessageStore accounts can use the utility. Also, the utility can be used as a report generator as described briefly in Section 6.15.2 and more completely in Chapter 9.

The utility is invoked with the command

```
# pmdf popstore
```

or

```
# pmdf msgstore
```

Use the `exit` or `quit` commands to exit the utility. Table Table 6–1 summarizes the recognized commands; see Section 6.18 for complete command descriptions.

Table 6–1 Summary of command line management commands (UNIX & NT)

Command	Description
`add`	Add a new user account
`copy`	Copy a user account (popstore only)
`delete`	Delete a user account or messages
`exit`	Exit the utility
`forward`	Establish a forwarding address
`group`	Manipulate management groups
`login`	Enable privileges by “logging in” to a privileged popstore/msgstore account
`logout`	Disable privileges
`modify`	Modify a user account
`noforward`	Remove a forwarding address
`quit`	Exit the utility
`rename`	Rename a user’s account (popstore only)
`set`	Select user domain or set units used for expression of storage and time
`show`	Display information about user accounts, messages, or forwardings

On UNIX, the command recall and editing capabilities are provided by the open source software `libedit` (also known as `editline`). By default, the standard "vi" key bindings are defined. You can change various elements of the editing environment, such as using "Emacs" key bindings instead of "vi", by creating in your home directory a file called `.editrc`. See the `editrc` manpage for more information.

6.1 Basic operation

Popstore and MessageStore accounts are managed using four basic commands: add, delete, modify, and show. These four commands add accounts, remove accounts, modify accounts, and display information about accounts. For popstore accounts, two additional commands, copy and rename, can be used to create new accounts which look like existing accounts and to change the name of an existing account.

When an account is created with the add command, you can also specify various account settings such as the account password, the name of the account's owner, and storage quotas. Once the account is created, you can subsequently change these settings with the modify command.

Some of the account settings involve the use of values expressed in units of storage or time. By default, the units of storage are kbytes (1024 bytes), and the units of time are days. Within a session with the utility, these units can be changed with the set command; see the command descriptions in Section 6.18 for details.

6.2 Adding new accounts

New accounts can be added in one of two ways: by creating a new account or (for popstore only) copying an existing account. The former is done with the add command while the latter with the copy command. Regardless of the method chosen, when a new account is added a name and password to associate with the account should be chosen. This is the name and password which the owner of the account must use in order to access messages stored for the account. Specify -flags=pwd_elsewhere to create the account with an externally stored password (*e.g.*, an /etc/passwd password). Optionally, account quotas and an ownership field identifying the owner of the account can also be specified. Quotas not specified will be copied from the default account.

See Section 1.3.1 for a discussion of the name space allowed for account usernames.

For instance, suppose that Jane Doe is to be given the account jdoe with the password SecRet. The command to add the account would then be

```
popstore> add jdoe -password=SecRet -owner="Jane Doe"
popstore> show jdoe
Username:           jdoe
Owner:              Jane Doe
Group:
Store Type:     popstore
Usage flags:
Site-defined:

Last pwd change:    Fri Nov 15 12:02:22 2012
Last connect:       No time recorded
Last disconnect:    No time recorded
Total connect time: 0 00:00:00
Total connections:  0
Past block days:    0
Last billing:       Fri Nov 15 12:02:22 2012
```

```
Message count:              1 (1 total messages received)
Quota used:              0.71 Kbytes
Quota:                2048.00 Kbytes
Overdraft:              20.00 Kbytes
```

Note that POP passwords are case sensitive.

It is important to note that when the popstore was configured, default account settings were specified. These settings are kept in the form of a default account named `default`. Whenever an account is added with the `add` command, the popstore actually copies the `default` account to the new account thereby using the `default` account's settings as the basis for the new account.

Copying an existing popstore account with the `copy` command also creates a new account. In the example below, the new account `doe` is created by copying the account `ariel`:

```
popstore> show ariel

Username:        ariel
Owner:           Tempest Ariel
Group:
Store Type:     popstore
Usage flags:
Site-defined:

Last pwd change:    Fri Oct 18 09:12:23 2012
Last connect:       Fri Nov 15 16:09:38 2012
Last disconnect:    Fri Nov 15 16:09:39 2012
Total connect time: 0 00:10:03
Total connections:  145
Past block days:    123953
Last billing:       Fri Oct 18 09:12:23 2012

Message count:              0 (189 total messages received)
Quota used:              0.00 Kbytes
Quota:              102400.00 Kbytes
Overdraft:              10.00 Kbytes

popstore> copy ariel doe -password=secret -owner="John Doe"
popstore> show doe

Username:           doe
Owner:              John Doe
Group:
Store Type:     popstore
Usage flags:
Site-defined:

Last pwd change:    Fri Nov 15 13:23:18 2012
Last connect:       No time recorded
Last disconnect:    No time recorded
Total connect time: 0 00:00:00
Total connections:  0
Past block days:    0
Last billing:       Fri Nov 15 13:23:18 2012
```

```
Message count:              0 (0 total messages received)
Quota used:              0.00 Kbytes
Quota:              102400.00 Kbytes
Overdraft:              10.00 Kbytes
```

Note that when an account is copied to make a new account, the new account does not inherit the messages or accounting information of the original account.

Note: Your PMDF-POPSTORE license controls the number of popstore user accounts which you can have at any one time. When you reach this limit, you will not be allowed to create additional accounts without first deleting some accounts or obtaining a new license with an increased limit. Sites without a PMDF-POPSTORE license are allowed to use the popstore and create up to ten user accounts. This limit does not include the `default` account. Use the `show -count_users` command to display the number of currently defined accounts as well as the limit allowed by your license.

6.3 Listing accounts

Verbose and brief listings of accounts can be generated with the `show` command. When no username parameter is supplied, all accounts are listed:

```
popstore> show -brief
                                      Quota  Message  Quota used
 Username                           (kbytes)   Count    (kbytes)
 ---------------------------------------------------------------
 crw                                33333.00       0        0.00
 david                             102400.00       0        0.00
*dan                                33333.00       0        0.00
 default                             1024.00       0        0.00
 kevin                              40960.00       0        0.00
 kristin                            10240.00       0        0.00
 pekie                              40960.00       0        0.00
 ---------------------------------------------------------------
*Note: privileged users are flagged with an asterisk
```

In the above output, an asterisk is displayed by each account which has the management privilege. The `-brief` switch, causes a brief listing to be generated. Omitting that switch generates a much more detailed listing.

When a `username` parameter is specified, wild cards can be used:

```
popstore> show -brief d*
                                      Quota  Message  Quota used
 Username                           (kbytes)   Count    (kbytes)
 ---------------------------------------------------------------
 david                             102400.00       0        0.00
*dan                                33333.00       0        0.00
 default                             1024.00       0        0.00
 ---------------------------------------------------------------
*Note: privileged users are flagged with an asterisk
```

As discussed in Section 6.15.2, the `-format` switch of the `show` command can be used to generate custom listings. Section 6.11 describes how to use the `-forwardings` switch to display mail forwardings.

6.4 Modifying accounts

The `modify` command can be used to change fields associated with an account. For instance, to reset accounting information, change a password, or to increase or decrease an account quota. In the following example, the password for the account `jdoe` is changed to MYDUCKY:

```
popstore> modify jdoe -password=MYDUCKY
```

The `modify` command does not accept wild cards and the `username` parameter must be specified.

Note that the last connect and disconnect time accounting fields can only be reset to signify a "never connected" state. They can not be changed to an arbitrary time. Also note that when the message count is set to zero, any messages stored for the account are deleted. If the count is reduced but not set to zero then starting with the oldest message, messages are deleted until the desired count is reached. For instance, if the current message count is 5 messages and the count is set to 2 messages, then messages 1, 2, and 3 will be deleted.

6.5 Removing accounts

Accounts are removed with the `delete` command. By default, any messages stored for the account are deleted silently. Use the `-return` switch to return as undelivered any unread messages.

In the example below, the `delete` command is used to delete the `jdoe` account.

```
popstore> show -brief jdoe
                                     Quota  Message  Quota used
 Username                          (kbytes)    Count    (kbytes)
 ----------------------------------------------------------------
 jdoe                               1000.00        2        6.42
 ----------------------------------------------------------------
*Note: privileged users are flagged with an asterisk

popstore> delete -return jdoe
popstore> show -brief jdoe
%POPMGR-E-CANTSHOW, cannot show data
        popstore error #68: No such user
```

To delete more than one account, issue multiple `delete` commands, or use wild cards in the `username` parameter, or use the `-group` switch. For instance, to delete all accounts in the `class97` group, issue the command

```
popstore> delete -group=class97
```

6.6 Changing new account defaults

Recall that when a new account is created with the add command, the default account is used as a source of defaults for the new account. Thus, by changing the settings for the default account with the modify command, you change the defaults which will be applied to subsequently created accounts. For instance, to change the default account quota to 100 kbytes with an overdraft quota of 10 kbytes, you would modify the default account accordingly:

```
popstore> modify default -quota=100 -overdraft=10
```

Note that the settings for the default account only applies to new accounts created with the add command. The default account settings do not apply to all existing accounts.

6.7 Bulk loading accounts

To create many accounts at once, use the run command. Before using that command, first create a text file containing commands with one command per line. For instance,

```
# cat a.com
add jdoe -owner "John Doe" -password 133458
add csmith -owner "Cathy Smith" -password 244587
add rbrown -owner "Randy Brown" -password 569214
...
#
```

Then use the run command to execute the commands from the file:

```
# pmdf popstore
popstore> run -log a.com
popstore> add bjames -owner "Bob Jones" -password 133458
popstore> add csmith -owner "Cathy Smith" -password 244587
popstore> add rbrown -owner "Randy Brown" -password 569214
popstore> ...
popstore> quit
#
```

Each command will be read from the specified file and executed. By default, if an error occurs, it will be reported and processing of the file will be aborted. Use the -ignore switch to cause processing to continue despite any errors which might occur.

6.8 Returning or deleting messages

A user's stored messages can be deleted with the delete -messages command. When the -return switch is also specified, any unread messages are returned as unread to their sender. For instance, to delete the messages for the account jdoe, returning any unread messages, use the command

```
popstore> delete -messages -return jdoe
```

Note that the `-return` switch can also be used with the `delete` command as described in Section 6.5.

6.9 Account quotas

Account quotas are used to control how much mail a user can store. Each account has a primary quota and an overdraft quota which are established when the account is created and can be changed with the `modify` command. An account can not receive a new message if either the account's present storage exceeds the account's quota, or storage of the message would cause the account to exceed the sum of its quota and overdraft quota.[1]

Quotas are set and changed with the `-quota` and `-overdraft` switches of the `add`, `copy`, and `modify` commands. An account can be granted unlimited quota with the `-noquota` switch. By default, quotas are expressed in units of kbytes (1024 bytes). Alternate units can be selected with the `set storage_units` command.

6.10 Blocking access, blocking new mail, and locking passwords

There are three account flags which can be set to influence account access:

`dismail`
The `dismail` flag is used to prevent an account from receiving new mail messages. When this flag is set for an account, new messages are rejected and returned to their sender. The account owner can, however, read any existing messages they might have unless the account is also flagged with the `disuser` flag.

`disuser`
The `disuser` flag is used to deny access to an account. The account can, however, continue to receive new messages unless it is either over quota or also flagged with the `dismail` flag.

`lockpwd`
The `lockpwd` flag prevents users from changing the account's password. The password can only be changed by a user with the management privilege or operating system privileges.

These flags are set on an account with the `modify -flags` command. A flag can be negated by prefixing the name with `no`; for example, `nodismail`. In the following example, the `jdoe` account is marked `dismail` and `disuser`:

```
popstore> modify jdoe -flags=(dismail,disuser)
```

To undo those settings, issue the command:

```
popstore> modify jdoe -flags=(nodismail,nodisuser)
```

[1] See Section 1.3.3 for a discussion of the rationale behind the use of an overdraft quota.

6.11 Forwarding mail

Mail to the popstore can automatically be forwarded to other addresses. The addresses can be within the popstore or outside of it; the forwarding can be for actual popstore accounts or for non-existent accounts.

For example, suppose the host name associated with the popstore is naples.example.com. Suppose further that the popstore user `jdoe` would like her mail forwarded to the address jane.doe@naples.example.com. This forwarding would be accomplished with the `forward` command:

```
popstore> forward jdoe jane.doe@naples.example.com
popstore> show -forwardings
Username                     Forwarding address
-----------------------------------------------------------------------
jdoe                         jane.doe@naples.example.com
```

To forward mail for `jdoe` to herself and her assistant `aclarke@naples.example.com`, you would use the command

```
popstore> forward jdoe "jdoe,aclarke@naples.example.com"
popstore> show -forwardings
Username                     Forwarding address
-----------------------------------------------------------------------
jdoe                         jdoe@naples.example.com, aclarke@naples.example.com
```

Use the `noforward` command to remove a forwarding address:

```
popstore> noforward jdoe
```

As mentioned earlier, forwarding can be established for non-existent accounts. For instance, suppose that the address `list@naples.example.com` is to be forwarded to a PMDF mailing list `list@example.com`. This is accomplished with the `forward` command:

```
popstore> forward list list@example.com
```

Note that currently stored messages are not affected by a forwarding address. Only new messages are affected: new incoming messages will be routed to the forwarding address and not stored. Note also that the presence of a forwarding address does not prevent a user from reading any stored messages which they might have.

6.12 Management groups

Management groups are manipulated with the `group` command. This command has four variants:

`group -add`	Add a new management group
`group -delete`	Delete a management group
`group -list`	List one or more management groups and any subgroups
`group -modify`	Modify an existing management group

It is important to note that use of management groups is not required by the popstore. Moreover, when you place an account into a management group, that group is automatically created if it was not already defined. As such, you only need to use the `group` command to establish group-subgroup relationships and to delete groups which are no longer being used.

Use the `group -add` command to add a new management group. If the group already exists, then an error will ensue and the existing group left unchanged. In that case, use the `group -modify` command to modify the existing group.

To add the management groups `class_97`, `class_98`, `class_99`, and `class_00` use the commands

```
popstore> group -add class_97
popstore> group -add class_98
popstore> group -add class_99
popstore> group -add class_00
```

To then make a new group named `students` containing these four groups, issue the command

```
popstore> group -add students class_97,class_98,class_99,class_00
```

The results can then be listed with the `group -list` command

```
popstore> group -list students
      Group name:  Subgroups contained within
--------------------------------------------
        students:  class_97, class_98, class_99, class_00
        class_97:
        class_98:
        class_99:
        class_00:
popstore>
```

After the Class of 2010 has graduated and left and the Class of 2014 arrives, the `class_10` group can be removed, the `class_14` group added, and the `students` group modified as follows

```
popstore> group -add class_14
popstore> group -modify students class_11,class_12,class_13,class_14
popstore> group -delete class_10
popstore> group -list students
      Group name:  Subgroups contained within
--------------------------------------------
        students:  class_11, class_12, class_13, class_14
        class_11:
        class_12:
        class_13:
        class_14:
popstore>
```

Note that the `group -delete` command deletes just the specified group. It will recursively delete the subgroups contained within the specified group only when `-recur` is specified. Also, it does not delete the accounts contained within the group; to do that, use the `delete -group=group_name *` command; *e.g.*,

```
popstore> delete -group=class_97 -return -log -noconfirm *
```

It is also important to note that a group can contain only a limited number of subgroups as immediate subgroups. Those subgroups can, of course, contain other subgroups. The limit on the number of immediate subgroups of a given group is controlled by the length in bytes of the names of each of those immediate subgroups. If there are to be N immediate subgroups and their combined name lengths are L, then N+L must be less than 236. If a group needs to contain more subgroups than allowed by that limit, nest its definition an extra level. For example,

```
popstore> group -add blah_x blah_01,blah_02,blah_03,...,blah_29
popstore> group -add blah_y blah_30,blah_33,blah_34,...,blah_52
popstore> group -add blah blah_x,blah_y
```

In the above example, the names `blah_01`, ..., `blah_52` are collectively too long to be contained as immediate subgroups of a given group. Therefore they are split between the two groups `blah_x` and `blah_y`. Those two groups are then made subgroups of `blah`. Consequently, the group `blah` contains the all of the groups `blah_01`, ..., `blah_52` as subgroups despite the length limitation.

6.13 User domains

By default, all popstore accounts for a given installation share the same name space. This means that if the two distinct e-mail addresses sandy@example.com and sandy@example.org are directed to the same popstore, then e-mail to either of these addresses will be delivered to the same user account `sandy`. At some sites, however, it is useful to have distinct communities of users, each with their own name spaces. This is achieved with the popstore[2] through the use of “user domains”. Please see Section 1.3.5 for a basic discussion of popstore user domains. This section focuses on the mechanics of creating and managing user domains and assumes knowledge of the material presented in Section 1.3.5.

Note that all e-mail messages for all user domains are co-mingled in the same directory tree. That is, messages for one user domain are not stored distinct from other user domains.

6.13.1 Enabling user domains

To enable the use of user domains, specify USER_DOMAINS=1 in the popstore option file. See Section 3.4 for further details *including an important note for sites who are already associating more than one Internet host name with the popstore.*

Note: You must not use the `filter` channel keyword on the popstore or MessageStore delivery channel if `USER_DOMAINS=1` is set. So doing will cause the wrong filters to be used for users in domains other than the `default` domain.

[2] The PMDF MessageStore does not at present support the use of user domains.

6.13.2 Creating a new user domain

New user domains are created with the `add -domain` command. For example, to create the user domain `example.org`, issue the command

```
popstore> add -domain example.org
```

This will create a new sub-directory tree in the popstore profiles directory. In that sub-directory tree, all profile files for the `example.org` user domain will be stored.

Issuance of the `add -domain` command also creates a `default` user account in the new user domain. This new account will be a copy of the `default` account from the `default` user domain.

A list of existing user domains can be obtained with the `show -domains` command:

```
popstore> show -domains
Cannonicalized domain name
--------------------------
default
example.org
```

6.13.3 Managing a user domain

When you invoke the command-line management utility, it will be set to manage the `default` user domain. To manage a different user domain, you need to issue a `set domain` command. This command tells the utility that all further commands will affect the specified user domain. For instance, to manage the `example.org` domain, issue the command

```
popstore> set domain example.org
```

If you are logged in to a privileged management account at the time you issue the `set domain` command, then you will automatically be logged out unless the management account was in the `default` user domain and not in any management group. This is a reflection of the management policy that only privileged accounts in the `default` user domain and in no management group can manage all accounts in all user domains.

Once the user domain has been selected, all subsequent commands will affect accounts in only that user domain. The `default` user domain can be re-selected with either the command `set domain default` or, more succinctly, `set domain` (no domain name parameter). The currently selected user domain can be shown by issuing the `set` command without any parameters:

```
popstore> set
Using the "default" user domain
...
popstore> set domain naples
popstore> set
Using the "example.org" user domain
...
```

Note that presently, no commands are provided to copy or rename an account between user domains. The popstore API does, however, provide this functionality via the `POPSTORE_user_copy_d` subroutine.

6.13.4 Deleting a user domain

Owing to the rarity of the event, no single command is provided with which to delete an existing user domain. To delete an existing user domain, first select that domain and then delete every account within the domain. This ensures that messages for the account are deleted and that license limits are correctly credited for the deleted accounts. For example,

```
popstore> set domain example.org
popstore> delete -noconfirm *
Delete the specified accounts [Yes/No/Quit]? yes
User account "aalan" deleted
User account "aabe" deleted
...
```

Then, delete the `example.org` directory in the popstore profiles directory (usually, /pmdf/user/).

6.14 Use of the utility by non-privileged users

The command line utility requires operating system privileges in order to operate. So as to control who can or cannot use the utility, the utility requires that the operator either have the necessary operating system privileges or that they have a popstore account which has the `manage` flag set. (The `manage` flag is also required to use the web-based management interface.) Thus, a user lacking operating system privileges can use the utility to manage the popstore provided that a privileged user first creates them an account and grants that account the `manage` flag:

```
popstore> add oper -password=secret -owner="popstore operator" -flag=manage
```

In the above command, a "privileged" popstore account named `oper` with password secret is created. The account's privileges can then be used to manage the popstore via the `login` command:

```
popstore> login oper
Password: secret
Login succeeded; management capabilities enabled
popstore>
```

Once logged in, the utility will allow the user to perform management functions on any account within the same management group and user domain as the user. If the user's account is in no management group — that is has a zero length group name — then the account can manage all accounts within the same user domain. If the user's account is in no management group *and* is in the `default` user domain, then the user can manage *all* accounts in all groups and all user domains.

6.15 Information display formats

Formatting files control the choice and format of information presented with the `show` and `group` commands. These formatting files are located in the `/pmdf/www/popstore/` directory. To display different choices of information or to change the formatting of the information, do not edit the Process Software supplied formatting files — your changes will be lost when you next upgrade PMDF. Instead, create new formatting files and use them instead. Once you have created a new formatting file, you can configure the utility to always use it, as described in Section 6.15.1. Or, you can use it occassionally such as to generate a monthly report. Such occassional usage is effected with the `-format` switch as described in Section 6.15.2.

6.15.1 Changing default display formats

You can change this utility's default display formats through the use of an option file. The file is a PMDF-style option file named `/pmdf/table/popstore_formats`. Each option setting in the file takes the form

```
option-name=option-value
```

where `option-name` is the name of an option to set and `option-value` is the value to set for that option. The recognized option names and their default values are shown below

Option name	Default value	Used with
FORWARD_FORMAT	popmgr_forward.txt	show -forwardings
GROUP_FORMAT	popmgr_groups.txt	group -list
MSG_FORMAT	popmgr_message.txt	show -messages
MSG_BRIEF_FORMAT	popmgr_message_brief.txt	show -messages -brief
PROFILE_FORMAT	popmgr_profile.txt	show
PROFILE_BRIEF_FORMAT	popmgr_profile_brief.txt	show -brief

As an example, suppose you want to change the `show` command's output. You could then copy the `popmgr_profile.txt` file to, say, `site/profile.txt` and then edit the new file.[3] Then, create the `/pmdf/table/popstore_formats` file and in it place the line

```
PROFILE_FORMAT=site/profile.txt
```

Make sure that these file are world readable and owned by the PMDF account. Once you have done this, the `show` output will by default use your new formatting file. Note that if you make this change while running the utility, you will need to exit it and restart it in order for the change to be seen.

[3] Recall that these files are stored in the `/pmdf/www/popstore/` directory tree.

6.15.2 Report generation

Customized reports can be generated using the -format_file switch of the show command. With that switch, a formatting file can be specified. The file will then be used to format the information to be displayed. For example, The syntax of formatting files is described in Section 4.3.4. suppose that the file /pmdf/www/popstore/usage.txt contains the lines

```
%first{                                             Quota}
%first{                                   Owner      Used}
%first{--------------------------------------------------}
%owner{%40s}  %quota_used_k{%8.2f}
%last{--------------------------------------------------}
```

That file could then be used as follows:

```
# pmdf popstore show -format_file=usage.txt
                                                Quota
                                       Owner     Used
-------------------------------------------------
                               John      Doe    24.02
                               Anne   Clarke     8.56
                               Andy   Harris    36.72
                        Default user profile     0.00
                              Karen    Russo   133.98
                            Deanne     Fagan    73.22
-------------------------------------------------
#
```

Note that for security reasons, the formatting files must be kept in the directory tree /pmdf/www/popstore/. This is enforced so as to prevent users with popstore management privileges from using the -format_file switch as a means of displaying protected files from other directories.

See Chapter 9 for further discussion of generating reports.

6.16 Recreating the Default Account

Should you accidentally delete the default account, you can recreate it using the x-add-default command:

```
popstore> x-add-default
```

You can then set settings for the default account using the modify command:

```
popstore> modify default -quota=1000 -overdraft=15
```

6.17 Recreating the User Database

Should the user database become corrupted or be accidentally deleted, you can recreate it using the `x-build-user-db` command:

```
popstore> x-build-user-db
```

This utility will create a new user database and populate it with entries found by scanning the profile directory tree.

6.18 Command descriptions

The remainder of this chapter describes the individual utility commands.

add—Add a new account

Add a new user account to the popstore or MessageStore.

SYNTAX

add *username*
add -domain *domain-name*

Command Switches

-confirm
-domain
-flags=flags
-log
-overdraft=value
-owner=owner
-password=password
-private=data
-prompt
-pwdexpired
-quota=value

PARAMETERS

username
Username to associate with the account or accounts being created.

domain-name
Name of the user domain to create.

DESCRIPTION

The `add` command is used to create a new popstore user accounts. Initial settings for the account are taken from the `default` account. Those settings can then be overridden with the command line switches described below.

If a supplied username conflicts with an existing account, no new account is created and an error message is issued. Note that account usernames are case insensitive. That is the usernames `JDOE`, `JDoe`, and `jdoe` are all identical.

To create a new user domain, specify the `-domain` switch. If the domain already exists, an error will be issued. Otherwise, it will be created and a `default` user account for that domain created. The new `default` account will be a copy of the `default` account from the `default` domain. To begin creating accounts in the new domain, use the `set domain` command. The maximum length of a user domain name is 40 bytes.

Note: Your PMDF-POPSTORE license controls the number of popstore user accounts which you can have at any one time. When you reach this limit, you will not be allowed to create additional accounts without first deleting some accounts or obtaining a new license with an increased limit. Sites without a PMDF-POPSTORE license are allowed to use the popstore and create up to ten user accounts. This limit does not include the `default` account. Use the `show -count_users` command to display the number of currently defined accounts as well as the limit allowed by your license.

COMMAND SWITCHES

-confirm
-noconfirm ***(default)***
Prompt for positive confirmation before carrying out the indicated operation. `-noconfirm` is the default behavior.

-domain
Create a new user domain. This switch can not be used in conjunction with any of the other `add` command switches.

-flags=(flag[,...])
Specify one or more usage flags to associate with the new account. The recognized flags are as follows:

dismail	User is not allowed to receive new mail messages.
disuser	User is not allowed to access their account.
lockpwd	User is not allowed to change their password.
manage	User is allowed to manage popstore accounts.
migrated	Internal flag used by the PMDF migration utilities.
pwd_elsewhere	Password information is stored outside of the popstore.
nodismail	User is allowed to receive new mail messages.
nodisuser	User is allowed to access their account.
nolockpwd	User is allowed to change their password.
nomanage	User is not allowed to manage popstore accounts.
nomigrated	Internal flag used by the PMDF migration utilities.
nopwd_elsewhere	Password information is stored within the popstore.

-log
-nolog ***(default)***
When the operation is successful, output a status message stating that the operation succeeded. Note that error messages are always indicated. `-nolog` is the default behavior.

-overdraft=value
-nooverdraft
The `-overdraft` switch specifies the amount of message storage by which the account can exceed its message storage quota. If the account is currently using less than its storage quota, then a new message can be stored provided that it will

not result in the account's storage exceeding the sum of its storage and overdraft quotas.

By default, this quantity is specified in units of kbytes; however, the `set storage_units` command can be used to change the units used.

The `-nooverdraft` switch is equivalent to specifying `-overdraft=0` and indicates that the account has no overdraft quota.

The maximum value is 4 gigabytes minus 1. If the value specified exceeds the maximum, the value is set to zero (no overdraft quota).

`-owner=owner`
A text string specifying the name of the owner of the account. The length of the string can not exceed 40 bytes. The owner field is not used by the popstore itself; it is generally used by humans to associate account usernames with the actual owner of the account.

`-password=password`
`-nopassword`
Specifies the account's access password. The length of the password can not exceed 32 bytes. Access by non-managers to the account requires knowledge of this password. For instance, to access the account from a POP3 client, the correct username and password associated with the account must be supplied.

When `-nopassword` is specified, the account has no password and anyone can access it.

Note that passwords are case sensitive.

`-private=data`
Site-specific account data can be stored in the account profile file using this switch. The data string can not exceed a length of 64 bytes. This data is not used by the popstore itself but can be used by site-developed procedures which access account profiles.

`-prompt` ***(default)***
`-noprompt`
By default if a wildcard is used, even if `-noconfirm` is specified, one confirmation prompt is issued. If `-noprompt` is specified, there is no prompting at all.

`-pwdexpired`
`-nopwdexpired` ***(default)***
If `-pwdexpired` is specified, then the account is marked as pre-expired. This means that if password expiration is enabled through the `PASSWORD_LIFETIME` option, then the user must change their password immediately.

If `-nopwdexpired` is specified (the default), then the account is marked as not pre-expired. The time of last password change is set to the current time. If password expiration is enabled, then the user does not have to change the password until the `PASSWORD_LIFETIME` has run out.

-quota=value
-noquota
The account's message storage quota. The account can continue to receive new messages so long as the storage consumed by its currently stored messages does not exceed its message storage quota. See also `-overdraft`.

A quota value of zero, conveys unlimited storage. That is, to grant an account unlimited storage set its quota to zero.

By default, this quantity is specified in units of kbytes; however, the `set storage_units` command can be used to change the units used.

When `-noquota` is specified, the account is granted unlimited storage quota. `-noquota` is equivalent to `-quota=0`.

The maximum value is 4 gigabytes minus 1. If the value specified exceeds the maximum, the value is set to zero (unlimited quota).

EXAMPLES

To create the account `jdoe` for Jane Doe with the password `SeCrEt`, and a quota of 10 Mbytes (10240 Kbytes), use the command

```
popstore> add jdoe -password=SeCrEt -owner="Jane Doe" -quota=10240
popstore> show jdoe
Username:           jdoe
Owner:              Jane Doe
Group:
Store Type:     popstore
Usage flags:
Site-defined:

Last pwd change:    Fri 15 Nov 15:33:02 2012
Last connect:       No time recorded
Last disconnect:    No time recorded
Total connect time: 0 00:00:00
Total connections:  0
Past block days:    0
Last billing:       Fri 15 Nov 15:33:02 2012

Message count:              0 (0 total messages received)
Quota used:              0.00 Kbytes
Quota:               10240.00 Kbytes
Overdraft:              20.00 Kbytes
```

copy—Duplicate an account

Create a new account which duplicates an existing account (popstore only).

SYNTAX

copy *from-username to-username*

Command Switches

-confirm
-flags=flags
-group_name=name
-log
-overdraft=value
-owner=owner
-password=password
-private=data
-prompt
-pwdexpired
-quota=value

PARAMETERS

from-username
The name of the popstore account to copy.

to-username
The name of the popstore account to create.

DESCRIPTION

Use the `copy` command to create a new popstore account which duplicates an existing popstore account. Note that the new account will have its usage accounting fields set to zero (*e.g.*, last connect, total connect, past block days, *etc.*). Also, the new account will not have any stored messages, even if the account being duplicated has stored messages.

If the username of the new account conflicts with an existing account, no new account is created and an error message is issued.

See also the `rename` command.

Note: Your PMDF-POPSTORE license controls the number of popstore user accounts which you can have at any one time. When you reach this limit, you will not be allowed to create additional accounts without first deleting some accounts or obtaining a new license with an increased limit. Sites without a PMDF-POPSTORE license are allowed to use the popstore and create up to ten user accounts. This limit does not include the `default` account. Use the `show -count_users` command to display the number of currently defined accounts as well as the limit allowed by your license.

COMMAND SWITCHES

-confirm
-noconfirm ***(default)***
Prompt for positive confirmation before carrying out the indicated operation. `-noconfirm` is the default behavior.

-flags=(flag[,...])
Specify one or more usage flags to associate with the new account. The recognized flags are as follows:

`dismail`	User is not allowed to receive new mail messages.
`disuser`	User is not allowed to access their account.
`lockpwd`	User is not allowed to change their password.
`manage`	User is allowed to manage popstore accounts.
`migrated`	Internal flag used by the PMDF migration utilities.
`pwd_elsewhere`	Password information is stored outside of the popstore.
`nodismail`	User is allowed to receive new mail messages.
`nodisuser`	User is allowed to access their account.
`nolockpwd`	User is allowed to change their password.
`nomanage`	User is not allowed to manage popstore accounts.
`nomigrated`	Internal flag used by the PMDF migration utilities.
`nopwd_elsewhere`	Password information is stored within the popstore.

-group_name=name
Place the new account into the specified management group. If not specified, the the management group of the account being copied is assumed. A manager can not create an account into a group which they cannot manage.

-log
-nolog ***(default)***
When the operation is successful, output a status message stating that the operation succeeded. Note that error messages are always indicated. `-nolog` is the default behavior.

-overdraft=value
-nooverdraft
The `-overdraft` switch specifies the amount of message storage by which the new account can exceed its message storage quota. If the account is currently using less than its storage quota, then a new message can be stored provided that it will not result in the account's storage exceeding the sum of the its storage and overdraft quotas.

By default, this quantity is specified in units of kbytes; however, the `set storage_units` command can be used to change the units used.

The `-nooverdraft` switch is equivalent to `-overdraft=0` and indicates that the new account has no overdraft quota.

The maximum value is 4 gigabytes minus 1. If the value specified exceeds the maximum, the value is set to zero (no overdraft quota).

-owner=owner
A text string specifying the name of the owner of the new account. The length of the string can not exceed 40 bytes. The owner field is not used by the popstore itself; it is generally used by humans to associate account usernames with the actual owner of the account.

-password=password
-nopassword
Specifies the new account's access password. The length of the password can not exceed 32 bytes. Access by non-managers to the account requires knowledge of this password. For instance, to access the account from a POP3 client, the correct username and password associated with the account must be supplied.

Specifying `-nopassword` indicates that the new account does not require a password to access it.

Note that passwords are case sensitive.

-private=data
Site-specific account data for the new account can be stored in the account profile file using this switch. The data string can not exceed a length of 64 bytes. This data is not used by the popstore itself but can be used by site-developed procedures which access account profiles.

-prompt ***(default)***
-noprompt
By default if a wildcard is used, even if `-noconfirm` is specified, one confirmation prompt is issued. If `-noprompt` is specified, there is no prompting at all.

-pwdexpired
-nopwdexpired ***(default)***
If `-pwdexpired` is specified, then the account is marked as pre-expired. This means that if password expiration is enabled through the `PASSWORD_LIFETIME` option, then the user must change their password immediately.

If `-nopwdexpired` is specified (the default), then the account is marked as not pre-expired. The time of last password change is set to the current time. If password expiration is enabled, then the user does not have to change the password until the `PASSWORD_LIFETIME` has run out.

-quota=value
-noquota
The new account's message storage quota. The account can continue to receive new messages so long as the storage consumed by its currently stored messages does not exceed its message storage quota. See also `-overdraft`.

A quota value of zero, conveys unlimited storage. That is, to grant an account unlimited storage set its quota to zero. Specifying `-noquota` is equivalent to `-quota=0`.

By default, this quantity is specified in units of kbytes; however, the `set storage_units` command can be used to change the units used.

The maximum value is 4 gigabytes minus 1. If the value specified exceeds the maximum, the value is set to zero (unlimited quota).

EXAMPLES

To create a new account `jdoe` for Jane Doe which duplicates the account `bsmith` but has different owner and password fields, use the command

```
popstore> copy bsmith jdoe -password=SeCrEt -owner="Jane Doe"
```

delete—Remove a user account or a user's messages

Remove user accounts from the popstore or delete users' stored messages.

SYNTAX

delete *username*

Command Switches

-confirm	
-group=name	
-log	
-messages	
-prompt	
-return	*-noreturn*

PARAMETERS

username
Name of the account to delete. Can contain wild card characters.

DESCRIPTION

Use the `delete` command to remove a user account. By default, stored messages for the account are deleted silently. To cause unread messages to be returned to their originator as undelivered, specify `-return`.

Use the `-messages` switch to delete or return a user's messages. The account itself will not be deleted.

When the username parameter contains wild card characters, all matching accounts within the manager's management group and subgroups thereof will be deleted. The `-group` switch can be used to further constrain which accounts are deleted.

COMMAND SWITCHES

`-confirm`
`-noconfirm`
Prompt for positive confirmation before carrying out the indicated operation. When wild cards are not used, `-noconfirm` is the default. When wild cards are used, `-confirm` is the default and a prompt is issued for each account to be operated upon. Moreover, when wild cards are used, `-noconfirm` causes only a single prompt to be issued—it does not eliminate the prompt altogether.

-group=name
Name of a management group to constrain the operation to. This switch can be used in conjunction with a username parameter containing wild card characters so as to further constrain the delete operation.

-log
-nolog
When the operation is successful, output a status message stating that the operation succeeded. Note that error messages are always indicated. `-nolog` is the default behavior unless wild card characters are used in which case `-log` is the default.

-messages
When the `-messages` switch is specified, only the user's messages are deleted or returned. The account itself is not deleted.

-prompt ***(default)***
-noprompt
By default if a wildcard is used, even if `-noconfirm` is specified, one confirmation prompt is issued. If `-noprompt` is specified, there is no prompting at all.

-return
-noreturn ***(default)***
When `-return` is specified, unread messages are returned to their originator as undelivered. By default unread messages are deleted without sending a non-delivery notice back to their originators.

EXAMPLES

To delete the account `jdoe`, issue the command

```
popstore> delete jdoe
```

exit—Exit the utility

Exit the utility.

SYNTAX

exit

Command Switches

None

PARAMETERS *None.*

DESCRIPTION The `exit` command exits the utility.

forward—Establish a forwarding address

Establish a forwarding address.

SYNTAX

forward *username forward-to-address*

Command Switches

-override	*-override*

PARAMETERS

username
Username for which to establish a forwarding address.

forward-to-address
Address to which to forward messages. Must be a single, fully-qualified RFC822 address—specifically, an RFC822 "addr-spec".

DESCRIPTION

Messages destined for the popstore can be automatically forwarded to a different popstore addressee or to another address outside of the popstore altogether. This is done by establishing a forwarding address with the forward command. For instance, to forward all mail for the popstore user `sandy` to the address `sandra@example.com`, issue the command

```
popstore> forward sandy sandra@example.com
```

The username supplied (*e.g.*, `sandy`) need not correspond to an actual popstore account.

Note that if more than one forwarding address is supplied, then each address should be separated by commas and all the addresses enclosed in a set of double quotes. For example,

```
popstore> forward sandy "sandy, sandra@example.com"
```

When a forwarding address is established for an actual popstore user, that user will no longer receive mail in the popstore unless the forwarding includes their account in the list of addresses to forward to. For instance, the first example above would cause the account `sandy` to no longer receive into the popstore mail sent to it. The mail is instead directed to sandra@example.com. In the second example, however, mail will still be stored into the popstore for the account `sandy`. In addition, a copy of the mail will be forwarded to `sandra@example.com`.

COMMAND SWITCHES

`-override` ***(default)***
`-nooverride`
By default, forwarding addresses can be established for existing popstore users. Specify `-nooverride` to prevent inadvertently forwarding an existing user's messages elsewhere.

A manager can not establish a forwarding address which will override a popstore account outside of their own management group.

group—Manipulate management groups

Manipulate management groups.

SYNTAX

group -add *[group-name [subgroup-name[,...]]]*
group -delete *group-name*
group -list *[group-name]*
group -modify *group-name [subgroup-name[,...]]*

Command Switches

-add
-confirm
-delete
-format_file=file-spec
-list
-log
-modify
-output=file-spec
-prompt
-recur

PARAMETERS

group-name
Name of the group to add, delete, list, or modify. Wild cards can be used in conjunction with the `-list` switch.

subgroup-name[,...]
A comma separated list group names to associated with the group being added or modified. The listed groups will become subgroups of the group being added or modified.

DESCRIPTION

The `group` command is used to manipulate the popstore management groups. Only managers with either operating system privileges or a privileged popstore account with access to the `world` group can use this command. In regards to the latter case, that means that the manager's account must have the `MANAGE` usage flag set and either have no group name associated with the account—the empty group—or be in a management group which contains as a subgroup the `world` group. The one exception to this rule is that a manager can always use the `-list` switch to list their own management group and subgroups thereof.

For further details on the usage of this command as well as usage examples, see Section 6.12.

COMMAND SWITCHES

-add
This switch indicates that a new management group is to be added. If a group already exists with the same name, then an error will be output.

-confirm
-noconfirm ***(default)***
Prompt for positive confirmation before carrying out the indicated operation. `-noconfirm` is the default behavior.

-delete
This switch indicates that the specified management group is to be deleted. Note that subgroups contained within the group are not deleted unless `-recur` is also specified. Moreover, the actual accounts in the group are not deleted either. They can only be deleted with a `delete -group=`*`group_name`* `*` command.

-format_file=file-spec
Specify a formatting file to use to format the output of a `group -list` command.

-list
List the specified groups and subgroups. When this switch is used, the `group-name` parameter can contain wild card characters. When the parameter is omitted, `*` is assumed.

-log
-nolog ***(default)***
When the operation is successful, output a status message stating that the operation succeeded. Note that error messages are always indicated. `-nolog` is the default behavior.

-modify
Modify the specified group, replacing its list of subgroups with the specified list. If no list is specified, then the group is changed to contain no subgroups.

-output=file-spec
Write the output to the specified file rather than to the terminal. The file will be created as a new file each time it is specified.

-prompt ***(default)***
-noprompt
By default if a wildcard is used, even if `-noconfirm` is specified, one confirmation prompt is issued. If `-noprompt` is specified, there is no prompting at all. This qualifier can be used in conjunction with the `-add`, `-delete`, or `-modify` switches.

-recur
-norecur ***(default)***
This qualifier can be used in conjunction with the `-delete` switch. By default, only the specified group is deleted. Subgroups of that group are not deleted unless `-recur` is also specified.

login—Activate management privileges

Activate management privileges.

SYNTAX

login *[username]*

Command Switches

None

PARAMETERS

username
Name of the account under which to log in.

DESCRIPTION

Popstore users who have popstore management privileges but lack operating system privileges must log in to their account with the `login` command in order to perform management operations. Once logged in, the utility will then allow the user to perform management operations. Users who have operating system privileges need not log in to their account.

To log in to a popstore account, the popstore account's username should be supplied using the username parameter to the `login` command. If the username parameter is omitted, the utility will use the name of the operating system account under which the user is logged in. The utility will then prompt for a password. If the correct password for the popstore account is supplied, and the account has the manage flag set, then the utility will allow management operations to be undertaken using the utility's image privileges.

A popstore account is granted management privileges by specifying

```
-flags=manage
```

when creating or modifying it with the `add` or `modify` commands.

See also the `logout` command.

EXAMPLES

To log in to the account `bob`, issue the command

```
popstore> login bob
Password: santaclaus
Login succeeded; management capabilities enabled
```

logout—Deactivate management privileges

Deactivate management privileges.

SYNTAX

logout

Command Switches

None

PARAMETERS *None.*

DESCRIPTION Use the `logout` command to deactivate privileges activated with the `login` command. Note that management privileges are also deactivated when the utility is exited.

modify—Change an existing account

Change characteristics of one or more existing accounts.

SYNTAX

modify *username*

Command Switches

-confirm
-flags=flags
-group=name
-group_name=name
-last_connect
-last_disconnect
-log
-message_count=value
-overdraft=value
-owner=owner
-password=password
-past_block_days=value
-private=data
-prompt
-pwdexpired
-quota=value
-received_bytes=value
-received_messages=value
-total_connect=value
-total_connections=value

PARAMETERS

username
Name of the account for which to make the modifications. Can contain wild card characters.

DESCRIPTION

The `modify` command changes one or more characteristics of an existing account. Characteristics not specified with switches in the command are left unchanged.

When the username parameter contains wild card characters, all matching accounts within the manager's management group and subgroups thereof will be modified. The `-group` switch can be used to further constrain which accounts are modified.

COMMAND SWITCHES

-confirm
-noconfirm
Prompt for positive confirmation before carrying out the indicated operation. When wild cards are not used, `-noconfirm` is the default. When wild cards are used, `-confirm` is the default and a prompt is issued for each account to be operated upon. Moreover, when wild cards are used, `-noconfirm` causes only a single prompt to be issued.

-flags=(flag[,...])
Change the usage flags associated with the account. The recognized flags are as follows:

`dismail`	User is not allowed to receive new mail messages.
`disuser`	User is not allowed to access their account.
`lockpwd`	User is not allowed to change their password.
`manage`	User is allowed to manage popstore accounts.
`migrated`	Internal flag used by the PMDF migration utilities.
`pwd_elsewhere`	Password information is stored outside of the popstore.
`nodismail`	User is allowed to receive new mail messages.
`nodisuser`	User is allowed to access their account.
`nolockpwd`	User is allowed to change their password.
`nomanage`	User is not allowed to manage popstore accounts.
`nomigrated`	Internal flag used by the PMDF migration utilities.
`nopwd_elsewhere`	Password information is stored within the popstore.

-group=name
Name of a management group to constrain the operation to. This switch can be used in conjunction with a username parameter containing wild card characters so as to further constrain the modify operation.

-group_name=name
Change the accounts to be in the specified management group. A manager can not change an account's management group to be a group outside of the manager's group.

-last_connect
Clear the user's last connect time field.

-last_disconnect
Clear the user's last disconnect time field.

-log
-nolog
When the operation is successful, output a status message stating that the operation succeeded. Note that error messages are always indicated. `-nolog` is the default behavior.

-message_count=value
Reduce the user's message count to the specified value, deleting stored messages if necessary. The act of deleting stored message will change the past block days field.

-overdraft=value
-nooverdraft
Change the account's overdraft quota which is the amount of message storage by which the account can exceed its primary message storage quota. By default, this quantity is specified in units of kbytes; however, the `set storage_units` command can be used to change the units used.

Specifying `-nooverdraft` is equivalent to specifying `-overdraft=0`.

The maximum value is 4 gigabytes minus 1. If the value specified exceeds the maximum, the value is set to zero (no overdraft quota).

-owner=owner
Change the accounts ownership field. The length of the string can not exceed 40 bytes. The owner field is not used by the popstore itself; it is generally used by humans to associate account usernames with the actual owner of the account.

-password=password
-nopassword
Change the account's password. The length of the password can not exceed 32 bytes. Access by non-managers to the account requires knowledge of this password. For instance, to access the account from a POP3 client, the correct username and password associated with the account must be supplied.

When `-nopassword` is specified, the account is changed to not require any password in order to access it.

Note that passwords are case sensitive.

-past_block_days=value
Set the user's past block days field to the specified, integer value. Changing this value clears the past block days remainder field.

-private=data
Change the site-specific account data stored in the account profile file. The data string can not exceed a length of 64 bytes. This data is not used by the popstore itself but can be used by site-developed procedures which access account profiles.

-prompt ***(default)***
-noprompt
By default if a wildcard is used, even if `-noconfirm` is specified, one confirmation prompt is issued. If `-noprompt` is specified, there is no prompting at all.

-pwdexpired
-nopwdexpired
If `-pwdexpired` is specified, then the account is marked as pre-expired. This means that if password expiration is enabled through the `PASSWORD_LIFETIME` option, then the user must change their password immediately.

If -nopwdexpired is specified, then the account is marked as not pre-expired. The time of last password change is set to the current time. If password expiration is enabled, then the user does not have to change the password until the PASSWORD_LIFETIME has run out.

The default is to not change the pre-expired status of the account.

-quota=value
-noquota
Change the account's message storage quota. The account can continue to receive new messages so long as the storage consumed by its currently stored messages does not exceed its message storage quota. See also -overdraft.

A quota value of zero, conveys unlimited storage. That is, to grant an account unlimited storage set its quota to zero. This can be done by specifying either -quota=0 or -noquota.

By default, this quantity is specified in units of kbytes; however, the set storage_units command can be used to change the units used.

The maximum value is 4 gigabytes minus 1. If the value specified exceeds the maximum, the value is set to zero (unlimited quota).

-received_bytes=value
Set the cumulative count of received message bytes to the specified, integer value. By default, this quantity is specified in units of kbytes; however, the set storage_units command can be used to change the units used. The maximum value is 4 gigabytes minus 1. If the value specified exceeds the maximum, the value is set to zero.

-received_messages=value
Set the cumulative count of received messages to the specified, integer value.

-total_connect=value
Set the user's total connect field to the specified, integer value.

-total_connections=value
Set the user's count of total connections to the specified, integer value.

EXAMPLES

In the following example, the quota and password fields are changed for the user jdoe:

```
popstore> modify jdoe -password=TodaY -quota=20000
```

noforward—Remove a forwarding address

Remove a forwarding address.

SYNTAX

noforward *username*

Command Switches

None

PARAMETERS

username
Username for which to remove the forwarding.

DESCRIPTION

Forwarding addresses are removed with the `unforward` command. If the supplied username also matches an existing popstore account, then that account will resume receiving new mail messages.

quit—Exit the utility

Exit the utility.

SYNTAX

quit

Command Switches

None

PARAMETERS *None.*

DESCRIPTION The `quit` command exits the utility. The `quit` command is a synonym for the `exit` command.

rename—Rename an account

Change the username associated with an account (popstore only).

SYNTAX

rename *old-username new-username*

Command Switches

-confirm
-log
-prompt

PARAMETERS

old-username
The old name of the account.

new-username
The new name for the account.

DESCRIPTION

The `rename` command changes the username associated with a popstore account. Once an account is renamed, it can no longer receive mail under the old name unless a forwarding from the old name to the new name is also established with the `forward` command.

COMMAND SWITCHES

`-confirm`
`-noconfirm` ***(default)***
Prompt for positive confirmation before carrying out the indicated operation. `-noconfirm` is the default behavior.

`-log`
`-nolog` ***(default)***
When the operation is successful, output a status message stating that the operation succeeded. Note that error messages are always indicated. `-nolog` is the default behavior.

`-prompt` ***(default)***
`-noprompt`
By default if a wildcard is used, even if `-noconfirm` is specified, one confirmation prompt is issued. If `-noprompt` is specified, there is no prompting at all.

EXAMPLES

To rename the popstore account `jdoe` to `janedoe`, issue the command:

```
popstore> rename jdoe janedoe
```

run—Run commands from a file

Execute commands from a file.

SYNTAX

run *file-spec*

Command Switches
-ignore
-log

PARAMETERS

file-spec
Name of the file to read commands from.

DESCRIPTION

The `run` command reads lines from the specified file and then interprets the lines as commands and executes them. Any command recognized by this utility can be specified, including the `run` command itself. By default, if an error is encountered, it is reported and processing of the file is aborted. Specify `-ignore` to cause processing to continue when errors are encountered and reported. The `-log` switch command can be used to cause the commands to be echoed to the terminal as they are executed.

COMMAND SWITCHES

`-ignore`
`-noignore` ***(default)***
By default, processing of commands from the input file is aborted when an error is encountered. This behavior corresponds to the `-noignore` switch. Specify `-ignore` to cause command processing to not abort when an error occurs.

`-log`
`-nolog` ***(default)***
When `-log` is specified, each command is echoed to the terminal as it is executed.

EXAMPLES

To process the commands from the file `doit.com`, issue the command

```
popstore> run doit.com
```

To cause the commands to be echoed to the screen, instead use the command

```
popstore> run -log doit.com
```

set domain—Set user domain

Select the user domain to manage.

SYNTAX

set domain *domain-name*

Command Qualifiers

None

PARAMETERS

domain-name

Name of the user domain to manage.

DESCRIPTION

By default, the `default` user domain is managed with this utility. To manage a different user domain, select that domain with the `set domain` command.

For example, to manage the `example.com` domain, specify

```
popstore> set
Using the "default" user domain
...
popstore> set domain example.com
popstore> set
Using the "example.com" user domain
...
popstore>
```

set storage_units—Set storage units

Set the units used to measure byte-counted values.

SYNTAX

set storage_units *type*

Command Switches
None

PARAMETERS

type
Type of units to use. Must be one of `bytes`, `kbytes`, `mbytes`, or `gbytes`.

DESCRIPTION

By default, units of `kbytes` (1024 bytes) are used when specifying values for byte-count valued fields such as message quotas.

To select a unit of measure other than `kbytes`, use the `set storage_units` command. `bytes` selects bytes, `kbytes` selects 1024 bytes, `mbytes` selects 1024 `kbytes`, and `gbytes` selects 1024 `mbytes`.

After issuing a `set storage_units` command, all byte-count valued numbers input on the command line will be interpreted as being measured in the newly selected units. Note that displayed values are displayed in the units called for by the formatting template used to generate the display.

For example, to use units of `mbytes`, specify

```
popstore> set storage_units mbytes
```

set time_units—Set time units

Set the units used to measure time-valued fields.

SYNTAX

set time_units *type*

Command Switches

None

PARAMETERS

type
Type of units to use. Must be one of `seconds`, `minutes`, `hours`, or `days`.

DESCRIPTION

By default, time units of `days` are used when specifying values for time-valued fields. Presently, the only time-valued field is the total connect time field which can be modified with the `-total_connect` switch of the `modify` command.

To select a unit of measure other than `days`, use the `set time_units` command. After issuing a `set time_units` command, all time-valued numbers input on the command line will be interpreted as being measured in the newly selected units. Note that displayed values are displayed in the units called for by the formatting template used to generate the display. For example, to use units of `hours`, specify

```
popstore> set time_units hours
```

show—Show information about user profiles

Display user accounts.

SYNTAX

show *username*

Command Switches

-all
-brief
-count_users
-domains
-format_file=file-spec
-forwardings
-group=name
-messages
-output=file-spec
-store=store-type

PARAMETERS

username
Name of the account for which to display information. Wild cards are permitted.

DESCRIPTION

The `show` command shows settings for one or more user profiles, displays established forwarding addresses, and lists information about messages received by users. The `username` parameter can contain wild cards when displaying account information; it can not contain wild cards when listing forwardings.

Use the `show -forwardings` and `show -domains` commands to generate listings of, respectively, user e-mail forwardings and user domains.

Use the `show -count_users` command to list the number of currently defined accounts as well as any licensing limits.

COMMAND SWITCHES

`-all`
By default, only popstore accounts are displayed: MessageStore and native accounts are not displayed. Specify `-all` to list all accounts. Note that `-all` and `-store=all` are synonyms.

-brief
Generate a brief profile or message listing. By default, the formatting file `popmgr_profile_brief.txt` is used to format the output for profile displays and `popmgr_messsage_brief.txt` for message displays. This switch can not be used in conjunction with the `-forwardings` switch.

-count_users
Display the number of currently defined user accounts as well as the number allowed by your PMDF-POPSTORE license. Specify -all to see both the popstore and MessageStore counts.

-domains
Generate a list of defined user domains. By default, the formatting file `popmgr_domains.txt` is used to format the output.

-format_file=file-spec
Specify a formatting file to use to format the output.

-forwardings
Display information about established forwarding addresses. By default, the formatting file `popmgr_forward.txt` is used to format the output.

-group=name
Confine the listing to the specified management group and its subgroups.

-full ***(default)***
Generate verbose output. By default, the formatting file `popmgr_profile.txt` is used to format profile information; `popmgr_message.txt` for message listings; `popmgr_domains.txt` for domain listings; and, `popmgr_forward.txt` for forwarding addresses. Those formatting files are found with the other formatting files in the /pmdf/www/popstore/ directory tree.

-messages
Display information on the users' messages. By default, the formatting file `popmgr_message.txt` is used to format the display.

-output=file-spec
Write the output to the specified file rather than to the terminal. The file will be created as a new file each time it is specified.

-store=store-type
By default, only popstore accounts are displayed: MessageStore and native accounts are not displayed. Specify `-store=all` to list all accounts; `-store=msgstore` or `-store=imap` to list only MessageStore accounts; `-store=popstore` or `-store=pop` to list only popstore accounts; and, `-store=native` to list only profiles marked as being native.

EXAMPLES

In the following example, full and brief listings are generated for the `default` popstore account:

```
popstore> show default

Username:        default
Owner:           Default user profile
Group:
Store Type:     popstore
Usage flags:
Site-defined:

Last pwd change:    No time recorded
Last connect:       No time recorded
Last disconnect:    No time recorded
Total connect time: 0 00:00:00
Total connections:  0
Past block days:    0
Last billing:       Fri Nov 15 10:23:54 2012

Message count:              0 (0 total messages received)
Quota used:              0.00 Kbytes
Quota:                1024.00 Kbytes
Overdraft:              51.00 Kbytes

popstore> show -brief default
                                         Quota  Message  Quota used
 Username                              (kbytes)   Count    (kbytes)
 ----------------------------------------------------------------
 default                                1024.00        0       0.00
 ----------------------------------------------------------------
*Note: privileged users are flagged with an asterisk
```

test—Test site-supplied subroutines

Test optional, site-supplied subroutines to verify that they load and function correctly.

SYNTAX

test -block_days *image-spec starting-time ending-time size remainder*
test -connect *image-spec starting-time ending-time*
test -message_mapping *image-spec*
test -paths *path-file-spec*
test -profile_mapping *image-spec*

Command Switches

-block_days
-connect
-message_mapping
-paths
-profile_mapping

PARAMETERS

image-spec
File specification for the shared image containing the subroutine to test.

starting-time
Starting time value to pass to the `compute_connect` or `compute_block_days` subroutine.

ending-time
Ending time value to pass to the `compute_connect` or `compute_block_days` subroutine.

size
Size value to pass to the `compute_block_days` subroutine.

remainder
Remainder value to pass to the `compute_block_days` subroutine.

path-file-spec
File specification for the file of directory paths to check.

DESCRIPTION

The `test` command provides a mechanism to test site-supplied subroutines intended for use with the popstore. The purpose and usage of those subroutines is described in Chapter 14.

The `test -message_mapping` and `test -profile_mapping` commands test, respectively, `map_message_filename` and `map_profile_filename` subroutines. The command will load the subroutine from the specified image and then, for each stored message or profile file, run the filename through the subroutine. The input and output file names for each file will be displayed along with diagnostic information, should an error occur.

The `test -connect` and `test -block_days` commands test, respectively, the `compute_connect` and `compute_block_days` subroutines. With each command, you can specify the values of the input arguments to be passed to those subroutines. The results produced by the subroutine will then be displayed. Should an error occur, diagnostic information will be displayed.

Text files intended for use as a `/pmdf/table/popstore_message_paths` or `/pmdf/table/popstore_profiles_paths` file can be tested with the `test -paths` command. The command will scan the directory trees listed in the specified file, displaying the files found in each directory.

COMMAND SWITCHES

`-block_days`
Test the `compute_block_days` subroutine from the shared image **image-spec**.

`-connect`
Test the `compute_connect` subroutine from the shared image **image-spec**.

`-message_mapping`
Test the `map_message_filename` subroutine from the shared image **image-spec**.

`-profile_mapping`
Test the `map_profile_filename` subroutine from the shared image **image-spec**.

`-paths`
List the files from the directory trees specified in the path file **path-file-spec**.

7 OpenVMS Command Line Management Utility

The command line management utility is an interactive, command oriented interface for managing popstore and MessageStore accounts. Users with operating system privileges as well as users with privileged popstore or MessageStore accounts can use the utility. Also, the utility can be used as a report generator as described briefly in Section 7.15.2 and more completely in Chapter 9.

To run the utility, issue the DCL command

```
$ PMDF POPSTORE

or

$ PMDF MSGSTORE
```

Use the `EXIT` or `QUIT` command to exit the utility. Table Table 7–1 summarizes the recognized commands; see Section 7.18 for complete command descriptions.

Table 7–1 Summary of command line management commands (OpenVMS)

Command	Description
ADD	Add new user accounts
COPY	Copy user accounts (popstore only)
DELETE	Delete user accounts or messages
EXIT	Exit the utility
FORWARD	Establish forwarding addresses
GROUP	Manipulate management groups
LOGIN	Enable privileges by ‘‘logging in’’ to a privileged popstore/msgstore account
LOGOUT	Disable privileges
MODIFY	Modify user accounts
NOFORWARD	Remove forwarding addresses
QUIT	Exit the utility
RENAME	Rename user accounts (popstore only)
SET	Select user domain or set units used for expression of storage and time
SHOW	Display information about user accounts, messages, or forwardings

7.1 Basic Operation

Popstore and MessageStore accounts are managed using four basic commands: `ADD`, `DELETE`, `MODIFY`, and `SHOW`. These four commands add accounts, remove accounts, modify accounts, and display information about accounts. For popstore accounts, two additional commands, `COPY` and `RENAME`, can be used to create new accounts which look like existing accounts and to change the name of an existing account.

When an account is created with the ADD command, you can also specify various account settings such as the account password, the name of the account's owner, and storage quotas. Once the account is created, you can subsequently change these settings with the MODIFY command.

Some of the account settings involve the use of values expressed in units of storage or time. By default, the units of storage are kbytes (1024 bytes), and the units of time are days. Within a session with the utility, these units can be changed with the SET command; see the command descriptions in Section 7.18 for details.

7.2 Adding New Accounts

New accounts can be added in one of two ways: by creating a new account or (for popstore only) copying an existing account. The former is done with the ADD command while the latter with the COPY command. Regardless of the method chosen, when a new account is added a name and password to associate with the account should be chosen. This is the name and password which the owner of the account must use in order to access messages stored for the account. Specify -flags=pwd_elsewhere to create the account with an externally stored password (*e.g.*, an /etc/passwd password). Optionally, account quotas Optionally, account quotas and an ownership field identifying the owner of the account can also be specified. Quotas not specified will be copied from the default account.

See Section 1.3.1 for a discussion of the name space allowed for account usernames.

For instance, suppose that Jane Doe is to be given the account jdoe with the password SecRet. The command to add the account would then be

```
popstore> ADD JDOE/PASSWORD="SecRet"/OWNER="Jane Doe"
popstore> SHOW JDOE
Username:          jdoe
Owner:             Jane Doe
Group:
Store Type:     popstore
Usage flags:
Site-defined:

Last pwd change:    Fri Nov 15 12:02:22 2012
Last connect:       No time recorded
Last disconnect:    No time recorded
Total connect time: 0 00:00:00
Total connections:  0
Past block days:    0
Last billing:       Fri Nov 15 12:02:22 2012

Message count:              0 (0 total messages received)
Quota used:              0.00 Kbytes
Quota:                2048.00 Kbytes
Overdraft:              20.00 Kbytes
```

Note that POP passwords are case sensitive. Moreover, the utility will lower case strings not placed within quotes. As such, it is necessary to quote strings such as passwords which contain upper case characters which must be preserved.

It is important to note that when the popstore was configured, default account settings were specified. These settings are kept in the form of a default account named `default`. Whenever an account is added with the ADD command, the popstore actually copies the `default` account to the new account thereby using the `default` account's settings as the basis for the new account.

Copying an existing popstore account with the `COPY` command also creates a new account. In the example below, the new account `jones` is created by copying the account `adams`:

```
popstore> SHOW ADAMS

Username:        adams
Owner:           David Adams
Group:
Store Type:     popstore
Usage flags:
Site-defined:

Last pwd change:    Fri Oct 18 09:12:23 2012
Last connect:       Fri Nov 15 16:09:38 2012
Last disconnect:    Fri Nov 15 16:09:39 2012
Total connect time: 0 00:10:03
Total connections:  145
Past block days:    123953
Last billing:       Fri Oct 18 09:12:23 2012

Message count:              0 (189 total messages received)
Quota used:              0.00 Kbytes
Quota:              102400.00 Kbytes
Overdraft:              10.00 Kbytes

popstore> COPY ADAMS JONES/PASSWORD=SECRET/OWNER="Daniel Jones"
popstore> SHOW JONES

Username:           jones
Owner:              Daniel Jones
Group:
Store Type:     popstore
Usage flags:
Site-defined:

Last pwd change:    Fri Nov 15 13:23:18 2012
Last connect:       No time recorded
Last disconnect:    No time recorded
Total connect time: 0 00:00:00
Total connections:  0
Past block days:    0
Last billing:       Fri Nov 15 13:23:18 2012

Message count:              0 (0 total messages received)
Quota used:              0.00 Kbytes
Quota:              102400.00 Kbytes
Overdraft:              10.00 Kbytes
```

Note that when an account is copied to make a new account, the new account does not inherit the messages or accounting information of the original account.

Note: Your PMDF-POPSTORE license controls the number of popstore user accounts which you can have at any one time. When you reach this limit, you will not be allowed to create additional accounts without first deleting some accounts or obtaining a new license with an increased limit. Sites without a PMDF-POPSTORE license are allowed to use the popstore and create up to ten user accounts. This limit does not include the `default` account. Use the `SHOW/COUNT_USERS` command to display the number of currently defined accounts as well as the limit allowed by your license.

7.3 Listing Accounts

Verbose and brief listings of accounts can be generated with the `SHOW` command. When no username parameter is supplied, all accounts are listed:

```
popstore> SHOW/BRIEF
                                      Quota  Message  Quota used
 Username                           (kbytes)    Count    (kbytes)
 ---------------------------------------------------------------
 anne                               33333.00        0        0.00
 david                             102400.00        0        0.00
*deanna                             33333.00        0        0.00
 default                             1024.00        0        0.00
 kevin                              40960.00        0        0.00
 karen                              10240.00        0        0.00
 marty                              40960.00        0        0.00
 ---------------------------------------------------------------
*Note: privileged users are flagged with an asterisk
```

In the above output, an asterisk is displayed by each account which has the management privilege. The `/BRIEF` qualifier, causes a brief listing to be generated. Omitting that qualifier generates a much more detailed listing.

When a `username` parameter is specified, wild cards can be used:

```
popstore> SHOW/BRIEF D*
                                      Quota  Message  Quota used
 Username                           (kbytes)    Count    (kbytes)
 ---------------------------------------------------------------
 david                             102400.00        0        0.00
*deanna                             33333.00        0        0.00
 default                             1024.00        0        0.00
 ---------------------------------------------------------------
*Note: privileged users are flagged with an asterisk
```

As discussed in Section 7.15.2, the `/FORMAT` qualifier of the `SHOW` command can be used to generate custom listings. Section 7.11 describes how to use the `/FORWARDINGS` qualifier to display mail forwardings.

7.4 Modifying Accounts

The MODIFY command can be used to change fields associated with an account. For instance, to reset accounting information, change a password, or to increase or decrease an account quota. In the following example, the password for the account jdoe is changed to MORGAN:

```
popstore> MODIFY JDOE/PASSWORD="MORGAN"
```

The MODIFY command does not accept wild cards and the username parameter must be specified.

Note that the last connect and disconnect time accounting fields can only be reset to signify a "never connected" state. They can not be changed to an arbitrary time. Also note that when the message count is set to zero, any messages stored for the account are deleted. If the count is reduced but not set to zero then starting with the oldest message, messages are deleted until the desired count is reached. For instance, if the current message count is 5 messages and the count is set to 2 messages, then messages 1, 2, and 3 will be deleted.

7.5 Removing Accounts

Accounts are removed with the DELETE command. By default, any messages stored for the account are deleted silently. Use the /RETURN qualifier to return as undelivered any unread messages.

In the example below, the DELETE command is used to delete the jdoe account.

```
popstore> SHOW/BRIEF JDOE
                                         Quota  Message  Quota used
 Username                              (kbytes)   Count    (kbytes)
 -----------------------------------------------------------------
 jdoe                                   1000.00        2        6.42
 -----------------------------------------------------------------
*Note: privileged users are flagged with an asterisk

popstore> DELETE/RETURN JDOE
popstore> SHOW/BRIEF JDOE
%POPMGR-E-CANTSHOW, cannot show data
        popstore error #68: No such user
```

To delete more than one account, issue multiple DELETE commands, or use wild cards in the username parameter, or use the /GROUP qualifier. For instance, to delete all accounts in the class97 group, issue the command

```
popstore> DELETE/GROUP=class97
```

7.6 Changing New Account Defaults

Recall that when a new account is created with the ADD command, the default account is used as a source of defaults for the new account. Thus, by changing the settings for the default account with the MODIFY command, you change the defaults which will be applied to subsequently created accounts. For instance, to change the default account quota to 100 kbytes with an overdraft quota of 10 kbytes, you would modify the default account accordingly:

```
popstore> MODIFY DEFAULT/QUOTA=100/OVERDRAFT=10
```

Note that the settings for the default account only applies to new accounts created with the ADD command. The default account settings do not apply to all existing accounts.

7.7 Bulk Loading Accounts

To create many accounts at once, use the @ command to execute commands from a command file. Before using that command, first create a file containing commands with one command per line. For instance,

```
$ TYPE A.COM
ADD BJAMES/OWNER="Bob James"/PASSWORD=837271
ADD CSMITH/OWNER="Cathy Smith"/PASSWORD=382374
ADD RBROWN/OWNER="Randy Brown"/PASSWORD=383838
...
$
```

Then use the @ command to execute the commands from the file:

```
$ PMDF POPSTORE
popstore> @A.COM
popstore> ADD BJAMES/OWNER="Bob James"/PASSWORD=837271
popstore> ADD CSMITH/OWNER="Cathy Smith"/PASSWORD=382374
popstore> ADD RBROWN/OWNER="Randy Brown"/PASSWORD=383838
popstore> ...
popstore> QUIT
$
```

Each command will be read from the specified file and executed. If an error occurs, it will be reported and processing of the file will continue. Each command from the file will be echoed as it is executed. Use the @@ command to suppress the command echo,

```
$ PMDF POPSTORE
popstore> @A.COM
popstore> QUIT
$
```

Any command recognized by this utility can be used in the command file, including @ and @@ commands.

7.8 Returning or Deleting Messages

A user's stored messages can be deleted with the DELETE/MESSAGES command. When the /RETURN qualifier is also specified, any unread messages are returned as unread to their sender. For instance, to delete the messages for the account jdoe, returning any unread messages, use the command

```
popstore> DELETE/MESSAGES/RETURN JDOE
```

Note that the /RETURN qualifier can also be used with the DELETE command as described in Section 7.5.

7.9 Account Quotas

Account quotas are used to control how much mail a user can store. Each account has a primary quota and an overdraft quota which are established when the account is created and can be changed with the MODIFY command. An account can not receive a new message if either the account's present storage exceeds the account's quota, or storage of the message would cause the account to exceed the sum of its quota and overdraft quota.[1]

Quotas are set and changed with the /QUOTA and /OVERDRAFT qualifiers of the ADD, COPY, and MODIFY commands. An account can be granted unlimited quota with the /NOQUOTA qualifier. By default, quotas are expressed in units of kbytes (1024 bytes). Alternate units can be selected with the SET STORAGE_UNITS command.

7.10 Blocking Access, Blocking New Mail, and Locking Passwords

There are three account flags which can be set to influence account access:

DISMAIL
The DISMAIL flag is used to prevent an account from receiving new mail messages. When this flag is set for an account, new messages are rejected and returned to their sender. The account owner can, however, read any existing messages they might have unless the account is also flagged with the DISUSER flag.

DISUSER
The DISUSER flag is used to deny access to an account. The account can, however, continue to receive new messages unless it is either over quota or also flagged with the DISMAIL flag.

LOCKPWD
The LOCKPWD flag prevents users from changing the account's password. The password can only be changed by a user with the management privilege or operating system privileges.

[1] See Section 1.3.3 for a discussion of the rationale behind the use of an overdraft quota.

These flags are set on an account with the `MODIFY/FLAGS` command. A flag can be negated by prefixing the name with `NO`; for example, `NODISMAIL`. In the following example, the `jdoe` account is marked `DISMAIL` and `DISUSER`:

```
popstore> MODIFY JDOE/FLAGS=(DISMAIL,DISUSER)
```

To undo those settings, issue the command:

```
popstore> MODIFY JDOE/FLAGS=(NODISMAIL,NODISUSER)
```

7.11 Forwarding Mail

Mail to the popstore can automatically be forwarded to other addresses. The addresses can be within the popstore or outside of it; the forwarding can be for actual popstore accounts or for non-existent accounts.

For example, suppose the host name associated with the popstore is naples.example.com. Suppose further that the popstore user `jdoe` would like her mail forwarded to the address jane.doe@naples.example.com. This forwarding would be accomplished with the `FORWARD` command:

```
popstore> FORWARD JDOE jane.doe@naples.example.com
popstore> SHOW/FORWARDINGS
Username                    Forwarding address
----------------------------------------------------------------------
jdoe                        jane.doe@naples.example.com
```

To forward mail for `jdoe` to herself and her assistant `aclarke@naples.example.com`, you would use the command

```
popstore> FORWARD JDOE "jdoe,aclarke@naples.example.com"
popstore> SHOW/FORWARDINGS
Username                    Forwarding address
----------------------------------------------------------------------
jdoe                        jdoe@naples.example.com, aclarke@naples.example.com
```

Use the `NOFORWARD` command to remove a forwarding address:

```
popstore> NOFORWARD JDOE
```

As mentioned earlier, forwarding can be established for non-existent accounts. For instance, suppose that the address `list@naples.example.com` is to be forwarded to a PMDF mailing list `list@example.com`. This is accomplished with the `FORWARD` command:

```
popstore> FORWARD list list@example.com
```

Note that currently stored messages are not affected by a forwarding address. Only new messages are affected: new incoming messages will be routed to the forwarding address and not stored. Note also that the presence of a forwarding address does not prevent a user from reading any stored messages which they might have.

7.12 Management Groups

Management groups are manipulated with the GROUP command. This command has four variants:

GROUP/ADD	Add a new management group
GROUP/DELETE	Delete a management group
GROUP/LIST	List one or more management groups and any subgroups
GROUP/MODIFY	Modify an existing management group

It is important to note that use of management groups is not required by the popstore. Moreover, when you place an account into a management group, that group is automatically created if it was not already defined. As such, you only need to use the GROUP command to establish group-subgroup relationships and to delete groups which are no longer being used.

Use the GROUP/ADD command to add a new management group. If the group already exists, then an error will ensue and the existing group left unchanged. In that case, use the GROUP/MODIFY command to modify the existing group.

To add the management groups class_97, class_98, class_99, and class_00 use the commands

```
popstore> GROUP/ADD class_97
popstore> GROUP/ADD class_98
popstore> GROUP/ADD class_99
popstore> GROUP/ADD class_00
```

To then make a new group named students containing these four groups, issue the command

```
popstore> GROUP/ADD STUDENTS CLASS_97,CLASS_98,CLASS_99,CLASS_00
```

The results can then be listed with the GROUP/LIST command as shown below.[2]

```
popstore> GROUP/LIST STUDENTS
      Group name:  Subgroups contained within
--------------------------------------------
        students:  class_97, class_98, class_99, class_00
        class_97:
        class_98:
        class_99:
        class_00:
popstore>
```

After the Class of 2010 has graduated and left and the Class of 2014 arrives, the class_10 group can be removed, the class_14 group added, and the students group modified as follows

[2] Note that group names are case insensitive and that the popstore converts all group names to lower case.

```
popstore> GROUP/ADD class_14
popstore> GROUP/MODIFY students class_11,class_12,class_13,class_14
popstore> GROUP/DELETE class_10
popstore> GROUP/LIST students
      Group name:  Subgroups contained within
--------------------------------------------
        students:  class_11, class_12, class_13, class_14
        class_11:
        class_12:
        class_13:
        class_14:
popstore>
```

Note that the GROUP/DELETE command deletes just the specified group. It will recursively delete the subgroups contained within the specified group only when /RECUR is specified. Also, it does not delete the accounts contained within the group; to do that, use the DELETE/GROUP=group_name * command; *e.g.*,

```
popstore> DELETE/GROUP=class_97/RETURN/LOG/NOCONFIRM *
```

It is also important to note that a group can contain only a limited number of subgroups as immediate subgroups. Those subgroups can, of course, contain other subgroups. The limit on the number of immediate subgroups of a given group is controlled by the length in bytes of the names of each of those immediate subgroups. If there are to be N immediate subgroups and their combined name lengths are L, then N+L must be less than 236. If a group needs to contain more subgroups than allowed by that limit, nest its definition an extra level. For example,

```
popstore> GROUP/ADD blah_x blah_01,blah_02,blah_03,...,blah_29
popstore> GROUP/ADD blah_y blah_30,blah_33,blah_34,...,blah_52
popstore> GROUP/ADD blah blah_x,blah_y
```

In the above example, the names blah_01, ..., blah_52 are collectively too long to be contained as immediate subgroups of a given group. Therefore they are split between the two groups blah_x and blah_y. Those two groups are then made subgroups of blah. Consequently, the group blah contains the all of the groups blah_01, ..., blah_52 as subgroups despite the length limitation.

7.13 User Domains

By default, all popstore accounts for a given installation share the same name space. This means that if the two distinct e-mail addresses sandy@example.com and sandy@example.org are directed to the same popstore, then e-mail to either of these addresses will be delivered to the same user account sandy. At some sites, however, it is useful to have distinct communities of users, each with their own name spaces. This is achieved with the popstore[3] through the use of “user domains”. Please see Section 1.3.5 for a basic discussion of popstore user domains. This section focuses on the mechanics of creating and managing user domains and assumes knowledge of the material presented in Section 1.3.5.

[3] The PMDF MessageStore does not at present support the use of user domains.

Note that all e-mail messages for all user domains are co-mingled in the same directory tree. That is, messages for one user domain are not stored distinct from other user domains.

7.13.1 Enabling User Domains

To enable the use of user domains, specify `USER_DOMAINS=1` in the popstore option file. See Section 3.4 for further details *including an important note for sites who are already associating more than one Internet host name with the popstore.*

Note: You must not use the `filter` channel keyword on the popstore or MessageStore delivery channel if `USER_DOMAINS=1` is set. So doing will cause the wrong filters to be used for users in domains other than the `default` domain.

7.13.2 Creating a New User Domain

New user domains are created with the `ADD/DOMAIN` command. For example, to create the user domain `example.org`, issue the command

```
popstore> ADD/DOMAIN example.org
```

This will create a new sub-directory tree in the popstore profiles directory. In that sub-directory tree, all profile files for the `example.org` user domain will be stored.

Issuance of the `ADD/DOMAIN` command also creates a `default` user account in the new user domain. This new account will be a copy of the `default` account from the `default` user domain.

A list of existing user domains can be obtained with the `SHOW/DOMAINS` command:

```
popstore> SHOW/DOMAINS
Cannonicalized domain name
--------------------------
default
example.org
```

7.13.3 Managing a User Domain

When you invoke the command-line management utility, it will be set to manage the `default` user domain. To manage a different user domain, you need to issue a `SET DOMAIN` command. This command tells the utility that all further commands will affect the specified user domain. For instance, to manage the `example.org` domain, issue the command

```
popstore> SET DOMAIN example.org
```

If you are logged in to a privileged management account at the time you issue the SET DOMAIN command, then you will automatically be logged out unless the management account was in the default user domain and not in any management group. This is a reflection of the management policy that only privileged accounts in the default user domain and in no management group can manage all accounts in all user domains.

Once the user domain has been selected, all subsequent commands will affect accounts in only that user domain. The default user domain can be re-selected with either the command "SET DOMAIN default" or, more succinctly, SET DOMAIN (no domain name parameter). The currently selected user domain can be shown by issuing the SET command without any parameters:

```
popstore> SET
Using the "default" user domain
...
popstore> SET DOMAIN example.org
popstore> SET
Using the "example.org" user domain
...
```

Note that presently, no commands are provided to copy or rename an account between user domains. The popstore API does, however, provide this functionality via the POPSTORE_user_copy_d subroutine.

7.13.4 Deleting a User Domain

Owing to the rarity of the event, no single command is provided with which to delete an existing user domain. To delete an existing user domain, first select that domain and then delete every account within the domain. This ensures that messages for the account are deleted and that license limits are correctly credited for the deleted accounts. For example,

```
popstore> SET DOMAIN example.org.
popstore> DELETE/NOCONFIRM *
Delete the specified accounts [Yes/No/Quit]? yes
User account "aalan" deleted
User account "aabe" deleted
...
```

Then, delete the example.org.dir directory in the popstore profiles directory (usually, PMDF_POPSTORE_PROFILES:[000000]).

7.14 Use of the Utility by Non-privileged Users

The command line utility requires operating system privileges in order to operate. So as to control who can or cannot use the utility, the utility requires that the operator either have the necessary operating system privileges or that they have a popstore account which has the MANAGE flag set. (The manage flag is also required to use the web-based management interface.) Thus, a user lacking operating system privileges can use the utility to manage the popstore provided that a privileged user first creates them an account and grants that account the MANAGE flag:

```
popstore> ADD OPER/PASSWORD=secret/OWNER="popstore operator"/FLAG=MANAGE
```

In the above command, a "privileged" popstore account named oper with password secret is created. The account's privileges can then be used to manage the popstore via the LOGIN command:

```
popstore> LOGIN OPER
Password: secret
Login succeeded; management capabilities enabled
popstore>
```

Once logged in, the utility will allow the user to perform management functions on any account within the same management group as the user. If the user's account is in no management group — that is has a zero length group name — then the account can manage all accounts in the popstore. If the user's account is in no management group *and* is in the default user domain, then the user can manage *all* accounts in all groups and all user domains.

7.15 Information Display Formats

Formatting files control the choice and format of information presented with the SHOW and GROUP commands. These files are located in the pmdf_root:[www.popstore] directory. To display different choices of information or to change the formatting of the information, do not edit the Process Software supplied formatting files — your changes will be lost when you next upgrade PMDF. Instead, create new formatting files and use them instead. Once you have created a new formatting file, you can configure the utility to always use it, as described in Section 7.15.1. Or, you can use it occasionally to generate a monthly report. Such occasional usage is effected with the /FORMAT switch as described in Section 7.15.2.

7.15.1 Changing Default Display Formats

You can change this utility's default display formats through the use of an option file. The file is a PMDF-style option file named PMDF_TABLE:popstore_formats.;. Each option setting in the file takes the form

```
option-name=option-value
```

where option-name is the name of an option to set and option-value is the value to set for that option. The recognized option names and their default values are shown below

Option name	Default value	Used with
FORWARD_FORMAT	popmgr_forward.txt	SHOW/FORWARDINGS
GROUP_FORMAT	popmgr_groups.txt	GROUP/LIST
MSG_FORMAT	popmgr_message.txt	SHOW/MESSAGES
MSG_BRIEF_FORMAT	popmgr_message_brief.txt	SHOW/MESSAGES/BRIEF
PROFILE_FORMAT	popmgr_profile.txt	SHOW
PROFILE_BRIEF_FORMAT	popmgr_profile_brief.txt	SHOW/BRIEF

As an example, suppose you want to change the show command's output. You could then copy the popmgr_profile.txt file to, say, site_profile.txt and then edit the new file.[4] Then, create the PMDF_TABLE:popstore_formats.; file and in it place the line

```
PROFILE_FORMAT=site_profile.txt
```

Make sure that these file are world readable and owned by the PMDF account. Once you have done this, the SHOW command output will by default use your new formatting file. Note that if you make this change while running the utility, you will need to exit it and restart it in order for the change to be seen.

7.15.2 Report Generation

Customized reports can be generated using the /FORMAT_FILE qualifier of the SHOW command. With that qualifier, a formatting file can be specified. The file will then be used to format the information to be displayed. The syntax of formatting files is described in Section 4.3.4. For example, suppose that the file PMDF_ROOT:[WWW.POPSTORE]usage.txt contains the lines

```
%first{                                             Quota}
%first{                                   Owner      Used}
%first{---------------------------------------------------}
%owner{%40s}  %quota_used_k{%8.2f}
%last{---------------------------------------------------}
```

That file could then be used as follows:

[4] Recall that these files are stored in the pmdf_root:[www.popstore] directory.

```
$ PMDF POPSTORE SHOW/FORMAT_FILE=USAGE.TXT
                                                Quota
                                     Owner       Used
--------------------------------------------------
                               Russ Barnes      24.02
                              Orla Sheehan       8.56
                              Deanne Fagan      36.72
                      Default user profile       0.00
                               Marty Rolfe     133.98
                               Karen Russo      73.22
--------------------------------------------------
$
```

Note that for security reasons, the formatting files must be kept in the directory PMDF_ROOT:[WWW.POPSTORE]. This is enforced so as to prevent users with popstore management privileges from using the /FORMAT_FILE qualifier as a means of displaying protected files from other directories.

See Chapter 9 for further discussion of generating reports.

7.16 Recreating the Default Account

Should you accidentally delete the default account, you can recreate it using the x-add-default command:

```
popstore> x-add-default
```

You can then set settings for the default account using the modify command:

```
popstore> MODIFY DEFAULT/QUOTA=1000/OVERDRAFT=15
```

7.17 Recreating the User Database

Should the user database become corrupted or be accidentally deleted, you can recreate it using the x-build-user-db command:

```
popstore> x-build-user-db
```

This utility will create a new user database and populate it with entries found by scanning the profile directory tree.

7.18 Command Descriptions

The remainder of this chapter describes the individual utility commands.

ADD—Add a new account

Add a new user account to the popstore or MessageStore.

SYNTAX

ADD *username[,...]*
ADD/DOMAIN *domain-name*

Command Qualifiers
/CONFIRM
/DOMAIN
/FLAGS=flags
/LOG
/OVERDRAFT=value
/OWNER=owner
/PASSWORD=password
/PRIVATE=data
/PROMPT
/PWDEXPIRED
/QUOTA=value

PARAMETERS

username
Username to associate with the account or accounts being created.

DESCRIPTION

The `ADD` command is used to create one or more new popstore user accounts. An account will be created for each username supplied on the command line. Initial settings for the accounts are taken from the `default` account. Those settings can then be overridden with the command line qualifiers described below.

If a supplied username conflicts with an existing account, no new account is created and an error message is issued. Note that account usernames are case insensitive. That is the usernames `JDOE`, `JDoe`, and `jdoe` are all identical.

To create a new user domain, specify the `/DOMAIN` qualifier. If the domain already exists, an error will be issued. Otherwise, it will be created and a `default` user account for that domain created. The new `default` account will be a copy of the `default` account from the `default` domain. To begin creating accounts in the new domain, use the `SET DOMAIN` command. The maximum length of a user domain name is 40 bytes.

Note: Your PMDF-POPSTORE license controls the number of popstore user accounts which you can have at any one time. When you reach this limit, you will not be allowed to create additional accounts without first deleting some accounts or obtaining a new license with an increased limit. Sites without a PMDF-POPSTORE license are allowed to use the popstore and create up to ten user accounts. This limit does not include the `default` account. Use the `SHOW/COUNT_USERS` command to display the number of currently defined accounts as well as the limit allowed by your license.

COMMAND QUALIFIERS

/CONFIRM
/NOCONFIRM (default)

Prompt for positive confirmation before carrying out the indicated operation. `/NOCONFIRM` is the default behavior.

/DOMAIN

Create a new user domain. This switch can not be used in conjunction with any of the other `ADD` command qualifiers.

/FLAGS=(flag[,...])

Specify one or more usage flags to associate with the new account. The recognized flags are as follows:

DISMAIL	User is not allowed to receive new mail messages.
DISUSER	User is not allowed to access their account.
LOCKPWD	User is not allowed to change their password.
MANAGE	User is allowed to manage popstore accounts.
MIGRATED	Internal flag used by the PMDF migration utilities.
PWD_ELSEWHERE	Password information is stored outside of the popstore.
NODISMAIL	User is allowed to receive new mail messages.
NODISUSER	User is allowed to access their account.
NOLOCKPWD	User is allowed to change their password.
NOMANAGE	User is not allowed to manage popstore accounts.
NOMIGRATED	Internal flag used by the PMDF migration utilities.
NOPWD_ ELSEWHERE	Password information is stored within the popstore.

/LOG
/NOLOG (default)

When the operation is successful, output a status message stating that the operation succeeded. Note that error messages are always indicated. `/NOLOG` is the default behavior.

/OVERDRAFT=value
/NOOVERDRAFT

The `/OVERDRAFT` qualifier specifies the amount of message storage by which the account can exceed its message storage quota. If the account is currently using less than its storage quota, then a new message can be stored provided that it will not result in the account’s storage exceeding the sum of its storage and overdraft quotas.

The `/NOOVERDRAFT` qualifier is equivalent to specifying `/OVERDRAFT=0` and indicates that the account has no overdraft quota.

By default, this quantity is specified in units of kbytes; however, the SET STORAGE_UNITS command can be used to change the units used.

The maximum value is 4 gigabytes minus 1. If the value specified exceeds the maximum, the value is set to zero (no overdraft quota).

/OWNER=owner

A text string specifying the name of the owner of the account. The length of the string can not exceed 40 bytes. The owner field is not used by the popstore itself; it is generally used by humans to associate account usernames with the actual owner of the account.

/PASSWORD=password
/NOPASSWORD

Specifies the account's access password. The length of the password can not exceed 32 bytes. Access by non-managers to the account requires knowledge of this password. For instance, to access the account from a POP3 client, the correct username and password associated with the account must be supplied.

The /NOPASSWORD qualifier specifies that the account does not require a password to access it.

Note that passwords are case sensitive. Note further that the command line reader will convert to lower case any string not enclosed in quotes. As such, a password containing upper case characters must be enclosed in quotes.

/PRIVATE=data

Site-specific account data can be stored in the account profile file using this qualifier. The data string can not exceed a length of 64 bytes. This data is not used by the popstore itself but can be used by site-developed procedures which access account profiles.

/PROMPT (default)
/NOPROMPT

By default if a wildcard is used, even if /NOCONFIRM is specified, one confirmation prompt is issued. If /NOPROMPT is specified, there is no prompting at all.

/PWDEXPIRED
/NOPWDEXPIRED (default)

If /PWDEXPIRED is specified, then the account is marked as pre-expired. This means that if password expiration is enabled through the PASSWORD_LIFETIME option, then the user must change their password immediately.

If /NOPWDEXPIRED is specified (the default), then the account is marked as not pre-expired. The time of last password change is set to the current time. If password expiration is enabled, then the user does not have to change the password until the PASSWORD_LIFETIME has run out.

/QUOTA=value
/NOQUOTA

The /QUOTA qualifier specifies the account's message storage quota. The account can continue to receive new messages so long as the storage consumed by its currently stored messages does not exceed its message storage quota. See also /OVERDRAFT.

A quota value of zero, as specified with /NOQUOTA or /QUOTA=0, conveys unlimited storage. That is, to grant an account unlimited storage set its quota to zero.

By default, this quantity is specified in units of kbytes; however, the SET STORAGE_UNITS command can be used to change the units used.

The maximum value is 4 gigabytes minus 1. If the value specified exceeds the maximum, the value is set to zero (unlimited quota).

EXAMPLES

To create the account jdoe for Jane Doe with the password SeCrEt, and a quota of 10 Mbytes (10240 Kbytes), use the command

```
popstore> ADD JDOE/PASSWORD="SeCrEt"/OWNER="Jane Doe"/QUOTA=10240
popstore> SHOW JDOE
Username:           jdoe
Owner:              Jane Doe
Group:
Store Type:     popstore
Usage flags:
Site-defined:

Last pwd change:    Fri 15 Nov 15:33:02 2012
Last connect:       No time recorded
Last disconnect:    No time recorded
Total connect time: 0 00:00:00
Total connections:  0
Past block days:    0
Last billing:       Fri Nov 15  5 15:33:02 2012

Message count:              0 (0 total messages received)
Quota used:              0.00 Kbytes
Quota:               10240.00 Kbytes
Overdraft:              20.00 Kbytes
```

COPY—Duplicate an account

Create a new account which duplicates an existing account (popstore only).

SYNTAX

COPY *from-username to-username[,...]*

Command Qualifiers
/CONFIRM
/FLAGS=flags
/GROUP_NAME=name
/LOG
/OVERDRAFT=value
/OWNER=owner
/PASSWORD=password
/PRIVATE=data
/PROMPT
/PWDEXPIRED
/QUOTA=value

PARAMETERS

from-username
The name of the popstore account to copy.

to-username
The name of the popstore account or accounts to create.

DESCRIPTION

Use the `COPY` command to create a new popstore account which duplicates an existing popstore account. Note that the new account will have its usage accounting fields set to zero (*e.g.*, last connect, total connect, past block days, *etc*.). Also, the new account will not have any stored messages, even if the account being duplicated has stored messages.

If the username of the new account conflicts with an existing account, no new account is created and an error message is issued.

See also the `RENAME` command.

Note: Your PMDF-POPSTORE license controls the number of popstore user accounts which you can have at any one time. When you reach this limit, you will not be allowed to create additional accounts without first deleting some accounts or obtaining a new license with an increased limit. Sites without a PMDF-POPSTORE license are allowed to use the popstore and create up to ten user accounts. This limit does not include the `default` account. Use the `SHOW/COUNT_USERS` command to display the number of currently defined accounts as well as the limit allowed by your license.

COMMAND QUALIFIERS

/CONFIRM
/NOCONFIRM (default)

Prompt for positive confirmation before carrying out the indicated operation. /NOCONFIRM is the default behavior.

/FLAGS=(flag[,...])

Specify one or more usage flags to associate with the new account. The recognized flags are as follows:

DISMAIL	User is not allowed to receive new mail messages.
DISUSER	User is not allowed to access their account.
LOCKPWD	User is not allowed to change their password.
MANAGE	User is allowed to manage popstore accounts.
MIGRATED	Internal flag used by the PMDF migration utilities.
PWD_ELSEWHERE	Password information is stored outside of the popstore.
NODISMAIL	User is allowed to receive new mail messages.
NODISUSER	User is allowed to access their account.
NOLOCKPWD	User is allowed to change their password.
NOMANAGE	User is not allowed to manage popstore accounts.
NOMIGRATED	Internal flag used by the PMDF migration utilities.
NOPWD_ ELSEWHERE	Password information is stored within the popstore.

/GROUP_NAME=name

Place the new account into the specified management group. If not specified, the the management group of the account being copied is assumed. A manager can not create an account into a group which they cannot manage.

/LOG
/NOLOG (default)

When the operation is successful, output a status message stating that the operation succeeded. Note that error messages are always indicated. /NOLOG is the default behavior.

/OVERDRAFT=value
/NOOVERDRAFT

The /OVERDRAFT qualifier specifies the amount of message storage by which the new account can exceed its message storage quota. If the account is currently using less than its storage quota, then a new message can be stored provided that it will not result in the account's storage exceeding the sum of the its storage and overdraft quotas.

By default, this quantity is specified in units of kbytes; however, the SET STORAGE_UNITS command can be used to change the units used.

The /NOOVERDRAFT qualifier is equivalent to /OVERDRAFT=0 and specifies that the new account has no overdraft quota.

The maximum value is 4 gigabytes minus 1. If the value specified exceeds the maximum, the value is set to zero (no overdraft quota).

/OWNER=owner

A text string specifying the name of the owner of the new account. The length of the string can not exceed 40 bytes. The owner field is not used by the popstore itself; it is generally used by humans to associate account usernames with the actual owner of the account.

/PASSWORD=password
/NOPASSWORD

Specifies the new account's access password. The length of the password can not exceed 32 bytes. Access by non-managers to the account requires knowledge of this password. For instance, to access the account from a POP3 client, the correct username and password associated with the account must be supplied.

The `/NOPASSWORD` qualifier specifies that the new account does not require a password to access it.

Note that passwords are case sensitive. Note further that the command line reader will convert to lower case any string not enclosed in quotes. As such, a password containing upper case characters must be enclosed in quotes.

/PRIVATE=data

Site-specific account data for the new account can be stored in the account profile file using this qualifier. The data string can not exceed a length of 64 bytes. This data is not used by the popstore itself but can be used by site-developed procedures which access account profiles.

/PROMPT (default)
/NOPROMPT

By default if a wildcard is used, even if `/NOCONFIRM` is specified, one confirmation prompt is issued. If `/NOPROMPT` is specified, there is no prompting at all.

/PWDEXPIRED
/NOPWDEXPIRED (default)

If `/PWDEXPIRED` is specified, then the account is marked as pre-expired. This means that if password expiration is enabled through the `PASSWORD_LIFETIME` option, then the user must change their password immediately.

If `/NOPWDEXPIRED` is specified (the default), then the account is marked as not pre-expired. The time of last password change is set to the current time. If password expiration is enabled, then the user does not have to change the password until the `PASSWORD_LIFETIME` has run out.

/QUOTA=value
/NOQUOTA

The `/QUOTA` qualifier specifies the new account's message storage quota. The account can continue to receive new messages so long as the storage consumed by its currently stored messages does not exceed its message storage quota. See also `/OVERDRAFT`.

A quota value of zero, as specified with either `/QUOTA=0` or `/NOQUOTA`, conveys unlimited storage. That is, to grant an account unlimited storage set its quota to zero.

By default, this quantity is specified in units of kbytes; however, the `SET STORAGE_UNITS` command can be used to change the units used.

The maximum value is 4 gigabytes minus 1. If the value specified exceeds the maximum, the value is set to zero (unlimited quota).

EXAMPLES

To create a new account `jdoe` for Jane Doe which duplicates the account `bsmith` but has different owner and password fields, use the command

```
popstore> COPY BSMITH JDOE/PASSWORD="SeCrEt"/OWNER="Jane Doe"
```

DELETE—Remove a user account or a user's messages

Remove user accounts from the popstore or delete users' messages.

SYNTAX

DELETE *username[,...]*

Command Qualifiers

/CONFIRM	
/GROUP=name	
/LOG	
/MESSAGES	
/PROMPT	
/RETURN	*/NORETURN*

PARAMETERS

username
Name of the account to delete. Can contain wild card characters.

DESCRIPTION

Use the `DELETE` command to remove one or more user accounts. By default, stored messages for the accounts are deleted silently. To cause unread messages to be returned to their originators as undelivered, specify `/RETURN`.

Use the `/MESSAGES` qualifier to delete or return a user's messages. The account itself will not be deleted.

When the `username` parameter contains wild card characters, all matching accounts within the manager's management group and subgroups thereof will be deleted. The `/GROUP` qualifier can be used to further constrain which accounts are deleted.

COMMAND QUALIFIERS

/CONFIRM
/NOCONFIRM
Prompt for positive confirmation before carrying out the indicated operation. When wild cards are not used, `/NOCONFIRM` is the default. When wild cards are used, `/CONFIRM` is the default and a prompt is issued for each account to be operated upon. Moreover, when wild cards are used, `/NOCONFIRM` causes only a single prompt to be issued—it does not eliminate the prompt altogether.

/GROUP=name

Name of a management group to constrain the operation to. This qualifier can be used in conjunction with a username parameter containing wild card characters so as to further constrain the delete operation.

/LOG
/NOLOG

When the operation is successful, output a status message stating that the operation succeeded. Note that error messages are always indicated. `/NOLOG` is the default behavior unless wild cards are used in which case `/LOG` is the default.

/MESSAGES

When the `/MESSAGES` qualifier is specified, only the user's messages are deleted or returned. The account itself is not deleted.

/PROMPT (default)
/NOPROMPT

By default if a wildcard is used, even if `/NOCONFIRM` is specified, one confirmation prompt is issued. If `/NOPROMPT` is specified, there is no prompting at all.

/RETURN
/NORETURN (default)

When `/RETURN` is specified, unread messages are returned to their originator as undelivered. By default unread messages are deleted without sending a non-delivery notice back to their originators.

EXAMPLES

To delete the accounts `jdoe` and `bsmith`, issue the command

```
popstore> DELETE JDOE,BSMITH
```

EXIT—Exit the utility

Exit the utility.

SYNTAX

EXIT

Command Qualifiers
None

PARAMETERS *None.*

DESCRIPTION The EXIT command exits the utility.

FORWARD—Establish a forwarding address

Establish a forwarding address.

SYNTAX

FORWARD *username forward-to-address*

Command Qualifiers	
/OVERRIDE	*/OVERRIDE*

PARAMETERS

username
Username for which to establish a forwarding address.

forward-to-address
Address to which to forward messages. Must be a single, fully-qualified RFC822 address—specifically, a RFC822 “addr-spec”.

DESCRIPTION

Messages destined for the popstore can be automatically forwarded to a different popstore addressee or to another address outside of the popstore altogether. This is done by establishing a forwarding address with the `FORWARD` command. For instance, to forward all mail for the popstore user `sandy` to the address sandra@example.com, issue the command

```
popstore> FORWARD SANDY SANDRA@EXAMPLE.COM
```

The username supplied (*e.g.*, `sandy`) need not correspond to an actual popstore account.

Note that if more than one forwarding address is supplied, then each address should be separated by commas and all the addresses enclosed in a set of double quotes. For example,

```
popstore> FORWARD SANDY "SANDY,SANDRA@EXAMPLE.COM"
```

When a forwarding address is established for an actual popstore user, that user will no longer receive mail in the popstore unless the forwarding includes their account in the list of addresses to forward to. For instance, the first example above would cause the account `sandy` to no longer receive into the popstore mail sent to it. The mail is instead directed to sandra@example.com. In the second example, however, mail will still be stored into the popstore for the account `sandy`. In addition, a copy of the mail will be forwarded to sandra@example.com.

COMMAND QUALIFIERS

/OVERRIDE (default)
/NOOVERRIDE

By default, forwarding addresses can be established for existing popstore users. Specify `/NOOVERRIDE` to prevent inadvertently forwarding an existing user's messages elsewhere.

A manager cannot establish a forwarding address which will override a popstore account outside of their own management group.

GROUP—Manipulate management groups

Manipulate management groups.

SYNTAX

GROUP/ADD *[group-name [subgroup-name[,...]]]*
GROUP/DELETE *group-name*
GROUP/LIST *[group-name]*
GROUP/MODIFY *group-name [subgroup-name[,...]]*

Command Qualifiers
/ADD
/CONFIRM
/DELETE
/FORMAT_FILE=file-spec
/LIST
/LOG
/MODIFY
/OUTPUT=file-spec
/PROMPT
/RECUR

PARAMETERS

group-name
Name of the group to add, delete, list, or modify. Wild cards can be used in conjunction with the `/LIST` qualifier.

subgroup-name[,...]
A comma separated list group names to associated with the group being added or modified. The listed groups will become subgroups of the group being added or modified.

DESCRIPTION

The `GROUP` command is used to manipulate the popstore management groups. Only managers with either operating system privileges or a privileged popstore account with access to the `world` group can use this command. In regards to the latter case, that means that the manager's account must have the `MANAGE` usage flag set and either have no group name associated with the account—the empty group—or be in a management group which contains as a subgroup the `world` group. The one exception to this rule is that a manager can always use the `/LIST` qualifier to list their own management group and subgroups thereof.

For further details on the usage of this command as well as usage examples, see Section 7.12.

COMMAND QUALIFIERS

/ADD

This qualifier indicates that a new management group is to be added. If a group already exists with the same name, then an error will be output.

/CONFIRM
/NOCONFIRM (default)

Prompt for positive confirmation before carrying out the indicated operation. /NOCONFIRM is the default behavior.

/DELETE

This qualifier indicates that the specified management group is to be deleted. Note that subgroups contained within the group are not deleted by default. Specify /RECUR to also delete any subgroups. Moreover, the actual accounts in the group are not deleted either. They can only be deleted with a DELETE/GROUP=group_name * command.

/FORMAT_FILE=file-spec

Specify a formatting file to use to format the output of GROUP/LIST command.

/LIST

List the specified groups and subgroups. When this qualifier is used, the group-name parameter can contain wild card characters. When the parameter is omitted, * is assumed.

/LOG
/NOLOG (default)

When the operation is successful, output a status message stating that the operation succeeded. Note that error messages are always indicated. /NOLOG is the default behavior.

/MODIFY

Modify the specified group, replacing its list of subgroups with the specified list. If no list is specified, then the group is changed to contain no subgroups.

/OUTPUT=file-spec

Write the output to the specified file rather than to the terminal.

/PROMPT (default)
/NOPROMPT

By default if a wildcard is used, even if /NOCONFIRM is specified, one confirmation prompt is issued. If /NOPROMPT is specified, there is no prompting at all. This qualifier can be used in conjunction with the /ADD, /DELETE, or /MODIFY switches.

/RECUR
/NORECUR (default)

This qualifier can be used in conjunction with /DELETE. By default, only the specified group is deleted. Subgroups of that group are not deleted unless /RECUR is also specified.

LOGIN—Activate management privileges

Activate management privileges.

SYNTAX

LOGIN *[username]*

Command Qualifiers

None

PARAMETERS

username

Name of the account under which to log in.

DESCRIPTION

Popstore users who have popstore management privileges but lack operating system privileges must log in to their account with the LOGIN command in order to perform management operations. Once logged in, the utility will then allow the user to perform management operations. Users who have operating system privileges (*e.g.*, SYSPRV and SYSLCK privileges on OpenVMS), need not log in to their account.

To log in to a popstore account, the popstore account's username should be supplied using the username parameter to the LOGIN command. If the username parameter is omitted, the utility will use the name of the operating system account under which the user is logged in. The utility will then prompt for a password. If the correct password for the popstore account is supplied, and the account has the manage flag set, then the utility will allow management operations to be undertaken using the utility's image privileges.

A popstore account is granted management privileges by specifying

```
/FLAGS=MANAGE
```

when creating or modifying it with the ADD or MODIFY commands.

See also the LOGOUT command.

EXAMPLES

To log in to the account bob, issue the command

```
popstore> LOGIN BOB
Password: santaclaus
Login succeeded; management capabilities enabled
```

LOGOUT—Deactivate management privileges

Deactivate management privileges.

SYNTAX

LOGOUT

Command Qualifiers
None

PARAMETERS

None.

DESCRIPTION

Use the LOGOUT command to deactivate privileges activated with the LOGIN command. Note that management privileges are also deactivated when the utility is exited.

MODIFY—Change an existing account

Change characteristics of one or more existing accounts.

SYNTAX

MODIFY *username[,...]*

Command Qualifiers

/CONFIRM
/FLAGS=flags
/GROUP=name
/GROUP_NAME=name
/LAST_CONNECT
/LAST_DISCONNECT
/LOG
/MESSAGE_COUNT=value
/OVERDRAFT=value
/OWNER=owner
/PASSWORD=password
/PAST_BLOCK_DAYS=value
/PRIVATE=data
/PROMPT
/PWDEXPIRED
/QUOTA=value
/RECEIVED_BYTES=value
/RECEIVED_MESSAGES=value
/TOTAL_CONNECT=value
/TOTAL_CONNECTIONS=value

PARAMETERS

username
Name of the account for which to make the modifications. Can contain wild card characters.

DESCRIPTION

The `MODIFY` command changes one or more characteristics of an existing account. Characteristics not specified with qualifiers in the command are left unchanged.

When the username parameter contains wild card characters, all matching accounts within the manager's management group and subgroups thereof will be modified. The `/GROUP` qualifier can be used to further constrain which accounts are modified.

COMMAND QUALIFIERS

/CONFIRM
/NOCONFIRM

Prompt for positive confirmation before carrying out the indicated operation. When wild cards are not used, /NOCONFIRM is the default. When wild cards are used, /CONFIRM is the default and a prompt is issued for each account to be operated upon. Moreover, when wild cards are used, /NOCONFIRM causes only a single prompt to be issued—it does not eliminate the prompt altogether.

/FLAGS=(flag[,...])

Change the usage flags associated with the account. The recognized flags are as follows:

DISMAIL	User is not allowed to receive new mail messages.
DISUSER	User is not allowed to access their account.
LOCKPWD	User is not allowed to change their password.
MANAGE	User is allowed to manage popstore accounts.
MIGRATED	Internal flag used by the PMDF migration utilities.
PWD_ELSEWHERE	Password information is stored outside of the popstore.
NODISMAIL	User is allowed to receive new mail messages.
NODISUSER	User is allowed to access their account.
NOLOCKPWD	User is allowed to change their password.
NOMANAGE	User is not allowed to manage popstore accounts.
NOMIGRATED	Internal flag used by the PMDF migration utilities.
NOPWD_ ELSEWHERE	Password information is stored within the popstore.

/GROUP=name

Name of a management group to constrain the operation to. This qualifier can be used in conjunction with a username parameter containing wild card characters so as to further constrain the modify operation.

/GROUP_NAME=name

Change the accounts to be in the specified management group. A manager can not change an account's management group to be a group outside of the manager's group.

/LAST_CONNECT

Clear the user's last connect time field.

/LAST_DISCONNECT

Clear the user's last disconnect time field.

/LOG
/NOLOG

When the operation is successful, output a status message stating that the operation succeeded. Note that error messages are always indicated. /NOLOG is the default behavior unless wild cards are used in which case /LOG is the default.

/MESSAGE_COUNT=value

Reduce the user's message count to the specified value, deleting stored messages if necessary. The act of deleting stored message will change the past block days field.

/OVERDRAFT=value
/NOOVERDRAFT

Change the account's overdraft quota which is the amount of message storage by which the account can exceed its primary message storage quota. By default, this quantity is specified in units of kbytes; however, the `SET STORAGE_UNITS` command can be used to change the units used.

The `/NOOVERDRAFT` qualifier is equivalent to specifying `/OVERDRAFT=0` and indicates that the account has no overdraft quota.

The maximum value is 4 gigabytes minus 1. If the value specified exceeds the maximum, the value is set to zero (no overdraft quota).

/OWNER=owner

Change the accounts ownership field. The length of the string can not exceed 40 bytes. The owner field is not used by the popstore itself; it is generally used by humans to associate account usernames with the actual owner of the account.

/PASSWORD=password
/NOPASSWORD

Change the account's password. The length of the password can not exceed 32 bytes. Access by non-managers to the account requires knowledge of this password. For instance, to access the account from a POP3 client, the correct username and password associated with the account must be supplied.

The `/NOPASSWORD` qualifier specifies that the account does not require a password to access it.

Note that passwords are case sensitive. Note further that the command line reader will convert to lower case any string not enclosed in quotes. As such, a password containing upper case characters must be enclosed in quotes.

/PAST_BLOCK_DAYS=value

Set the user's past block days field to the specified, integer value. Changing this value clears the past block days remainder field.

/PRIVATE=data

Change the site-specific account data stored in the account profile file. The data string can not exceed a length of 64 bytes. This data is not used by the popstore itself but can be used by site-developed procedures which access account profiles.

/PROMPT (default)
/NOPROMPT

By default if a wildcard is used, even if `/NOCONFIRM` is specified, one confirmation prompt is issued. If `/NOPROMPT` is specified, there is no prompting at all.

/PWDEXPIRED
/NOPWDEXPIRED

If `/PWDEXPIRED` is specified, then the account is marked as pre-expired. This means that if password expiration is enabled through the `PASSWORD_LIFETIME` option, then the user must change their password immediately.

If /NOPWDEXPIRED is specified, then the account is marked as not pre-expired. The time of last password change is set to the current time. If password expiration is enabled, then the user does not have to change the password until the PASSWORD_LIFETIME has run out.

The default is to not change the pre-expired status of the account.

/QUOTA=value
/NOQUOTA

Change the account's message storage quota. The account can continue to receive new messages so long as the storage consumed by its currently stored messages does not exceed its message storage quota. See also /OVERDRAFT.

A quota value of zero, as specified with /NOQUOTA or /QUOTA=0, conveys unlimited storage. That is, to grant an account unlimited storage set its quota to zero.

A quota value of zero, conveys unlimited storage. That is, to grant an account unlimited storage set its quota to zero.

By default, this quantity is specified in units of kbytes; however, the SET STORAGE_UNITS command can be used to change the units used.

The maximum value is 4 gigabytes minus 1. If the value specified exceeds the maximum, the value is set to zero (unlimited quota).

/RECEIVED_BYTES=value

Set the cumulative count of received message bytes to the specified, integer value. By default, this quantity is specified in units of kbytes; however, the set storage_units command can be used to change the units used. The maximum value is 4 gigabytes minus 1. If the value specified exceeds the maximum, the value is set to zero.

/RECEIVED_MESSAGES=value

Set the cumulative count of received messages to the specified, integer value.

/TOTAL_CONNECT=value

Set the user's total connect field to the specified, integer value.

/TOTAL_CONNECTIONS=value

Set the user's count of total connections to the specified, integer value.

EXAMPLES

In the following example, the quota and password fields are changed for the user jdoe:

```
popstore> MODIFY JDOE/PASSWORD="TodaY"/QUOTA=20000
```

NOFORWARD—Remove a forwarding address

Remove a forwarding address.

SYNTAX

NOFORWARD *username[,...]*

Command Qualifiers

None

PARAMETERS

username
Username for which to remove the forwarding.

DESCRIPTION

Forwarding addresses are removed with the `UNFORWARD` command. If the supplied username also matches an existing popstore account, then that account will resume receiving new mail messages.

QUIT—Exit the utility

Exit the utility.

SYNTAX

QUIT

Command Qualifiers
None

PARAMETERS

None.

DESCRIPTION

The `QUIT` command exits the utility. The `QUIT` command is a synonym for the `EXIT` command.

RENAME—Rename an account

Change the username associated with an account (popstore only).

SYNTAX

RENAME *old-username new-username*

Command Qualifiers

/CONFIRM
/LOG
/PROMPT

PARAMETERS

old-username
The old name of the account.

new-username
The new name for the account.

DESCRIPTION

The `RENAME` command changes the username associated with a popstore account. Once an account is renamed, it can no longer receive mail under the old name unless a forwarding from the old name to the new name is also established with the `FORWARD` command.

COMMAND QUALIFIERS

/CONFIRM
/NOCONFIRM (default)
Prompt for positive confirmation before carrying out the indicated operation. /NOCONFIRM is the default behavior.

/LOG
/NOLOG (default)
When the operation is successful, output a status message stating that the operation succeeded. Note that error messages are always indicated. `/NOLOG` is the default behavior.

/PROMPT (default)
/NOPROMPT
By default if a wildcard is used, even if `/NOCONFIRM` is specified, one confirmation prompt is issued. If `/NOPROMPT` is specified, there is no prompting at all.

EXAMPLES

To rename the popstore account `jdoe` to `janedoe`, issue the command:

```
popstore> RENAME JDOE JANEDOE
```

SET DOMAIN—Set user domain

Select the user domain to manage.

SYNTAX

SET DOMAIN *domain-name*

Command Qualifiers

None

PARAMETERS

domain-name
Name of the user domain to manage.

DESCRIPTION

By default, the `default` user domain is managed with this utility. To manage a different user domain, select that domain with the `SET DOMAIN` command.

For example, to manage the `example.com` domain, specify

```
popstore> SET
Using the "default" user domain
...
popstore> SET DOMAIN EXAMPLE.COM
popstore> SET
Using the "example.com" user domain
...
popstore>
```

SET STORAGE_UNITS—Set storage units

Set the units used to measure byte-counted values.

SYNTAX

SET STORAGE_UNITS *type*

Command Qualifiers
None

PARAMETERS

type
Type of units to use. Must be one of BYTES, KBYTES, MBYTES, or GBYTES.

DESCRIPTION

By default, units of KBYTES (1024 bytes) are used when specifying values for byte-count valued fields such as message quotas.

To select a unit of measure other than KBYTES, use the SET STORAGE_UNITS command. BYTES selects bytes, KBYTES selects 1024 bytes, MBYTES selects 1024 KBYTES, and GBYTES selects 1024 MBYTES.

After issuing a SET STORAGE_UNITS command, all byte-count valued numbers input on the command line will be interpreted as being measured in the newly selected units. Note that displayed values are displayed in the units called for by the formatting template used to generate the display.

For example, to use units of megabytes, specify

```
    popstore> SET STORAGE_UNITS MBYTES
```

SET TIME_UNITS—Set time units

Set the units used to measure time-valued fields.

SYNTAX

SET TIME_UNITS *type*

Command Qualifiers

None

PARAMETERS

type

Type of units to use. Must be one of SECONDS, MINUTES, HOURS, or DAYS.

DESCRIPTION

By default, time units of DAYS are used when specifying values for time-valued fields. Presently, the only time-valued field is the total connect time field which can be modified with the /TOTAL_CONNECT qualifier of the MODIFY command.

To select a unit of measure other than DAYS, use the SET TIME_UNITS command. After issuing a SET TIME_UNITS command, all time-valued numbers input on the command line will be interpreted as being measured in the newly selected units. Note that displayed values are displayed in the units called for by the formatting template used to generate the display. For example, to use units of HOURS, specify

```
popstore> SET TIME_UNITS HOURS
```

SHOW—Show information about user profiles

Display user accounts.

SYNTAX

SHOW *username[,...]*

Command Qualifiers
/ALL
/BRIEF
/COUNT_USERS
/DOMAINS
/FORMAT_FILE=file-spec
/FORWARDINGS
/GROUP=name
/MESSAGES
/OUTPUT=file-spec
/STORE=store-type

PARAMETERS

username
Names of the accounts for which to display information. Wild cards are permitted.

DESCRIPTION

The `SHOW` command shows settings for one or more user profiles, displays established forwarding addresses, and lists information about messages received by users. The username parameter can contain wild cards when displaying account information; it can not contain wild cards when listing forwardings.

Use the `SHOW/FORWARDINGS` and `SHOW/DOMAINS` commands to generate listings of, respectively, user e-mail forwardings and user domains.

Use the `SHOW/COUNT_USERS` command to list the number of currently defined accounts as well as any licensing limits.

COMMAND QUALIFIERS

/ALL
By default, only popstore accounts are displayed: MessageStore and native accounts are not displayed. Specify `/ALL` to list all accounts. Note that `/ALL` and `/STORE=ALL` are synonyms.

/BRIEF

Generate a brief profile or message listing. By default, the formatting file `popmgr_profile_brief.txt` is used to format the output for profile displays and `popmgr_messsage_brief.txt` for message displays. This qualifier has no effect when used in conjunction with the `/FORWARDINGS` qualifier.

/COUNT_USERS

Display the number of currently defined user accounts as well as the number allowed by your PMDF-POPSTORE license. Specify `/ALL` to see both the popstore and MessageStore counts.

/DOMAINS

Generate a list of defined user domains. By default, the formatting file `pop-mgr_domains.txt` is used to format the output.

/FORMAT_FILE=file-spec

Specify a formatting file to use to format the output.

/FORWARDINGS

Display information about established forwarding addresses. By default, the formatting file `popmgr_forward.txt` is used to format the output.

/FULL (default)

Generate verbose output. By default, the formatting file `popmgr_profile.txt` is used to format profile information; `popmgr_message.txt` for message listings; `popmgr_domains.txt` for domain listings; and, `popmgr_forward.txt` for forwarding addresses. Those formatting files are found with the other formatting files in the `PMDF_ROOT:[WWW.POPSTORE]` directory.

/GROUP=name

Confine the listing to the specified management group and its subgroups.

/MESSAGES

Display information on the users' messages. By default, the formatting file `popmgr_message.txt` is used to format the display.

/OUTPUT=file-spec

Write the output to the specified file rather than to the terminal.

/STORE=store-type

By default, only popstore accounts are displayed: MessageStore and native accounts are not displayed. Specify `/STORE=ALL` to list all accounts; use a store-type of `MSGSTORE` or `IMAP` to list only MessageStore accounts; use a store-type of `POPSTORE` or `POP` to list only popstore accounts; and, use a store-type of `NATIVE` to list only profiles marked as being native.

EXAMPLES

In the following example, full and brief listings are generated for the `default` popstore account:

```
popstore> SHOW DEFAULT

Username:         default
Owner:            Default user profile
Group:
Store Type:      popstore
Usage flags:
Site-defined:

Last pwd change:    No time recorded
Last connect:       No time recorded
Last disconnect:    No time recorded
Total connect time: 0 00:00:00
Total connections:  0
Past block days:    0
Last billing:       Fri Nov 15 10:23:54 2012

Message count:              0 (0 total messages received)
Quota used:              0.00 Kbytes
Quota:                1024.00 Kbytes
Overdraft:              51.00 Kbytes

popstore> SHOW/BRIEF DEFAULT
                                          Quota  Message  Quota used
 Username                               (kbytes)    Count    (kbytes)
 ----------------------------------------------------------------
 default                                 1024.00        0        0.00
 ----------------------------------------------------------------
*Note: privileged users are flagged with an asterisk
```

TEST—Test site-supplied subroutines

Test optional, site-supplied subroutines to verify that they load and function correctly.

SYNTAX

TEST/BLOCK_DAYS *image-spec starting-time ending-time size remainder*
TEST/CONNECT *image-spec starting-time ending-time*
TEST/MESSAGE_MAPPING *image-spec*
TEST/PATHS *path-file-spec*
TEST/PROFILE_MAPPING *image-spec*

Command Qualifiers
/BLOCK_DAYS
/CONNECT
/MESSAGE_MAPPING
/PATHS
/PROFILE_MAPPING

PARAMETERS

image-spec
Executive mode logical whose translation value is the file specification for the shareable image containing the subroutine to test.

starting-time
Starting time value to pass to the `compute_connect` or `compute_block_days` subroutine.

ending-time
Ending time value to pass to the `compute_connect` or `compute_block_days` subroutine.

size
Size value to pass to the `compute_block_days` subroutine.

remainder
Remainder value to pass to the `compute_block_days` subroutine.

path-file-spec
File specification for the file of directory paths to check.

DESCRIPTION

The TEST command provides a mechanism to test site-supplied subroutines intended for use with the popstore. The purpose and usage of those subroutines is described in Chapter 14. Note that the shareable image containing the subroutine to be tested must be installed as a known image with the DCL INSTALL CREATE command. Moreover, an executive mode logical must be used to reference the image. The name of that logical is specified with the **image-spec** parameter. And, any logical referenced by that logical must also be an executive mode logical. These requirements are OpenVMS requirements and are enforced by LIB$FIND_IMAGE_SYMBOL, the run-time library subroutine used by the popstore to dynamically load and link to the subroutine.

Note that if you use the TEST command and then subsequently change your subroutine, then you will need to exit the utility and restart it before you can retest your subroutine. This is because LIB$FIND_IMAGE_SYMBOL won't reload the subroutine a second time. Also, when rebuilding a shareable image, be sure to use the DCL INSTALL REPLACE command to install the new version of the image.

The TEST/MESSAGE_MAPPING and TEST/PROFILE_MAPPING commands test, respectively, map_message_filename and map_profile_filename subroutines. The command will load the subroutine from the specified image and then, for each stored message or profile file, run the filename through the subroutine. The input and output file names for each file will be displayed along with diagnostic information, should an error occur.

The TEST/CONNECT and TEST/BLOCK_DAYS commands test, respectively, the compute_connect and compute_block_days subroutines. With each command, you can specify the values of the input arguments to be passed to those subroutines. The results produced by the subroutine will then be displayed. Should an error occur, diagnostic information will be displayed.

Text files intended for use as PMDF_TABLE:popstore_message_paths. or PMDF_TABLE:popstore_profiles_paths. files can be tested with the TEST /PATHS command. The command will scan the directory trees listed in the specified file, displaying the files found in each directory tree.

COMMAND QUALIFIERS

/BLOCK_DAYS

Test the compute_block_days subroutine from the shareable image **image-spec**.

/CONNECT

Test the compute_connect subroutine from the shareable image **image-spec**.

/MESSAGE_MAPPING

Test the map_message_filename subroutine from the shareable image **image-spec**.

/PROFILE_MAPPING

Test the map_profile_filename subroutine from the shareable image **image-spec**.

/PATHS

List the files from the directory trees specified in the path file **path-file-spec**.

EXAMPLES

In the following example, the `map_profile_filename` subroutine of Example 14–5 is tested with the TEST/PROFILE_MAPPING command on an OpenVMS Alpha system.

```
$ DEFINE/SYSTEM/EXECUTIVE_MODE POP_MAP_PROFILES -
_$      DISK3:[IMAGES]MAP_PROFILES.EXE
$ CC MAP_PROFILES.C
$ LINK/SHAREABLE=POP_MAP_PROFILES MAP_PROFILES.OBJ,SYS$INPUT:/OPT
SYMBOL_VECTOR=(map_profile_filename=PROCEDURE)
CTRL/Z
$ INSTALL CREATE POP_MAP_PROFILES
$ PMDF POPSTORE
popstore> TEST/PROFILE_MAPPING POP_MAP_PROFILES
PMDF_POPSTORE_PROFILES:[C.R.W]CRW.;1 -> DISK0:[PROFILES.C.R.W]CRW.;
PMDF_POPSTORE_PROFILES:[D.A.D]DAVID.;1 -> DISK0:[PROFILES.D.A.D]DAVID.;
PMDF_POPSTORE_PROFILES:[D.A.N]DAN.;1 -> DISK0:[PROFILES.D.A.N]DAN.;
PMDF_POPSTORE_PROFILES:[D.E.T]DEFAULT.;1 -> DISK0:[PROFILES.D.E.T]DEFAULT.;
PMDF_POPSTORE_PROFILES:[K.E.N]KEVIN.;1 -> DISK0:[PROFILES.K.E.N]KEVIN.;
PMDF_POPSTORE_PROFILES:[K.R.N]KRISTIN.;1 -> DISK0:[PROFILES.K.R.N]KRISTIN.;
PMDF_POPSTORE_PROFILES:[P.E.E]PEKIE.;1 -> DISK1:[PROFILES.P.E.E]PEKIE.;
PMDF_POPSTORE_PROFILES:[T.E.T]TEST.;1 -> DISK1:[PROFILES.T.E.T]TEST.;
popstore>
```

8 Migration

This chapter discusses two different forms of migration. In Section 8.1, the subject of moving the popstore from one machine to another is discussed. In Section 8.2, migrating Berkeley and VMS MAIL mailboxes to the popstore is discussed.

8.1 Migrating the popstore to Another Platform

Should you want to migrate the popstore to another platform, then you can simply copy the profile and message directory trees to the other platform: the files are architecture and operating system independent. On UNIX and NT, these are the directory trees `/pmdf/user/` and `/pmdf/popstore/messages/`. On OpenVMS, these are the directory trees `PMDF_POPSTORE_PROFILES:[*...]` and `PMDF_POPSTORE_MESSAGES:[*...]`.

However, PMDF database files are not in general architecture or operating system independent. They can only be exchanged between OpenVMS systems (Alpha, VAX, or I64). So, when moving the popstore to a different operating system or between Solaris SPARC and Solaris x86 systems, you will need to regenerate the popstore's management databases as described below.

User database

Once the profile files have been moved, simply run the PMDF POPSTORE utility and issue the X-BUILD-USER-DB command. The utility will create a new user database and populate it with entries found by scanning the profile directory tree.

Group database

Before moving the popstore, on the old system run the PMDF POPSTORE utility and issue the command

```
popstore> group -list -format=dump_groups_2unix.txt -output=groups.com
```

On OpenVMS systems, instead issue the command

```
popstore> GROUP/LIST/FORMAT=DUMP_GROUPS_2UNIX.TXT/OUTPUT=GROUPS.COM
```

If moving to an OpenVMS system, use the name `dump_groups_2vms.txt` in place of `dump_groups_2unix.txt` in the above commands.

The resulting file, `groups.com`, can then be used with the PMDF POPSTORE utility on the new system to build a new group database. If the new system is a UNIX or NT platform, then on the new system issue the command

```
popstore> run groups.com
```

If the new system is an OpenVMS system, instead issue the command

```
popstore> <<GROUPS.COM
```

Forwarding database

Before moving the popstore, on the old system run the PMDF POPSTORE utility and issue the command

```
popstore> show -forwardings -format=dump_forwardings.txt -output=forwardings.com
```

On OpenVMS systems, instead issue the command

```
popstore> SHOW/FORWARDINGS/FORMAT=DUMP_FORWARDINGS.TXT/OUTPUT=FORWARDINGS.COM
```

The resulting file, `forwardings.com` can then be used with the PMDF POPSTORE utility on the new system to build a new forwarding database. If the new system is a UNIX or NT platform, then on the new system issue the command

```
popstore> run forwardings.com
```

If the new system is an OpenVMS system, issue the command

```
popstore> <<FORWARDINGS.COM
```

Once the three databases have been built, the migration should be completed. Of course, you still have to configure PMDF on the new system.

8.2 Migrating Mailboxes

The popstore provides a migration utility which can be used to migrate mail mailboxes for login accounts to either the PMDF MessageStore or the PMDF popstore:

- On UNIX platforms, the utility migrates BSD-style mail files to the popstore or MessageStore as well as migrating popstore accounts to the MessageStore.
- On OpenVMS platforms, the utility migrates VMS MAIL NEWMAIL folders to the popstore, migrates entire VMS MAIL `mail.mai` files to the MessageStore, and migrates popstore accounts to the Message Store.
- On NT platforms the utility can migrate popstore accounts to the MessageStore.

The utility, described below, can be used to simultaneously migrate one or many accounts. It can either create new accounts as it runs (*e.g.*, when migrating native mail boxes to the popstore or MessageStore), or use pre-created accounts (*e.g.*, when moving popstore accounts to the MessageStore).

8.2.1 Migrating UNIX and NT Mailboxes

On UNIX and NT platforms, the migration utility is the `pmdf movein` utility. Its usage is described below.

MOVEIN—Move a mailbox

Migrate mail files between message stores or between accounts.

SYNTAX

pmdf movein *[username [password]]*

Command Switches	Default
`-before=time`	
`-debug`	`-nodebug`
`-destination=store-type`	
`-dstdmn=user-domain`	
`-dstusername=dest-username`	*source username*
`-exception_file=file-spec`	
`-force_migrate`	`-noforce_migrate`
`-forward`	`-forward`
`-help`	
`-host=hostname`	
`-inbox=file-spec`	
`-input=file-spec`	
`-log`	`-log`
`-since=time`	
`-source=store-type`	`-source=native`
`-srcdmn=user-domain`	
`-use_existing`	`-nouse_existing`

restrictions

You must be user `root` to run this utility.

PARAMETERS

username
Optional name of a single source message store user account to migrate. If not supplied, then the names of accounts to migrate are read from either an input file or standard input, stdin.

password
Optional password to set for the new account in the destination message store. Only used when migrating to the popstore or MessageStore.

DESCRIPTION

The `pmdf movein` utility migrates mailboxes from one message store to another, or from one account to another. The supported message stores are: BSD-style mail files (the “native” message store), the PMDF MessageStore, and the PMDF popstore. When moving from or to the native or popstore message stores, only messages in the inbox are migrated. For migrations from one msgstore account to another, messages in all folders are moved.

The `pmdf movein` utility performs all of the necessary locking so as to allow migrations to be carried out on an active system without loss of mail. Forwardings from the source message store account to the destination message store account are automatically established for each migrated user. Note, however, that use of the `-noforwarding` switch can compromise this robustness.

The basic inputs to the utility are: the names of the source and destination message stores; the names of the source store user accounts to migrate; optionally, the names of the destination store user accounts to create; and, also optionally, passwords to set for these new accounts. The source store username is assumed for the destination store if a desintation username is not specified. That is, by default, the name of the account in the destination store is the same as that in the source store.

When a username is supplied on the command line, then just that one user account is migrated. To migrate multiple accounts, do not specify a username on the command line and instead use the `-input` switch. Each line of the input file specifies the username of a single source store account to migrate. In addition, a line can specify a password to associate with the new account in the destination store, and the username to create in the destination store. The password and destination username, if supplied, must be on the same input line following the username with one or more white space characters separating each of the three. If the password contains a space itself, it must be enclosed in double quotes. To specify a destination username without specifying a password, the password must be specfied as an empty string (two double-quotes next to each other).

Comment lines can appear in the input file: a comment line is any line which begins with a `#` or `!` character. Leading and trailing white space characters on lines are ignored. The following sample input should give the flavor of input files:

```
# A comment line
#
# to supply a password and destination username:
angel "alex's password" alex
bailey blakes-password blake
#
# to supply a destination username, but no password:
chris "" casey
#
# to supply a password but no destination username:
dana danas-password
#
# to supply neither a password nor a destination username:
emile
```

When no username or input file is specified, input lines are read from standard input, stdin. The format of those input lines is identical to that of lines read from an input file. Input is terminated by typing a `CTRL/D` (EOD).

When migrating to a msgstore or popstore account, the destination account is automatically created. If a password is supplied, then the password is set in the new account. If no password is supplied, then the new account is marked with the `PWD_ELSEWHERE` flag. Normally, that causes authentication to then be performed against `/etc/passwd`. See the popstore documentation for further information. When migrating from a BSD-style mail file, the owner field in the new account is set from the gcos field in the `/etc/passwd` file. If moving between a popstore and msgstore account, the owner field of the destination account is set from the source account.

If there already is a pre-existing msgstore or popstore account—such as when migrating from the popstore to the msgstore—then the migration fails unless `-use_existing` is specified. Moreover, if the existing msgstore or popstore account has the MIGRATED flag set, then the migration fails unless `-force_migrate` is specified. When the account is successfully migrated, the `MIGRATED` flag is set in the account.

When migrating to a BSD-style mail file, a login account must already exist for the user. The account will not automatically be created. Any supplied password is ignored.

When migrating either to or from BSD-style mail files, the PMDF local delivery channel's methodology for locating a user's BSD-style mailbox file is used: the user's `.forward` file is first consulted, the PMDF profile database is checked, and as a last resort the path specified with the `-inbox` switch is used. If that switch is not specified, then the platform-specific `/var/mail` location is used as a last resort.

Note: After a successful migration, the account in the source message store is left intact with its mail in place. In the case of a BSD-style mail file, the mail file is protected against further writing. In the case of migration from a popstore to MesageStore store type with the same username (or vice-versa), the account is changed in place from the one store type to the other.

The `pmdf movein` utility is extremely careful to back out of a failed migration, leaving the user account in the source message store in its original state. When multiple accounts are migrated, a failure to migrate an account does not terminate the utility. Instead, the utility backs out of the migration for the failed account and continues on to the next account. Failures are reported to standard error, stderr. The `-exception_file` switch can be used to log errors in an exception file. The format of the exception file is suitable for re-use as an input file.

COMMAND SWITCHES

`-before=hh:mm:ss:dd-mmm-yyyy` ***(UNIX only)***
By default, all messages are migrated. The `-before` switch can be used to only migrate those messages stored on or before the specified time. For instance, to migrate all messages received between 08:30 and 13:00 on 1 April 2012, specify

```
    -since=8:30:00:1-april-2012 -before=13:00:00:1-april-2012
```

The actual date and time time specification is parsed by the C run-time library function `strptime`. The formatting string used with `strptime` is

```
    %H:%M:%S:%d-%b-%Y
```

For further information on specifying date and time specifications, please consult your platform's `strptime` documentation.

-debug
-nodebug ***(default)***
Debug output can be enabled with the `-debug` switch.

-destination=store-type
This required switch specifies the name of the target message store. The accepted names are `native` (BSD-style mail files), `msgstore` (PMDF MessageStore), and `popstore` (PMDF popstore).

-dstdmn=user-domain
The name of the user domain to use in the destination message store. If not specified, then the `default` domain is used. Only applicable in conjunction with destination message stores which support user domains (currently, only popstore).

-dstusername=dest-username
The username to use in the destination message store. If not specified, the source store username is used.

-exception_file=file-spec
By default, failures are only written to standard error, stderr. Failures can also be reported to an exception file. The name of the exception file is given with the `-exception_file` switch. The file will only be created should a failure occur; the file will have the permissions 0600. For each account whose migration fails, an entry will be made to the exception file. The entry takes the form of one or more lines of comments followed by a line containing the failed username and any optional password and destination username. The format of the exception file is such that it can be re-used as an input file.

-force_migrate
-noforce_migrate ***(default)***
This switch can be used in conjunction with the `-use_existing` switch. See the description of that switch for further details.

-forward ***(default)***
-noforward
By default, a forwarding address for each migrated user account is established. This forwarding causes undelivered mail sent to the user's source message store address to be automatically forwarded to their account in the destination message store.

When migrating from the native message store, a forwarding to the account in the destination store is placed in the PMDF alias database. In addition, a `.forward` file with the same forwarding is created in the user's login directory. Any previously existing `.forward` file is renamed to `.forward.save`. If a file named `.forward.save` already exists, the previously existing `.forward` file is instead renamed to `.forward.save.`nnnnnn where nnnnnn is a six-digit number between `000000` and `999999`. The new `.forward` file will either have the same ownership and permissions as the previous file or, if there was no previous file, it will be owned by the user and have the permissions 0600.

When migrating to the native message store, any forwarding for the account in the PMDF alias database is removed, and any `.forward` file is displaced in accord with the rules cited in the prior paragraph.

When migrating from the popstore or msgstore, a forwarding to the account in the destination store is added to the msgstore's forwarding database. When migrating to the popstore or msgstore, any msgstore forwarding for the destination account is removed from the msgstore's forwarding database. If the account is just being converted from popstore to msgstore (or vice-versa) in place, no forwarding is necessary.

Specify `-noforward` to prevent forwardings from being established. Note, however, that when `-noforward` is used, undelivered mail directed to the user's old address in the source message store can be returned as undeliverable (native message store) or delivered to the old, unused account (popstore or msgstore) until such time that the old account is deleted at which point the mail will be returned as undeliverable.

-host=hostname
Optional switch to specify the host name associated with the destination message store. If omitted, the host name used will be determined from PMDF configuration information: the official local host name when the destination store is the native message store; the host name on the msgstore channel when the destination store is popstore or msgstore. This host name is only used in conjunction with the `-forward` switch to construct a forwarding address for each migrated account.

-inbox=file-spec
This switch can be used when the native message store is used as either a source or destination message store or both. In those cases, it provides a default file specification to use when attempting to locate a user's BSD-style inbox file. The utility will always first look for a `.forward` file. If that does not produce an inbox file location, then the PMDF profile database is consulted. If that does not provide an inbox file location, then the location specified with the `-inbox` switch is used. If that switch is not specified, then `/var/mail` (Solaris) or `/var/spool/mail` (Linux) is used.

The file specification supplied with the `-inbox` switch can include the following substitution strings:

`%+`, `%s`, `%u`	Substitute in the name of the user account being migrated
`%d`, `%h`	Substitute in the home directory of the user account being migrated
`\x`	Substitute in the literal character `x` (*e.g.,* `\%` substitutes in `%`).

Thus, for example, on Solaris platforms the default path is `-inbox=/var/mail/%s`.

-input=file-spec
Specify an input file containing names of user accounts to migrate. Can not be used in conjunction with the **username** command line parameter. See the description above for further information on input files.

`-log` ***(default)***
`-nolog`
By default, the utility reports each successfully migrated account to standard error, stderr. To suppress this reporting, specify `-nolog`. Note that errors are always reported.

`-since=hh:mm:ss:dd-mmm-yyyy` ***(UNIX only)***
By default, all messages are migrated. The `-since` switch can be used to only migrate those messages stored on or after the specified time. See the description of the `-before` switch for further details.

`-source=store-type`
By default, the source message store is assumed to be the native message store. The permitted values are `native`, `popstore`, and `msgstore`.

`-srcdmn=user-domain`
The name of the user domain to use in the source message store. If not specified, then the `default` domain is used. Only applicable in conjunction with source message stores which support user domains (currently, only popstore).

`-use_existing`
`-nouse_existing` ***(default)***
This switch is only honored when the destination message store is the popstore or msgstore. It is ignored when migrating to the native message store.

By default, a migration to a popstore or msgstore account will fail when there already is a pre-existing account with the same name as the requested destination account name (by default, the name of the source store account being migrated). To override this behavior, specify `-use_existing`. However, if the pre-existing account has the msgstore/popstore `MIGRATED` flag set, the migration will fail unless `-force_migrate` is also specified. Note that when migrating to a pre-existing account, the password for the account is left unchanged and migrated messages are added to those already stored for the account.

EXAMPLES

1
```
# pmdf movein -destination=popstore sue SeCreT
movein: Destination message store host name not supplied; using pop.com
movein: User "sue" successfully migrated (/var/mail/sue)
```

This example shows migrating to the popstore the login user `sue`. The password on the new popstore account is set to `SeCreT`.

2
```
# pmdf movein -source=popstore -destination=msgstore -use_existing sue
movein: Destination message store host name not supplied; using imap.com
movein: User "sue" successfully migrated
```

This example shows migrating the popstore user `sue` to the MessageStore. The `-use_existing` switch instructs the utility to use Sue's existing profile file for her MessageStore account. This way, her password and other usage information is preserved. Note that if `-use_existing` was not specified, this command would fail.

3

```
# cat user_list
smith SeCreT jones
williams "" johnson
# pmdf movein -destination=msgstore -source=msgstore -input=user_list
movein: Destination message store host name not supplied; using imap.com
movein: "smith" migrated
movein: "williams" migrated
```

In the above example an input file is used to direct the `pmdf movein` utility. The file instructs movein to copy the messages from one msgstore account to another. For example, in the first case, a `jones` msgstore account is created, its password is set to "SeCreT", and all of the messages in the msgstore `smith` account are copied over.

8.2.2 Migrating OpenVMS Mailboxes

On OpenVMS platforms, the migration utility is the PMDF MOVEIN utility. Its usage is described below.

MOVEIN—Move a mailbox

Migrate mail files between message stores or between accounts.

SYNTAX

PMDF MOVEIN *[username [password]]*

Command Qualifiers	**Default**
/BEFORE=time	
/DEBUG	*/NODEBUG*
/DESTINATION=store-type	
/DSTDMN=user-domain	
/DSTUSERNAME=dest-username	*source username*
/EXCEPTION_FILE=file-spec	
/FORCE_MIGRATE	*/NOFORCE_MIGRATE*
/FORWARD	*/FORWARD*
/HOST=hostname	
/INPUT=file-spec	
/LOG	*/LOG*
/LOWERCASE	*/NOLOWERCASE*
/SINCE=time	
/SOURCE=store-type	*/SOURCE=NATIVE*
/SRCDMN=user-domain	
/USE_EXISTING	*/NOUSE_EXISTING*
/WASTEBASKET	*/WASTEBASKET*

restrictions

Operating system privileges are required to run this utility.

PARAMETERS

username
Optional name of a single source message store user account to migrate. If not supplied, then the names of accounts to migrate are read from either an input file or `SYS$INPUT`.

password
Optional password to set for the new account in the destination message store. Only used when migrating to the popstore or MessageStore.

DESCRIPTION

The `PMDF MOVEIN` utility migrates mailboxes from one message store to another, or from one account to another. The supported message stores are: VMS MAIL (the “native” message store), the PMDF MessageStore, and the PMDF popstore. When moving from or to the popstore, only messages in the inbox are migrated.

For migrations from VMS MAIL to msgstore, or from one msgstore account to another, messages in all folders are moved.

Forwardings from the source message store account to the destination message store account are automatically established for each migrated user. Note, however, that use of the `-noforwarding` switch can compromise this robustness.

As it is not possible to temporarily block delivery of mail to a user by VMS MAIL, when migrating mail from a VMS MAIL account, the `PMDF MOVEIN` utility should not be run on an active system. Running it on an active system can lead to loss of mail when the migration utility migrates a user's `NEWMAIL` folder and, at the same time, VMS MAIL delivers a new message to that same folder. Depending upon the timing, that new message can be left behind and not migrated. To prevent this from happening, ensure that PMDF's local delivery channel is not running (*e.g.*, stop the `MAIL$BATCH` queue or whatever queue the l channel runs in) and that any other VMS MAIL applications are not running (*e.g.*, no login users using VMS MAIL, no Pathworks users, DECnet MAIL-11 shutdown, *etc.*).

When migrating mail from a popstore or msgstore account, and to a popstore or msgstore account, the `PMDF MOVEIN` utility performs all of the necessary locking so as to allow migrations to be carried out on an active system without loss of mail.

The basic inputs to the utility are: the names of the source and destination message store; the names of the source store user accounts to migrate; optionally, the names of the destination store user accounts to create; and, also optionally, passwords to set for these new accounts. The source store username is assumed for the destination store if a desintation username is not specified. That is, by default, the name of the account in the destination store is the same as that in the source store.

When a username is supplied on the command line, then just that one user account is migrated. To migrate multiple accounts, do not specify a username on the command line and instead use the `/INPUT` qualifier. Each line of the input file specifies the username of a single source store account to migrate. In addition, a line can specify a password to associate with the new account in the destination store, and the username to create in the destination store. The password and destination username, if supplied, must be on the same input line following the username with one or more white space characters separating each of the three. If the password contains a space itself, it must be enclosed in double quotes. To specify a destination username without specifying a password, the password must be specfied as an empty string (two double-quotes next to each other).

Comment lines can appear in the input file: a comment line is any line which begins with a `#` or `!` character. Leading and trailing white space characters on lines are ignored. The following sample input should give the flavor of input files:

```
! A comment line
!
! to supply a password and destination username:
angel "alex’s password" alex
bailey blakes-password blake
!
! to supply a destination username, but no password:
chris "" casey
!
! to supply a password but no destination username:
dana danas-password
!
! to supply neither a password nor a destination username:
emile
```

When no username or input file is specified, input lines are read from SYS$INPUT. The format of those input lines is identical to that of lines read from an input file. Input is terminated by typing a CTRL/Z.

When migrating to a msgstore or popstore account, the destination account is automatically created. If a password is supplied, then the password is set in the new account. If no password is supplied, then the new account is marked with the PWD_ELSEWHERE flag. Normally, that causes authentication to then be performed against the SYSUAF database. See the popstore documentation for further information. When migrating from VMS MAIL, the owner field in the new account is set using the owner field from the SYSUAF. When migrating between a popstore and msgstore account, the owner field of the destination account is set from the source account.

If there already is an pre-existing MessageStore or popstore account—such as when migrating from the popstore to MessageStore—then the migration fails unless /USE_EXISTING is specified. Moreover, if the existing account has the MIGRATED flag set, then the migration fails unless /FORCE_MIGRATE is specified. When the account is successfully migrated, the MIGRATED flag is set in the account.

Note: After a successful migration, the account in the source message store is left intact with its mail in place. In the case of migration from a popstore to MesageStore store type with the same username (or vice-versa), the account is changed in place from the one store type to the other.

The PMDF MOVEIN utility is extremely careful to back out of a failed migration, leaving the user account in the source message store in its original state. When multiple accounts are migrated, a failure to migrate an account does not terminate the utility. Instead, the utility backs out of the migration for the failed account and continues on to the next account. Failures are reported to SYS$ERROR. The /EXCEPTION_FILE qualifier can be used to log errors in an exception file. The format of the exception file is suitable for re-use as an input file.

COMMAND QUALIFIERS

/BEFORE=dd-mmm-yyyy:hh:mm:ss

By default, all messages are migrated. The /BEFORE qualifier can be used to only migrate those messages stored on or before the specified time. For instance, to migrate all messages received between 08:30 and 13:00 on 15 November 2012, specify

```
/SINCE=15-NOV-2012:8:30:00 /BEFORE=15-NOV-2012:13:00:00
```

/DEBUG
/NODEBUG (default)

Debug output can be enabled with the /DEBUG qualifier.

/DESTINATION=store-type

This required qualifier specifies the name of the target message store. The accepted names are MSGSTORE (PMDF MessageStore) and POPSTORE (PMDF popstore).

/DSTDMN=user-domain

The name of the user domain to use in the destination message store. If not specified, then the `default` domain is used. Only applicable in conjunction with destination message stores which support user domains (currently, only popstore).

/DSTUSERNAME=dest-username

The username to use in the destination message store. If not specified, the source store username is used.

/EXCEPTION_FILE=file-spec

By default, failures are only written to SYS$ERROR. Failures can also be reported to an exception file. The name of the exception file is given with the /EXCEPTION_FILE qualifier. The file will only be created should a failure occur; the file will have the protection mask (S:RWED,O:RWED,G,W). For each account whose migration fails, an entry will be made to the exception file. The entry takes the form of one or more lines of comments followed by a line containing the failed username and any optional password and destination username. The format of the exception file is such that it can be re-used as an input file.

/FORCE_MIGRATE
/NOFORCE_MIGRATE (default)

This qualifier can be used in conjunction with the /USE_EXISTING qualifier. See the description of that qualifier for further details.

/FORWARD (default)
/NOFORWARD

By default, a forwarding address for each migrated user account is established. This forwarding causes undelivered mail sent to the user's source message store address to be automatically forwarded to their account in the destination message store.

When migrating an account from VMS MAIL, a forwarding for the VMS MAIL user is is placed both in the PMDF alias database and VMS MAIL's forwarding database. When migrating from the popstore or msgstore, a forwarding to the account in the destination store is added to the msgstore's forwarding database.

When migrating to the popstore or msgstore, any msgstore forwarding for the destination account is removed from the msgstore's forwarding database. If the account is just being converted from popstore to msgstore (or vice-versa) in place, no forwarding is necessary.

Specify /NOFORWARD to prevent forwardings from being established. Note, however, that when /NOFORWARD is used, undelivered mail directed to the user's old address in the source message store can be returned as undeliverable or delivered to the old, unused account until such time that the old account is deleted at which point the mail will be returned as undeliverable.

/HOST=hostname

Optional qualifier to specify the host name associated with the destination message store. If omitted, the host name used will be determined from PMDF configuration information: the official local host name when the destination store is the native message store; the host name on the msgstore channel when the destination store is msgstore or popstore. This host name is only used in conjunction with the /FORWARD qualifier to construct a forwarding address for each migrated account.

/INPUT=file-spec

Specify an input file containing names of user accounts to migrate. Can not be used in conjunction with the **username** command line parameter. See the description above for further information on input files.

/LOG (default)
/NOLOG

By default, the utility reports each successfully migrated account to SYS$ERROR. To suppress this reporting, specify /NOLOG. Note that errors are always reported.

/LOWERCASE
/NOLOWERCASE (default)

This qualifier is only used when the source store is VMS MAIL and the destination store is MessageStore. By default, VMS MAIL folder names are not modified when they are migrated. This default behavior corresponds to /NOLOWERCASE. When /LOWERCASE is specified, all migrated folder names are converted to lower case.

/SINCE=dd-mmm-yyyy:hh:mm:ss

By default, all messages are migrated. The /SINCE qualifier can be used to only migrate those messages stored on or after the specified time. See the description of the /BEFORE qualifier for further details.

/SOURCE=store-type

By default, the source message store is assumed to be the native message store, VMS MAIL. The permitted values are NATIVE (VMS MAIL), POPSTORE (PMDF popstore), and MSGSTORE (PMDF MessageStore).

/SRCDMN=user-domain

The name of the user domain to use in the source message store. If not specified, then the default domain is used. Only applicable in conjunction with source message stores which support user domains (currently, only popstore).

/USE_EXISTING
/NOUSE_EXISTING (default)

By default, a migration to a popstore or msgstore account will fail when there already is a pre-existing account with the same name as the requested destination account name (by default, the name of the source store account being migrated). To override this behavior, specify `/USE_EXISTING`. However, if the pre-existing account has the msgstore/popstore `MIGRATED` flag set, the migration will fail unless `/FORCE_MIGRATE` is also specified. Note that when migrating to a pre-existing account, the password for the account is left unchanged and migrated messages are added to those already stored for the account.

/WASTEBASKET (default)
/NOWASTEBASKET

When migrating a VMS MAIL user to the MessageStore, the `WASTEBASKET` folder will, by default, also be migrated. Specify `/NOWASTEBASKET` to prevent the `WASTEBASKET` folder from being migrated.

EXAMPLES

1
```
$ PMDF MOVEIN /DESTINATION=POPSTORE SUE "SeCreT"
movein: Destination message store host name not supplied; using pop.com
movein: User "SUE" successfully migrated (D1:[USERS.SUE]MAIL.MAI)
```

This example shows migrating to the popstore the login user `SUE`. The password on the new popstore account is set to `SeCreT`.

2
```
$ PMDF MOVEIN /SOURCE=POPSTORE /DESTINATION=MSGSTORE -
_$ /USE_EXISTING SUE
movein: Destination message store host name not supplied; using imap.com
movein: User "SUE" successfully migrated
```

This example shows migrating the popstore user `sue` to the MessageStore. The `/USE_EXISTING` qualifier instructs the utility to use Sue's existing profile file for her MessageStore account. This way, her password and other usage information is preserved. Note that if `/USE_EXISTING` was not specified, this command would fail.

3
```
$ TYPE USERS.LIS
smith SeCreT jones
williams "" johnson
$ PMDF MOVEIN /DESTINATION=MSGSTORE /SOURCE=MSGSTORE -
_$/INPUT=USERS.LIS
movein: Destination message store host name not supplied; using imap.com
movein: "smith" migrated
movein: "williams" migrated
```

In the above example an input file is used to direct the `PMDF MOVEIN` utility. The file instructs MOVEIN to copy the messages from one msgstore account to another. For example, in the first case, a `jones` msgstore account is created, its password is set to "SeCreT", and all of the messages in the msgstore `smith` account are copied over.

9 Report Generation

The command line management utility described in Chapters 6 and 7 can be used to generate reports and account listings.

There are two steps to generating a report:

1. writing a formatting file which formats the desired data, and
2. invoking the command line management utility to process the formatting file.

Each of these steps are discussed in this chapter.

9.1 Writing a Formatting File

Three types of information can be reported:

1. user account information,
2. user account message lists, and
3. listings of forwarding addresses.

The information is formatted using formatting files as described in Section 4.3.2. The substitution strings allowed in the formatting files are listed in the tables cited below:

Information type	Substitution strings
Account information	Tables 4–10 and 4–15
Message lists	Tables 4–10 and 4–11
Forwarding addresses	Tables 4–10 and 4–24

The formatting files, once written, must be stored in the `/pmdf/www/popstore/` directory tree on UNIX and NT systems or the in the `PMDF_ROOT:[WWW.POPSTORE]` directory on OpenVMS systems.

9.2 Producing a Report

Once a formatting file has been written, it can be used in conjunction with the command line management utility with a command from the table below:

	Information type	Command
UNIX, NT	Account information	show -format_file=file-spec [username]
	Message lists	show -messages -format_file=file-spec username
	Forwarding addresses	show -forwardings -format_file=file-spec [username]
VMS	Account information	SHOW/FORMAT_FILE=file-spec [username]
	Message lists	SHOW/MESSAGES/FORMAT_FILE=file-spec username
	Forwarding addresses	SHOW/FORWARDINGS/FORMAT_FILE=file-spec [username]

where `file-spec` is the name of the formatting file to use and `username` is the name of the account to report on. The account name is optional when listing account or forwarding information.

For example, to list all user accounts using the formatting file `list.txt`, issue the command

```
# pmdf popstore show -format_file=list.txt
```

on UNIX and NT systems. On OpenVMS systems, issue the command

```
$ PMDF POPSTORE SHOW/FORMAT_FILE=LIST.TXT
```

9.3 An Example

The following example demonstrates how to generate a report of all popstore accounts. In the listing, the account name, message count, and total size of stored messages is shown for each account. The formatting file is shown in Example 9–1 and provided with PMDF under the name `report.txt`.

Example 9–1 Account Listing Report

```
%first{                                Message  Quota used}
%first{ Username                         Count    (kbytes)}
%first{ ----------------------------------------------------------}
%flags_manage{ |*}%username{%-32s} %message_count{%7u}    %quota_used_k{%8.2f}
%last{ ---------------------------------------------------------------}
%last{*Note: privileged users are flagged with an asterisk}
```

To generate the report on UNIX and NT systems, issue the command:

```
# pmdf popstore show -format_file=report.txt
```

and on OpenVMS systems, issue the command:

```
$ PMDF POPSTORE SHOW/FORMAT_FILE=REPORT.TXT
```

10 Inbound Message Delivery & Message Bouncer

The popstore and MessageStore share a common inbound delivery channel. The popstore also has a message bouncer process which returns or removes old, undeleted e-mail from the popstore. The function of these two agents are described in this chapter. See the *PMDF Installation Guide* for directions on configuring these agents.

10.1 The Inbound Delivery Channel

The inbound delivery channel runs as a normal PMDF delivery channel. Messages are queued to it by PMDF for delivery to the popstore or MessageStore. The delivery channel then processes each inbound message, either delivering it to the popstore or MessageStore, forwarding it elsewhere, or returning it to its originator as undeliverable.

In the popstore, the messages are stored in a ready-to-download format so that the POP3 server can simply map them into memory and send the bytes down to the client without the need for any further processing. Envelope information is also stored in the message file.

On UNIX systems, the message files are kept in the directory tree specified by the `PMDF_POPSTORE_MESSAGES` option in the PMDF tailor file; On NT systems, the message files are kept in the directory tree specified by the `PMDF_POPSTORE_MESSAGES` registry entry; and, on OpenVMS systems, they are stored in the `PMDF_POPSTORE_MESSAGES:` directory tree. Read and write access to these files is controlled using private locks.

In the MessageStore, the messages are stored in a ready-to download format for IMAP, and the index and cache files in the appropriate folder are updated to include pre-calculated responses to common IMAP queries. This makes the MessageStore delivery process a bit slower in exchange for making all IMAP queries much faster.

The MessageStore message files are kept in a directory subtree with the user profiles. On UNIX systems, user profiles are located in the directory tree specified by the `PMDF_POPSTORE_PROFILES` option in the PMDF tailor file (usually `/pmdf/user`); on Windows NT systems, the directory tree is specified by the `PMDF_POPSTORE_PROFILES` registry entry; on OpenVMS systems, they are stored in the `PMDF_POPSTORE_PROFILES:` directory tree. Read and write access to these files is controlled using private locks.

10.1.1 Validating Accounts

There is a channel keyword `validatelocalmsgstore` which is enabled by default on the inbound delivery channel. This causes PMDF to validate each envelope `To:` address destined for the popstore and MessageStore. The checks consist of the following three steps:

1. Check that the address specifies a valid account. If there is no valid account, the message is rejected for that addressee using a permanent error response (*e.g.*, a SMTP 5yz response), unless there is an entry in the forward database with that account name.
2. If there is a valid account, the account is then checked to see if it is marked DISMAIL. If it is marked DISMAIL, the message is rejected for that addressee using a permanent error response.
3. Finally, if the option `REJECT_OVER_QUOTA` is set to 1, check to see if the user is over quota. If so, reject the message for that addressee using a temporary error response (*e.g.*, an SMTP 4yz response). This handling is suppressed for the delivery channel itself which occasionally needs to requeue messages back to itself.

Use of this channel keyword enables PMDF to perform the above checks as a message is presented to PMDF. This allows PMDF to reject outright messages which should not be accepted and thus preventing cases where a message is received only to have to be bounced back to the sender by PMDF. Since the message is never accepted, network and CPU resources are not consumed receiving it, generating a non-delivery notification, and then sending that notification back to the originator.

Account validation may be disabled by specifying the channel keyword `validate-localnone` on the inbound delivery channel.

10.1.2 Storage Quotas

At time of delivery, the delivery channel checks the message size against the user's current disk usage and allowed quota. If storage of the message would exceed the sum of the user's primary and overdraft quotas, then one of the following three actions is taken:

1. If the delivery channel is marked with the `exquota` channel keyword, then the message is delivered to the user.
2. If the delivery channel is marked with the `holdexquota` channel keyword, then delivery of the message is deferred until either the user has space for the message, or the message "times out" and is returned by the PMDF message bouncer.
3. If the delivery channel is marked with the `noexquota` channel keyword, then the message is returned as undeliverable to its sender.

Case 2 above is the default case. Note that users with a primary quota value of zero have unlimited storage quota.

10.1.3 Delivery Notifications

The delivery channel supports the generation of delivery notifications as described in the NOTARY specifications, RFCs 1891 and 1894. Note that the delivery channel itself handles the success and failure notifications while the PMDF message bouncer process generates delay notifications.

10.2 The Message Bouncer

The popstore has its own message bouncer job which runs once a day around midnight and deletes old, stored messages whose age exceeds the maximum allowed storage age. That age is controlled with the RETURN_AFTER popstore option as described in Chapter 3. By default, messages older than 14 days are deleted. If one or more of the message's recipients have not read the message, a non-delivery notification is sent to the message's originator. Through the RETURN_AFTER option, the maximum allowed age can be changed. When the maximum age is set to zero, messages are retained until explicitly deleted by their respective recipients or manually deleted by a popstore manager.

In addition, the message bouncer performs sanity checks on the popstore. For instance, if the system crashes during a message delete operation, it is possible that the message file can be left behind with no pointers to it. Such orphans are detected by the message bouncer.[1]

Note that the message bouncer is not used by the MessageStore.

[1] When RETURN_AFTER is set to zero, the message bouncer still performs these sanity checks.

11 Servers

There are five servers which interact with the popstore and MessageStore: two POP3 servers, the MessageStore's IMAP server, the poppassd server, and the HTTP server. Each of these servers are documented in the *PMDF System Manager's Guide*.

11.1 POP3 Servers

The configuration of the POP3 servers is discussed in the *PMDF Installation Guide*. Refer to that manual for directions on configuring the POP3 server.

On UNIX and OpenVMS platforms, a "legacy" multi-threaded POP3 server which serves out both native mailboxes as well as popstore mailboxes is provided. Specifically, on UNIX systems the legacy POP3 server serves out both popstore and BSD-style mailboxes. On OpenVMS systems the legacy server serves out both popstore and VMS MAIL mailboxes. When a client attempts to authenticate against the legacy POP3 server, the server first checks for a popstore account for the username. If there is a match, the associated popstore mailbox is served out. If there is no match, then the server checks for a native mailbox and serves it out if there is one. If you are not using the native mailbox feature, the non-legacy POP3 server should be used instead as it has better performance and security characteristics.

The "non-legacy" POP3 server is a multi-threaded server which is capable of serving out both popstore and MessageStore mailboxes. It cannot, however, serve out native mailboxes. This server has improved performance over the legacy POP3 server. In addition, this server is required to take advantage of the multiple domain facility of the popstore.

Both POP3 servers use PMDF's authentication services and as such supports plain text passwords as well as `APOP`, `CRAM-MD5` and `DIGEST-MD5` authentication. Additionally, the server supports SASL (RFC 2222). For further information on the POP3 server or authentication services, consult the "Connection Authentication and Password Management" chapter of the *PMDF System Manager's Guide*.

11.2 IMAP Server

The configuration of the IMAP server is discussed in the *PMDF Installation Guide*. Refer to that manual for directions on configuring the IMAP server.

The MessageStore IMAP server is a high-performance, multi-threaded server which serves out MessageStore accounts. The MessageStore stores pre-computed responses to many IMAP queries at delivery time so that the IMAP server has much less work to do.

The IMAP server uses PMDF's authentication services and as such supports plain text passwords as well as `APOP`, `CRAM-MD5` and `DIGEST-MD5` authentication. Additionally, the server supports SASL (RFC 2222). For further information on the POP3 server or authentication services, consult the "Connection Authentication and Password Management" chapter of the *PMDF System Manager's Guide*.

11.3 poppassd Server

`Poppassd` is an *ad hoc* protocol for changing POP, IMAP, and native account passwords. This protocol is used by several clients such as Eudora and Mulberry. PMDF provides a poppassd server which can be used by clients to change their password. The server is configured as part of the POP3 and IMAP server configuration process. Consult the *PMDF Installation Guide* for details on configuring the POP3 and IMAP servers.

Note that under the `poppassd` protocol, plain text passwords are exchanged in the clear between the client and poppassd server — even when the client is otherwise using `APOP`, `CRAM-MD5`, or `DIGEST-MD5` for authentication. Owing to this limitation of poppassd, its usage should be questioned in settings where `APOP`, `CRAM-MD5`, `DIGEST-MD5` or similar password security is expected.

11.4 HTTP Server

The PMDF HTTP server is configured as part of the PMDF MTA configuration. See the *PMDF Installation Guide* for details. The HTTP server provides the web-based management interfaces for both the popstore and MessageStore as described in Chapter 4.

12 Application Program Interface (API)

The popstore provides an application program interface (API). With this interface, sites can write their own code to directly create, manipulate, and delete user accounts as well as interface the popstore to billing utilities and other management interfaces.

12.1 Fundamentals

The popstore API is a re-entrant, thread-safe API. User and message contexts are shared between API client code and the API so as to facilitate the handling of multiple accounts and messages simultaneously from either single or multi-threaded programs. Note that writers of multi-threaded API client code should read Section 12.9.

Access to the underlying profile and message files is controlled and coordinated by the API subroutines. Programs must not attempt to access those files directly. The contexts used by the API are not opaque: clients of the API are provided with structure definitions for those contexts and can view the individual data fields in them. However, these fields must be treated as read only data and must not be changed. Indeed, changes to them generally will not affect the actual on-disk data. To change the actual, underlying on-disk data, call the appropriate API subroutines to effect the desired change.

For C programmers, a C header file declaring basic constants, data types, structures, and API subroutines in provided. This header file is the `popstore.h` header file and is, on UNIX and NT platforms, located in the `/pmdf/include/` directory. On OpenVMS platforms, it is located in the `PMDF_COM:` directory.

In order to use the popstore API, you must first call the `POPSTORE_init` subroutine so as to initialize the API. When finished with the API, be sure to call `POPSTORE_end`. The sections 12.2–12.8 provide information and code examples for performing common tasks. Section 12.12 provides complete descriptions for each API subroutine. See Section 12.10 for information on linking programs against the API.

12.2 Creating Accounts

There are two approaches to creating accounts: either create an account from scratch with no default settings for fields in the account's profile, or create the account by copying fields from another account profile and then subsequently changing desired fields in the new account. Either approach is acceptable; both methods are described below. The advantage to the latter approach is that by copying the `default` account, the site's default account settings are used as an initial basis for the new account.

12.2.1 From Scratch

The subroutines POPSTORE_user_create_set and POPSTORE_user_create are used to create an account from scratch. With a variable of type

```
POPSTORE_user_context *user_context
```

initialized to NULL, make successive calls to POPSTORE_user_create_set to set values for the account to be created. The username field must be set, and it is always a good idea to set a password too unless you genuinely want the account to require no password to access it. After setting the desired fields, call POPSTORE_user_create to actually create the account and dispose of the context created by the first call to POPSTORE_user_create_set. That call will write to disk the profile file for the account and create an entry in the user database. If the username for the account conflicts with another existing account, the account will not be created and a POPSTORE_USEREXISTS error will be returned.

Note that the POPSTORE_user_create_set subroutine will not allow the MANAGE usage flag to be set or cleared without first calling POPSTORE_manage to explicitly authorize such actions. (cleared). Note further that regardless of any value set for the last billing field, that field will be set to the current date and time when POPSTORE_user_create is called.

An example program which creates the profile

```
# pmdf popstore show joe
Username:           joe
Owner:              Joe User
Group:              staff
Store Type:         popstore
Usage flags:
Site-defined:

Last pwd change:    No time recorded
Last connect:       No time recorded
Last disconnect:    No time recorded
Total connect time: 0 00:00:00
Total connections:  0
Past block days:    0
Last billing:       Tue Nov 19 10:25:18 2012

Message count:              0 (0 total messages received)
Quota used:              0.00 Kbytes
Primary quota:          90.00 Kbytes
Overdraft quota:        10.00 Kbytes
```

is shown in Example 12–1. The following items of note are identified with callouts in the sample program:

1 check is a subroutine used to check the return status from each call to the popstore API. In the event of an error, an error message is output, the API shutdown, and the program exited.

2 Before making any other API calls, the API is first initialized with a call to POPSTORE_initialize.

3 Values for fields in the structure are set as desired.

4 The account is created with a call to POPSTORE_user_add.

5 Finally, the API is shutdown.

Example 12–1 Creating a New Account from Scratch

```
/*****************************************************************
 *                                                               *
 *  create_sample.c                                              *
 *  Sample subroutine to create a popstore account from scratch.  *
 *                                                               *
 *****************************************************************/

#include <stdio.h>
#include <stdlib.h>
#ifdef __VMS
#  include "pmdf_com:popstore.h"
#else
#  include "/pmdf/include/popstore.h"
#endif

void check (int stat) 1
{
  if (stat == POPSTORE_SUCCESS) return;
  fprintf (stderr, "Error %d: %s\n", stat, POPSTORE_error_to_text (stat));
  (void) POPSTORE_end ();
  exit (1);
}

main ()
{
  POPSTORE_user_context *ctx;

  /*
   *  Initialize the popstore
   */
  check (POPSTORE_init (1, NULL, "create_sample", 13)); 2

  /*
   *  Set values for various fields
   */
  ctx = NULL;
  check (POPSTORE_user_create_set (&ctx, POPSTORE_SET_USERNAME, 3, "joe")); 3
  check (POPSTORE_user_create_set (&ctx, POPSTORE_SET_GROUP_NAME, 5, "staff"));
  check (POPSTORE_user_create_set (&ctx, POPSTORE_SET_OWNER, 10, "Joe User"));
  check (POPSTORE_user_create_set (&ctx, POPSTORE_SET_PASSWORD, 6, "secret"));
  check (POPSTORE_user_create_set (&ctx, POPSTORE_SET_QUOTA,     1024*90));
  check (POPSTORE_user_create_set (&ctx, POPSTORE_SET_OVERDRAFT, 1024*10));

  /*
   *  Create the account
   */
  check (POPSTORE_user_create (&ctx)); 4
```

Example 12–1 Cont’d on next page

Example 12–1 (Cont.) Creating a New Account from Scratch

```
  /*
   *  All done
   */
  (void) POPSTORE_end (); 5
}
```

12.2.2 By Copying

New accounts can also be created by copying an old account with the `POPSTORE_user_copy_d` subroutine and then changing the value of fields with `POPSTORE_user_update`. For instance, both of the popstore management interfaces create new accounts by copying the `default` account to the new account and then changing the requested fields in the new account.

Note that the `POPSTORE_user_update` subroutine will not allow the `MANAGE` usage flag can not be set or cleared without first calling `POPSTORE_manage` to explicitly authorize such actions.

The code in Example 12–2 creates the account `joe` by first copying the `default` account and then changing fields in the newly created `joe` account with `POPSTORE_user_update`. The account so created is identical to that created from scratch in Example 12–1. The following items of note are identified with callouts in the example program:

1. check is a subroutine used to check the return status from each call to the popstore API. In the event of an error, an error message is output, the API shutdown, and the program exited.
2. A macro to help construct an item list.
3. Before making any other API calls, the API is first initialized with a call to `POPSTORE_initialize`.
4. Values for entries in the item list are set. These entries describe settings to make for fields in the new account.
5. The new account is created. It is identical to the `default` account except for the username and password fields.
6. Fields in the new account are changed to reflect the desired values for those fields.
7. Finally, the API is shutdown.

Example 12–2 Creating a New Account by Copying the `default` Account

Example 12–2 Cont'd on next page

Example 12–2 (Cont.) Creating a New Account by Copying the `default` Account

```
/****************************************************************
 *                                                              *
 *  copy_sample.c                                               *
 *  Sample subroutine to create a popstore account by copying   *
 *  the default account.                                        *
 *                                                              *
 ****************************************************************/

#include <stdio.h>
#include <stdlib.h>

#ifdef __VMS
#  include "pmdf_com:popstore.h"
#else
#  include "/pmdf/include/popstore.h"
#endif

static POPSTORE_user_context *user_context = NULL;

void check (int stat) 1
{
  if (stat == POPSTORE_SUCCESS) return;
  fprintf (stderr, "Error %d: %s\n", stat, POPSTORE_error_to_text (stat));
  if (user_context) (void) POPSTORE_user_end (user_context);
  (void) POPSTORE_end ();
  exit (1);
}

#define PUSH(list,code,addr,len) \ 2
  list[item_index].item_code      = (code); \
  list[item_index].item_address   = (addr); \
  list[item_index++].item_length  = (len);

main ()
{
  POPSTORE_item_list item_list[5];
  int item_index   = 0;
  char group[]     = "staff";
  char owner[]     = "Joe User";
  uint32 quota     = 1024*90;
  uint32 overdraft = 1024*10;
  char errmsg[80];
  int errmsg_len;

  /*
   *  Initialize the popstore
   */
  check (POPSTORE_init (1, NULL, "copy_sample", 11)); 3
```

Example 12–2 Cont’d on next page

Example 12–2 (Cont.) Creating a New Account by Copying the `default` Account

```
  /*
   *  Build an item list describing the fields to set new values for
   */
  PUSH (item_list, POPSTORE_SET_GROUP_NAME, group,      strlen (group)); 4
  PUSH (item_list, POPSTORE_SET_OWNER,      owner,      strlen (owner));
  PUSH (item_list, POPSTORE_SET_QUOTA,      &quota,     0);
  PUSH (item_list, POPSTORE_SET_OVERDRAFT,  &overdraft, 0);
  PUSH (item_list, POPSTORE_SET_END,        NULL,       0);

  /*
   *  Create the new account as a duplicate of the default account but
   *  with the desired new username and password
   */
  check (POPSTORE_user_copy_d (NULL, 0, "default", 7,  5
                               NULL, 0, "joe", 3,
                               "secret", 6, 0));

  /*
   *  Change the account settings
   */
  check (POPSTORE_user_begin_d (NULL, 0, &user_context, "joe", 3,
                                POPSTORE_NOACCOUNTING, "copy_sample", 11));
  check (POPSTORE_user_update (user_context, item_list,  6
         errmsg, &errmsg_len, sizeof(errmsg)));
  check (POPSTORE_user_end (user_context));
  user_context = NULL;

  /*
   *  All done
   */
  POPSTORE_end ();
}
```

12.3 Modifying Accounts

Fields in an existing account can be modified using the POPSTORE_user_update subroutine. This subroutine accepts as input a user context created by POPSTORE_user_begin_d and an item list describing the fields to change and the new values to use for those fields. Before making any changes, the validity of the entries in the item list is first determined. Only if the entries are all correct, are any changes then made. The changes are made atomically: the profile is locked, read, all changes made, written back to disk, and then unlocked. In the event of an unexpected error, the old profile data is restored if possible.

An example of using POPSTORE_user_update is given in Example 12–2. When modifying accounts, keep the following notes in mind:

1. The username field for an account can not be changed with POPSTORE_user_update. To change that field, use POPSTORE_user_copy_d specifying a non-zero value for the **do_rename** argument of that subroutine.

2. The MANAGE flag can only be set or cleared if a prior call to POPSTORE_manage has been made which enables the ability to set that bit.
3. When changing the count of stored messages for an account, the value can only be decreased. When the count is decreased from M to the value N, then the oldest M-N messages are deleted (*e.g.*, when M=N, all messages are deleted). Note that unread deleted messages are simply deleted: they are not returned to their originator. If they should instead be returned, use POPSTORE_message_return.
4. When an account's past block days field is changed, the past block days remainder field is reset to zero.
5. If just changing the account's password, consider using POPSTORE_user_pw_change_d.

12.4 Deleting Accounts

Accounts are deleted with POPSTORE_user_delete_d. When an account is deleted, any unread messages for that account can optionally be returned to their originators. Shown below is the call which would be made to delete the account jdoe, returning any unread messages to their originators:

```
stat = POPSTORE_user_delete_d ("jdoe", 4, 1);
```

12.5 Listing Accounts

Formatted listings of accounts can be generated with the POPSTORE_format_profiles_d procedure. Note that this same functionality is provided at the command line with the POPSTORE utility; see Chapter 9 for details.

In order to use POPSTORE_format_profiles_d, you must first decide upon a layout for the listing and then implement that layout with a formatting file. See the description of the POPSTORE_format_profiles_d subroutine for further details. Once a formatting file has been developed, it can then be used to format listings of user accounts. In the example code shown in Example 12–4, an actual formatting file used by the POPSTORE utility is used. That formatting file, as can be seen in Example 12–3, lists for each account the username, stored message count, and used quota.

Example 12–3 Formatting File for Account Listings

```
%first{                                  Message  Quota used}
%first{ Username                           Count    (kbytes)}
%first{ ---------------------------------------------------}
%flags_manage{ |*}%username{%-32s} %message_count{%7u}    %quota_used_k{%8.2f}
%last{ ------------------------------------------------------------}
%last{*Note: privileged users are flagged with an asterisk}
```

Example 12–4 Generating Account Listings

Example 12–4 Cont'd on next page

Example 12–4 (Cont.) Generating Account Listings

```
/****************************************************************
 *                                                              *
 *  format_sample.c                                             *
 *  Sample subroutine to generate an account listing.          *
 *                                                              *
 ****************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#ifdef __VMS
#  include "PMDF_COM:popstore.h"
#else
#  include "/pmdf/include/popstore.h"
#endif

void check (int stat)
{
  if (stat == POPSTORE_SUCCESS) return;
  fprintf (stderr, "Error %d: %s\n", stat, POPSTORE_error_to_text (stat));
  (void) POPSTORE_end ();
  exit (1);
}

int output (void *ctx, char *line, int len, int eol, int literal)
{
  if (!eol) printf ("%.*s", len, line);
  else printf ("%.*s\n", len, line);
  return (POPSTORE_SUCCESS);
}

main ()
{
  POPSTORE_format_element *format;

  /*
   *  Initialize the popstore
   */
  check (POPSTORE_init (0, NULL, "format_sample", 13));

  /*
   *  Read in the formatting file and get a formatting context
   */
  check (POPSTORE_format_read (&format, "popmgr_profile_brief.txt", 24,
#ifdef __VMS
                               "PMDF_HTTP_POPSTORE:[000000]", 27));
#else
                               "/pmdf/www/popstore/", 19));
#endif
```

Example 12–4 Cont'd on next page

Example 12–4 (Cont.) Generating Account Listings

```
  /*
   *  Display the profiles
   */
  check (POPSTORE_format_profiles_d (format, NULL, 0, NULL, 0, NULL, 0, NULL,
                                     NULL, output));

  /*
   *  Dispose of the formatting context
   */
  check (POPSTORE_format_dispose (format));

  /*
   *  All done
   */
  (void) POPSTORE_end ();
}
```

To generate a simple listing of accounts, the `POPSTORE_user_list_d` subroutine can prove sufficient. That subroutine is primarily intended for cases where a program needs to obtain one-by-one the usernames for each account. Sample code using `POPSTORE_user_list_d` to list all accounts and then the first account with a username starting with the letter “z” is given in Example 12–5.

Example 12–5 Simple Account Listing

```
/****************************************************************
 *                                                              *
 *  list_sample.c                                               *
 *  Sample subroutine to generate an account listing.           *
 *                                                              *
 ****************************************************************/

#include <stdio.h>

#ifdef __VMS
#  include "PMDF_COM:popstore.h"
#else
#  include "/pmdf/include/popstore.h"
#endif

main ()
{
  POPSTORE_list_context *list_context;
  int stat, userlen;
  char user[POPSTORE_MAX_USER_LEN+1];

  /*
   *  Initialize the popstore
   */
  stat = POPSTORE_init (1, NULL, "list_sample", 11);
  if (stat != POPSTORE_SUCCESS) exit (1);

  /*
   *  List all accounts
   */
  printf ("---- Listing of all accounts ----\n");
  list_context = NULL;
  stat = POPSTORE_SUCCESS;
  while (stat == POPSTORE_SUCCESS) {
     stat = POPSTORE_user_list_d (&list_context, "*", 1, NULL, 0, NULL, 0,
                                  user, &userlen, POPSTORE_MAX_USER_LEN + 1);
     if (stat == POPSTORE_SUCCESS) printf ("%s\n", user);
  }

  printf ("\n---- The first account starting with the letter \"z\" ----\n");
  list_context = NULL;
  stat = POPSTORE_user_list_d (&list_context, "z*", 2, NULL, 0, NULL, 0,
                               user, &userlen, POPSTORE_MAX_USER_LEN + 1);
  if (stat == POPSTORE_SUCCESS) {
     printf ("%s\n", user);
     /* Done with this context; dispose of it now */
     (void) POPSTORE_user_list_abort (&list_context);
  }
  else printf ("**** No accounts begin with the letter \"z\" ****\n");

  /*
   *  All done
   */
  (void) POPSTORE_end ();
}
```

12.6 Billing Accounts

Sites who want to bill users for usage should use the POPSTORE_user_billing_d subroutine to perform "atomic" billing operations. That subroutine locks a user profile, extracts billing information, resets accounting fields, writes the user profile back to disk, unlocks the profile, and then returns the extracted billing information to the caller.

The billing period is the intervening time between when POPSTORE_user_billing_d was last called and the current billing time supplied to POPSTORE_user_billing_d. The time when POPSTORE_user_billing_d was last called is stored in each account's last billing profile field.[1]

The extracted billing information includes the total connect time and past block days of message storage used during the billing period. After that information is extracted, those two fields are set to the value zero. In addition, the last billing time fields in the profile and message list are set to the new billing time supplied to POPSTORE_user_billing_d, and this data is then written out to the on-disk profile file.

Example 12–6 shows sample code which uses POPSTORE_user_billing_d in conjunction with POPSTORE_user_list_d to bill each account whose associated username starts with the letter "a" and is in the students management group.

Example 12–6 Account Billing Operations

```
/****************************************************************
 *                                                              *
 *  bill_sample.c                                               *
 *  Sample subroutine to bill each popstore account.            *
 *                                                              *
 ****************************************************************/

#include <stdio.h>
#include <times.h>

#ifdef __VMS
#  include "PMDF_COM:popstore.h"
#else
#  include "/pmdf/include/popstore.h"
#endif

main ()
{
  time_t billing_time;
  POPSTORE_user_data data;
  POPSTORE_list_context *list_context;
  int stat, stat2, userlen;
  char user[POPSTORE_MAX_USER_LEN + 1];
```

Example 12–6 Cont'd on next page

[1] For new accounts, that field is initialized to the time at which the account was created.

Example 12–6 (Cont.) Account Billing Operations

```
  /*
   *  Initialize the popstore
   */
  stat = POPSTORE_init (1, NULL, "list_sample", 11);
  if (stat != POPSTORE_SUCCESS) exit (1);

  /*
   *  Output a heading
   */
  printf ("                                  Connect    Block\n");
  printf ("Username                             Time     Days\n");
  printf ("--------------------------------  -------  -------\n");

  /*
   *  Now, loop over each account and bill it
   */
  billing_time = time (NULL);
  list_context = NULL;
  do {
     stat = POPSTORE_user_list_d (&list_context, "a", 1, NULL, 0,
                                  "students", 8, user, &userlen,
                                  POPSTORE_MAX_USER_LEN + 1);
     if (stat == POPSTORE_SUCCESS) {
        stat2 = POPSTORE_user_billing_d (NULL, 0, user, userlen, billing_time,
                                         &data);
        if (stat2 == POPSTORE_SUCCESS) {
           printf ("%-32s   %7u  %7u\n", user, data.total_connect,
                   data.past_block_days);
        } else {
           fprintf (stderr, "Unable to bill %.*s; error = %d\n", userlen, user,
                    stat2);
           fprintf (stderr, "%s\n", POPSTORE_error_to_text (stat2));
        }
     }
  } while (stat == POPSTORE_SUCCESS);
  if (stat != POPSTORE_EOM) {
     fprintf (stderr, "POPSTORE_user_list_d returned the error %d\n", stat);
     fprintf (stderr, "%s\n", POPSTORE_error_to_text (stat));
  }

  /*
   *  All done
   */
  (void) POPSTORE_end ();
}
```

12.7 Storing Messages

To store a message into the message store, enqueue it as a mail message to PMDF with the desired popstore recipients given as the message's envelope To: recipients. See the *PMDF Programmer's Reference Manual* for information on using the PMDF API to enqueue messages to PMDF.

12.8 Accessing Messages

A user's messages are accessed by first obtaining a user context, `user_context`, with `POPSTORE_user_begin_d`. The list of stored messages is the array of `POPSTORE_message_ref` structures which start at the address

```
user_context->messages
```

and contain

```
user_context->profile->message_count
```

array elements. For example, the following code fragment counts the number of unread messages stored for the user `sue`:

Example 12–7 Message Lists

```
int i, new_count, stat;
POPSTORE_message_ref *msg_ptr;
POPSTORE_user_context *user_context;

...

stat = POPSTORE_user_begin_d (&user_context, "sue", 3, POPSTORE_NOACCOUNTING,
                              "new message count", 17);
new_count = 0;
for (i = 0, msg_ptr = user_context->messages;
     i < user_context->profile->message_count;
     i++, msg_ptr++)
  if (!(msg_ptr->flags & POPSTORE_MFLAGS_READ)) new_count++;
stat = POPSTORE_user_end (user_context);
printf ("Sue has %d new messages\n", new_count);

...
```

Individual messages are accessed and manipulated with the `POPSTORE_message` subroutines. When accessing or manipulating a message, the message is referenced using its "message index". The first message stored for a user has message index value `1`, the second index value `2`, the third index value `3`, and so on.

The code shown in Example 12–8 displays each new message for the user `sue`, marking each message as read after displaying it.

Example 12–8 Displaying New Messages

```
/***************************************************************
 *                                                             *
 *  read_sample.c                                              *
 *  Sample subroutine to display new messages for an account.  *
 *                                                             *
 ***************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#ifdef __VMS
#  include "pmdf_com:popstore.h"
#else
#  include "/pmdf/include/popstore.h"
#endif

void check (int stat)
{
  if (stat == POPSTORE_SUCCESS) return;
  fprintf (stderr, "Error %d: %s\n", stat, POPSTORE_error_to_text (stat));
  (void) POPSTORE_end ();
  exit (1);
}

void display_message (POPSTORE_user_context *user_context, int msg_index)
{
  char buffer[1024+1];
  int len, message_context, stat;

  printf ("==========================================================\n");
  printf ("Message %d\n", msg_index);
  printf ("==========================================================\n");

  if (POPSTORE_SUCCESS !=
      POPSTORE_message_begin (user_context, msg_index, &message_context,
                              NULL, 0, NULL, 0)) return;
  do {
     stat = POPSTORE_message_read (message_context, buffer, 1024, &len);
     if (stat != POPSTORE_READERROR) {
        buffer[len] = '\0';
        printf ("%s", buffer);
     }
  } while (stat == POPSTORE_SUCCESS);
  (void) POPSTORE_message_end (message_context);
  printf ("==========================================================\n");
}

main ()
{
  POPSTORE_message_ref *msg_ptr;
  int i, stat;
  POPSTORE_user_context *user_context;

  /*
   *  Initialize the popstore
   */

  check (POPSTORE_init (1, NULL, "read_sample", 11));
```

Example 12–8 Cont'd on next page

Example 12–8 (Cont.) Displaying New Messages

```
  /*
   *  Access the popstore account
   */
  check (POPSTORE_user_begin_d (NULL, 0, &user_context, "sue", 3,
                                POPSTORE_ACCOUNTING, "read_sample", 11));

  /*
   *  Display the new messages
   */
  msg_ptr = user_context->messages;
  for (i = 0, msg_ptr = user_context->messages;
       i < user_context->profile->message_count;
       i++, msg_ptr++)
    if (!(msg_ptr->flags & POPSTORE_MFLAGS_READ)) {
       display_message (user_context, i+1);
       (void) POPSTORE_message_mark_read (user_context, i+1);
  }

  /*
   *  Deaccess the account
   */
  (void) POPSTORE_user_end (user_context);

  /*
   *  And shut down the popstore
   */
  (void) POPSTORE_end ();
}
```

12.9 Using the API from Multi-threaded Programs

Multi-threaded programs need to register mutex handling subroutines with the PMDF API subroutine `PMDF_set_mutex` prior to initializing either the PMDF or popstore APIs. In addition, when calling `POPSTORE_initialize`, supply the address of the sleep subroutine provided to `PMDF_set_mutex`.

12.10 Compiling and Linking Programs

OpenVMS Systems

To declare the API subroutines, data structures, and constants, C programs should use the `PMDF_COM:popstore.h` header file.

Linking programs to the popstore API is accomplished with a link command of the form

```
$ LINK program,PMDF_EXE:pmdfshr_link.opt/OPT
```

where *program* is the name of the object file to link.

Solaris Systems

To declare the API subroutines, data structures, and constants, C programs should use the `/pmdf/include/popstore.h` header file.

Linking a C program to the API is accomplished with a link command of the form

```
% cc -R/pmdf/lib/ -L/pmdf/lib/ -o program program.c \
        -lpmdf -lsocket -lintl -lnsl -lm -ldb -lldapv3
```

where *program* is the name of your program.

NT Systems

To declare the API subroutines, data structures, and constants, C programs should use the `C:\pmdf\include\popstore.h` header file.

When linking programs to the API with the Microsoft C/C++ compiler, use the switches

```
-mD -D_WIN32_WINNT=0x0400 C:\pmdf\lib\libpmdf.lib
```

12.11 Basic Constants, Types, and Data Structures

Basic constants and data types used by the popstore are listed in Tables 12–1 and 12–2. The data structures describing a user profile, `POPSTORE_user_data`, and a message list, `POPSTORE_message_list`, are shown in Sections 12.11.1 and Section 12.11.2. A user context as obtained from the POPSTORE_user_begin_d subroutine, is a pointer to a `POPSTORE_user_context` structure and is described in Section 12.11.3. Fields in that structure point to data structures containing the user's profile and message list. All of these constants, data types, and structures are declared in the `popstore.h` header file located in the `/pmdf/include/` directory on UNIX and NT platforms and the `PMDF_COM:` directory on OpenVMS.

Table 12–1 Constants

Constant	Value	Description
ALFA_SIZE	252	A basic PMDF string size
BIGALFA_SIZE	1024	A basic PMDF string size
POPSTORE_FLAGS_DELETE	16	Bit mask for the DELETE usage flag bit
POPSTORE_FLAGS_DISMAIL	2	Bit mask for the DISMAIL usage flag bit

Table 12–1 (Cont.) Constants

Constant	Value	Description
POPSTORE_FLAGS_DISUSER	1	Bit mask for the DISUSER usage flag bit
POPSTORE_FLAGS_LOCKPWD	4	Bit mask for the LOCKPWD usage flag bit
POPSTORE_FLAGS_MANAGE	8	Bit mask for the MANAGE usage flag bit
POPSTORE_FULL_UIDL_LEN	23	Length in bytes of a full message UIDL derived
POPSTORE_MAX_DOMAIN_LEN	40	Maximum length in bytes of a user domain name
POPSTORE_MAX_FILE_LEN	1024	Maximum length in bytes for full file paths
POPSTORE_MAX_GROUP_LEN	16	Maximum length in bytes of a management group name
POPSTORE_MAX_OWN_LEN	40	Maximum length in bytes for the profile owner field
POPSTORE_MAX_PRIV_LEN	64	Maximum length in bytes for the profile site-defined private data field
POPSTORE_MAX_PWD_LEN	32	Maximum length in bytes for the profile password field
POPSTORE_MAX_USER_LEN	32	Maximum length in bytes for the profile username field
POPSTORE_MFLAGS_READ	1	Bit mask for the READ message flag bit
POPSTORE_MSG_FILE_FORMAT_VERSION	0	Current value for message file format version
POPSTORE_MSG_NAME_LEN	19	Length in bytes of a message file name
POPSTORE_USERDATA_VERSION	2	Current value for the profile file format version

Table 12–2 Basic Data Types

Type	Size (bytes)	Underlying data type
int32	4	int
ubyte	1	unsigned char
uint32	4	unsigned int
ushort	2	unsigned short int

12.11.1 POPSTORE_user_data Structure

User profile information is stored in a `POPSTORE_user_data` structure of the form shown below. To change user profile information, the `POPSTORE_user_set` or `POPSTORE_user_update` subroutines should be used. The former is used when creating an account—an account which does not yet exist. The latter is used to modify an existing account. The profile information for an existing account is obtained with the `POPSTORE_user_begin_d` subroutine. That subroutine returns a pointer to a `POPSTORE_user_context` structure; the profile data is pointed at by the `profile` field in that structure.

The layout of the POPSTORE_user_data structure is shown below:

```
typedef struct {
  ubyte  version;
  ubyte  store_type;
  ushort flags;
  ubyte  ulen;
  ubyte  plen;
  ubyte  olen;
  ubyte  slen;
  char   username[POPSTORE_MAX_USER_LEN];
  char   password[POPSTORE_MAX_PWD_LEN];
  char   owner[POPSTORE_MAX_OWN_LEN];
  ubyte  private[POPSTORE_MAX_PRIV_LEN];
  uint32 quota;
  uint32 return_after;
  uint32 overdraft;
  time_t last_billing;
  uint32 total_conntections;
  time_t last_connect;
  time_t last_pwd_change;
  time_t last_disconnect;
  uint32 total_connect;
  uint32 past_block_days;
  uint32 past_block_days_remainder;
  uint32 message_count;
  uint32 quota_used;
  uint32 received_messages;
  uint32 received_bytes;
  ubyte  reserved5[3];
  ubyte  glen;
  char   group[POPSTORE_MAX_GROUP_LEN];
} POPSTORE_user_data;
```

The interpretation of these fields are as follows:

version
Data structure version number. The current version number for the data structure is given by the POPSTORE_USERDATA_VERSION constant.

store_type
Type of message store used for this profile. Value will typically be zero POPSTORE_STORE_TYPE_POP.

flags
Bit masked field containing usage flag settings. The bits in this field can be tested with the POPSTORE_FLAGS_ constants.

ulen
Length in bytes of the value stored in the username field.

plen
Length in bytes of the value stored in the password field. The value of this field is encrypted.

`olen`
Length in bytes of the value stored in the owner field.

`slen`
Length in bytes of the value stored in the private field.

`username`
The account username. This field is not null terminated; the length of the value stored in this field is given by the value of the ulen field.

`password`
The account password. The value stored in this field is encrypted and is not null terminated. The encrypted value of the plen field gives the length of the value stored in this field.

`owner`
Account owner information. This field is not null terminated; the length of the value stored in this field is given by the value of the olen field.

`private`
Site-defined private data field. The length of the value stored in this field is given by the value of the slen field. This field is not null terminated.

`quota`
Primary message storage quota measured in units of bytes. A value of zero for this field conveys unlimited storage quota.

`return_after`
Field reserved for future use.

`overdraft`
Message overdraft storage quota.

`last_billing`
Time of last billing as measured in seconds since 1 January 1970. This field is set to the current time when the account is first created.

`total_connections`
Cumulative count of connections made to the account since either the account was created or this field last cleared.

`last_connect`
Time of last connection attempt as measured in seconds since 1 January 1970. This field is set for both successful and unsuccessful login attempts. The time is recorded when the user context is created with `POPSTORE_user_begin_d` and written to the profile when the context is disposed of with `POPSTORE_user_end`.

`last_pwd_change`
Time of last password change as measured in seconds since 1 January 1970. If the value is 0, then the account is pre-expired. This field is set whenever the password is successfully changed, or when the account is set to pre-expired or not pre-expired through the web-based management interface or the command line management utility.

`last_disconnect`
Time of last disconnect as measured in seconds since 1 January 1970. This field is set for both successful and unsuccessful login attempts. The time is recorded when the user context is disposed of with `POPSTORE_user_end`.

`total_connect`
Total number of seconds spent connected since the account was created or this field last cleared.

`past_block_days`
Cumulative block days for previously stored and since deleted or returned messages since the account was created or this field last cleared. The value of this field does not take into account messages currently held in the store.

`past_block_days_remainder`
Remainder field used in computation of block days. This field is measured in units of byte seconds.

`message_count`
Count of messages currently stored for the account.

`quota_used`
Number of bytes currently being consumed to store messages for the account. This value only takes into consideration the size of the underlying messages themselves and not the size of the actual container files.

`received_messages`
Cumulative count of messages received and stored for the account since either the account was created or this field last cleared.

`received_bytes`
Cumulative count of message bytes received and stored for the account since either the account was created or this field last cleared.

`reserved5`
Field reserved for future use.

`glen`
Length in bytes of the value stored in the `group` field.

`group`
Name of the management group to which this account belongs.

12.11.2 POPSTORE_message_ref Structure

A user's list of stored messages is stored as an array of zero or more array elements each of type `POPSTORE_message_ref`. To change information in the list (*e.g.*, mark messages for deletion), use the appropriate POPSTORE_message_ subroutine. The list of an account's stored messages is obtained with the POPSTORE_user_begin_d subroutine. That subroutine returns a pointer to a `POPSTORE_user_context` structure; the message list is pointed at by the `messages` field in that structure. The count of stored messages, and hence the count of array elements in the message list, is given by the user profile information also available from the user context structure. See, for instance, Example 12–7.

The layout of the `POPSTORE_message_ref` structure is shown below:

```
typedef struct {
  uint32 size;
  uint32 last_billing;
  time_t created;
  uint32 flags;
  char   uidl[4];
  char   filename[POPSTORE_MSG_NAME_LEN];
  ubyte  pad;
} POPSTORE_message_ref;
```

The interpretation of these fields are as follows:

`size`
Size in bytes of the message. This size includes CRLF terminators but does not include any dots used for dot stuffing.

`last_billing`
Time at which usage billing was last done for the storage of this message. Measured as the count of seconds since 1 January 1970.

`created`
Time at which the message file was created. Measured as the count of seconds since 1 January 1970.

`flags`
Bit masked field containing message status flag bits. The bits in this field can be tested with the `POPSTORE_MFLAGS_` constants. Currently, only one flag bit is stored in the message list: the read/unread bit accessed with the `POPSTORE_MFLAGS_READ` bit mask.

`uidl`
The message UIDL begins with this field and includes the `filename` field. The length in bytes of the UIDL is given by `POPSTORE_FULL_UIDL_LEN`. This portion of the UIDL is unique for each recipient of the underlying message file.

`filename`
This field forms part of the message UIDL and is the same for all popstore recipients of this particular message. This field also is the name of the underlying file containing the message.

`pad`
A padding byte. The value of this byte is set to zero thus making the full UIDL accessible as a null terminated string.

12.11.3 POPSTORE_user_context Structure

Information about a popstore account is returned to callers of POPSTORE_user_begin_d in the form of a pointer to a structure of type POPSTORE_user_context—a "user context". Fields in this structure must not be changed; use the appropriate API subroutines to effect the needed changes. When through with a user context, call POPSTORE_user_end to dispose of the context. Note that it is important that that call be made: not only does it dispose of allocated resources, it also updates accounting information for the account and performs any requested message deletion operations.

The layout of the POPSTORE_user_context structure is shown below:

```
typedef struct {
  time_t               connect;
  uint32               log_subid;
  int                  do_accounting;
  uint32               block_days;
  uint32               block_days_remainder;
  POPSTORE_user_data   *profile;
  POPSTORE_message_ref *messages;
  POPSTORE_string       filespec;
  char                  pad[3];
  POPSTORE_userdb_data *userdb_data;
  void                 *reserved0;
  char                  domain[POPSTORE_MAX_DOMAIN_LEN];
  int                   dlen;
} POPSTORE_user_context;
```

The interpretation of these fields are as follows:

connect
Time at which this context was established.

log_subid
Sub-identifier used for logging purposes.

do_accounting
Flag indicating whether or not accounting should be done for this context.

block_days
Accumulated block days for messages deleted with this context.

block_days_remainder
Accumulated block days roundoff for messages deleted with this context.

profile
The user's profile data. The value of this field is a pointer to the profile information for this user account.

messages
The user's list of stored messages. The value of this field is a pointer to an array of POPSTORE_message_ref elements. This array contains profile->message_count elements.

`filespec`
File specification for the underlying profile file.

`pad`
Alignment padding bytes.

`userdb_data`
This field will usually be `NULL`. It is a pointer to the account's user database record. That record is only obtained when the popstore is performing management operations and even then only seldom.

`reserved0`
Field reserved for future use.

`domain`
User domain associated with this account.

`dlen`
Length in bytes of the user domain associated with this account. A length of zero indicates the `default` domain.

12.12 Subroutine Descriptions

This section documents the individual popstore API subroutines. A brief description of each subroutine is given in Table 12–3 below.

Table 12–3 Subroutines included in the API

Subroutine name	Description
POPSTORE_command	Obsolete: use the POPSTORE_command_d subroutine
POPSTORE_command_d	Process a management command
POPSTORE_end	End usage of the API
POPSTORE_error_to_text	Convert a numerical error to a textual error message
POPSTORE_format_counters	Format PMDF channel counter information
POPSTORE_format_dispose	Dispose of a formatting context
POPSTORE_format_forwarding	Obsolete: use the POPSTORE_format_forwarding_d subroutine
POPSTORE_format_forwarding_d	Format forwarding information
POPSTORE_format_message	Format a stored message
POPSTORE_format_messages	Format a user's list of stored messages
POPSTORE_format_profile	Format a user profile
POPSTORE_format_profiles	Obsolete: use the POPSTORE_format_profiles_d subroutine
POPSTORE_format_profiles_d	Format a list of user profiles
POPSTORE_format_read	Read and parse a formatting file
POPSTORE_init	Initialize the API
POPSTORE_manage	Allow changing of the MANAGE usage flag
POPSTORE_message_begin	Access a stored message
POPSTORE_message_end	Deaccess a stored message
POPSTORE_message_mark_delete	Mark a user's message copy for deletion
POPSTORE_message_mark_nodelete	Mark a user's message to be retained

Table 12–3 (Cont.) Subroutines included in the API

Subroutine name	Description
POPSTORE_message_mark_noread	Mark a user’s message as being unread
POPSTORE_message_mark_read	Mark a user’s message as read
POPSTORE_message_read	Sequentially read a message
POPSTORE_message_return	Return a message to its originator
POPSTORE_user_begin	Obsolete: use the POPSTORE_user_begin_d subroutine
POPSTORE_user_begin_d	Access a user account
POPSTORE_user_billing	Obsolete: use the POPSTORE_user_billing_d subroutine
POPSTORE_user_billing_d	Perform billing operations
POPSTORE_user_copy	Obsolete: use the POPSTORE_user_copy_d subroutine
POPSTORE_user_copy_d	Copy or rename an existing account
POPSTORE_user_create	Create a new account
POPSTORE_user_create_dispose	Abort creating a new account
POPSTORE_user_create_set	Set the value of a field for an account to be created with POPSTORE_user_create
POPSTORE_user_delete	Obsolete: use the POPSTORE_user_delete_d subroutine
POPSTORE_user_delete_d	Delete a user account
POPSTORE_user_end	Deaccess a user account
POPSTORE_user_exists	Obsolete: use the POPOPSTORE_user_exists_d subroutine
POPSTORE_user_exists_d	See if a username specifies a valid account
POPSTORE_user_list	Obsolete: use the POPSTORE_user_list_d subroutine
POPSTORE_user_list_abort	Prematurely dispose of a list
POPSTORE_user_list_d	Return the usernames associated with each account within an accounting group
POPSTORE_user_pw_change	Obsolete: use the POPSTORE_user_pw_change_d subroutine
POPSTORE_user_pw_change_d	Change a user’s password
POPSTORE_user_pw_check	Perform an authentication check
POPSTORE_user_update	Update a field in an existing account

POPSTORE_command

Obsolete subroutine: use the more general POPSTORE_command_d subroutine.

FORMAT

```
int POPSTORE_command (command, command_len,
                      password_required,
                      user, user_len,
                      password, password_len,
                      usage, usage_len,
                      out_info, out_info_len,
                      context, output_proc)
      char  *command;
      int    command_len;
      int    password_required
      char  *user;
      int    user_len;
      char  *password;
      int    password_len;
      char  *usage;
      int    usage_len;
      char  *out_info;
      int   *out_info_len;
      void  *context;
      int  (*output_proc)();
```

ARGUMENTS

`command`
The command to process. Used for input only.

`command_len`
Length in bytes of the command specified with `command`. Used for input only.

`password_required`
Boolean flag indicating whether or not password authentication is required in order to process the command. Used for input only.

`user`
Name of the popstore user account to authenticate against when password authentication is required. Used for input only.

`user_len`
Length in bytes of the username specified with `user`. Used for input only.

`password`
User-supplied, plain text password to use when performing a password authentication. Used for input only.

`password_len`
Length in bytes of the plain text password specified with `password`. Used for input only.

`usage`
Usage description to be passed to site-supplied logging subroutines explaining the usage. Used for input only.

`usage_len`
Length in bytes of the usage description specified with `usage`. Used for input only.

`out_info`
Output buffer to receive post-processing information. Used for output only.

`out_info_len`
On input, the maximum length in bytes of the buffer pointed at by `out_info`. On output, the length in bytes of the information placed in the `out_info` buffer. Used for input and output.

`context`
Pointer to private client data to be passed to the client-supplied `output_proc` procedure. Used for input only.

`output_proc`
Address of a client-supplied subroutine to receive command processing output. Used for input only.

DESCRIPTION

Although still supported, this subroutine is now obsolete. Use the more general `POPSTORE_command_d` subroutine instead.

RETURN VALUES

POPSTORE_SUCCESS	Normal, successful completion.
POPSTORE_BADARG	Null value received for the `out_proc` argument.
POPSTORE_ERRORS	The command was parsed and executed; execution of the command resulted in some errors which were reported via the `output_proc` procedure.
POPSTORE_AUTHFAIL	Invalid password or username supplied. The command was not executed.
POPSTORE_BADHEXVAL	The command contained an illegal hexadecimal encoded character. The command was not executed.
POPSTORE_CMDTOOLONG	The command contained too many name=value pairs. The command was not executed.
POPSTORE_CMDUNKOWN	The command contained an unrecognized right-hand-side in a name=value pair. The command was not executed.
POPSTORE_CMDUNKOWNL	The command contained an unrecognized left-hand-side in a name=value pair. The command was not executed.
POPSTORE_NOMANAGE	Specified account lacks management privileges. The command was not executed.
POPSTORE_PWDREQUIRED	A username and password are required. The command was not executed.
POPSTORE_VMERROR	Insufficient virtual memory. The command was not executed.
	Any error returned by the `output_proc` procedure.

POPSTORE_command_d

Process a popstore management command.

FORMAT

```
int POPSTORE_command_d (command, command_len,
                        password_required,
                        domain, domain_len
                        user, user_len,
                        password, password_len,
                        usage, usage_len,
                        out_info, out_info_len,
                        context, output_proc)
      char  *command;
      int    command_len;
      int    password_required
      char  *domain;
      int    domain_len;
      char  *user;
      int    user_len;
      char  *password;
      int    password_len;
      char  *usage;
      int    usage_len;
      char  *out_info;
      int   *out_info_len;
      void  *context;
      int  (*output_proc)();
```

ARGUMENTS

`command`
The command to process. Used for input only.

`command_len`
Length in bytes of the command specified with `command`. Used for input only.

`password_required`
Boolean flag indicating whether or not password authentication is required in order to process the command. Used for input only.

`domain`
Name of the user domain associated with the user `user`. Supply a value of `NULL` to indicate the `default` domain. Used for input only.

`domain_len`
Length in bytes of the user domain name, `domain`. Supply a value of zero to indicate the `default` domain. Used for input only.

`user`
Name of the popstore user account to authenticate against when password authentication is required. Used for input only.

`user_len`
Length in bytes of the username specified with `user`. Used for input only.

`password`
User-supplied, plain text password to use when performing a password authentication. Used for input only.

`password_len`
Length in bytes of the plain text password specified with `password`. Used for input only.

`usage`
Usage description to be passed to site-supplied logging subroutines explaining the usage. Used for input only.

`usage_len`
Length in bytes of the usage description specified with `usage`. Used for input only.

`out_info`
Output buffer to receive post-processing information. Used for output only.

`out_info_len`
On input, the maximum length in bytes of the buffer pointed at by `out_info`. On output, the length in bytes of the information placed in the `out_info` buffer. Used for input and output.

`context`
Pointer to private client data to be passed to the client-supplied `output_proc` procedure. Used for input only.

`output_proc`
Address of a client-supplied subroutine to receive command processing output. Used for input only.

DESCRIPTION

The management commands documented in Chapter 4 are processed by the POPSTORE_command_d subroutine. Sites wanting to produce their own management interfaces can do so by generating management commands in the URL format described in Chapter 4 and then invoking `POPSTORE_command` to process those commands. The commands should reference site-supplied formatting files which then format the command results in the desired format.[2]

When called, `POPSTORE_command_d` will use the active privileges of the caller to access and manipulate the popstore. The `password_required` argument can be used to implement a utility which itself has privileges but requires, for operation, that the user know the username and password of a popstore account with management privileges. When a non-zero value is supplied for `password_required`, values for the `user`, `user_len`, `password`, and `password_len` arguments must be supplied. The supplied plain text password will be authenticated against the specified account. If the password is correct and the account has the `MANAGE` usage flag set, then the indicated command will be processed. Otherwise, a `POPSTORE_AUTHFAIL` or `POPSTORE_NOMANAGE` error will be returned. Moreover, the

[2] This is precisely how the Web-based and command line oriented management utilities are implemented.

account will only be allowed to perform management functions on other accounts within the same management group.

When user and user_len arguments are supplied, the operation to be undertaken will be restricted to accounts within the same management group and subgroups of user. This behavior is regardless the setting of the password_required argument.

On return from calling POPSTORE_command, the out_info buffer will contain the name of the formatting file associated with either the processed command's on_success or on_error parameter. If the command was successfully processed, then the buffer will contain the value specified with the command's on_success parameter. If the command was not successfully processed, then the value specified by the on_error parameter will be returned. If you do not care to have this information returned, specify a null value for out_info or out_info_len or both.

The output_proc procedure is called repeatedly to received lines of output generated as a result of processing the command. The procedure takes the form

```
int output_proc (context, data, data_len, is_eol, is_literal,
                 is_error)
      void *context;
      char *data;
      int   data_len;
      int   is_eol;
      int   is_literal;
      int   is_error;
```

where the arguments to output_proc are as follows:

context	Pointer to the private client data supplied as input to POPSTORE_command.
data	Data to output. This string can not be null terminated.
data_len	Length in bytes of the data pointed at by data.
is_eol	When is_eol has a non-zero value, the output_proc procedure can want to output an end-of-line after this batch of formatted data.
is_literal	When is_literal has a non-zero value, the output_proc procedure should not apply any quoting to the output data. The output data is literal data which was contained within the formatting file.
is_error	When is_error has a non-zero value, the output data is an error message relating to the processed command.

Upon successful completion, output_proc should return the value POPSTORE_SUCCESS. In the event of an error, some value other than POPSTORE_SUCCESS should be returned. A user-requested abort can be signified by returning POPSTORE_ABORT.

RETURN VALUES

POPSTORE_SUCCESS	Normal, successful completion.
POPSTORE_BADARG	Null value received for the out_proc argument.

POPSTORE_ERRORS	The command was parsed and executed; execution of the command resulted in some errors which were reported via the `output_proc` procedure.
POPSTORE_AUTHFAIL	Invalid password or username supplied. The command was not executed.
POPSTORE_BADHEXVAL	The command contained an illegal hexadecimal encoded character. The command was not executed.
POPSTORE_CMDTOOLONG	The command contained too many name=value pairs. The command was not executed.
POPSTORE_CMDUNKOWN	The command contained an unrecognized right-hand-side in a name=value pair. The command was not executed.
POPSTORE_CMDUNKOWNL	The command contained an unrecognized left-hand-side in a name=value pair. The command was not executed.
POPSTORE_NOMANAGE	Specified account lacks management privileges. The command was not executed.
POPSTORE_PWDREQUIRED	A username and password are required. The command was not executed.
POPSTORE_VMERROR	Insufficient virtual memory. The command was not executed.
	Any error returned by the `output_proc` procedure.
	Any other popstore error.

POPSTORE_end

End usage of the popstore API.

FORMAT

```
int POPSTORE_end (void)
```

ARGUMENTS *None.*

DESCRIPTION

When a program is done using the popstore API subroutines, it should call `POPSTORE_end`. This call allows the popstore API to gracefully shut itself down and deallocate any dynamic memory it can have allocated. Note that this call will not also call PMDF_done. If you also need to show the PMDF API down, then follow the `POPSTORE_end` call with a PMDF_done call.

If a site-supplied logging subroutine is in use, then it will be called by `POPSTORE_end` with a log type of `POPSTORE_LOG_END`.

RETURN VALUES

POPSTORE_SUCCESS	Normal, successful completion.

POPSTORE_error_to_text

Convert a numerical popstore error to a textual error message.

FORMAT

```
char *POPSTORE_error_to_text (error_code)
        int error_code;
```

ARGUMENTS

`error_code`
Numerical popstore error code returned by a popstore API subroutine. Used for input only.

DESCRIPTION

With the exception of `POPSTORE_error_to_text`, all popstore API subroutines return a numeric status code indicating success or failure. A brief text message explaining a given numeric status code can be obtained by calling `POPSTORE_error_to_text`. The return value of `POPSTORE_error_to_text` will be a pointer to a static, null terminated string containing the explanation.

RETURN VALUES *None.*

POPSTORE_format_counters

Format PMDF channel counter information.

FORMAT

```
int POPSTORE_format_counters (format, channel, channel_len,
                              context, output_proc)
      POPSTORE_format_element  *format;
      char                     *channel;
      int                       channel_len;
      void                     *context;
      int                      (*output_proc)();
```

ARGUMENTS

`format`
Pointer to a formatting context returned by a previous call to `POPSTORE_format_read`. Used for input only.

`channel`
Name of the channel to display information for. This name can contain wild card characters. Used for input only.

`channel_len`
Length in bytes of the string passed in `channel`. Used for input only.

`context`
Pointer to private client data to be passed to the client-supplied `output_proc` procedure. Used for input only.

`output_proc`
Address of a client-supplied subroutine to call to output formatted data. Used for input only.

DESCRIPTION

PMDF channel counter information can be formatted with `POPSTORE_format_counters`. The channel name is case insensitive and can contain wild card characters. To format information for all channels, either specify asterisk, “*”, for the channel name or pass a null for `channel` and the value 0 for `channel_len`.

The formatting context should be derived from a formatting file using substitution strings from Tables 4–10 and 4–20.

Formatted data is passed to the output procedure `output_proc`. That procedure takes the form

```
int output_proc (context, data, data_len, is_eol, is_literal)
      void *context;
      char *data;
      int   data_len;
      int   is_eol;
      int   is_literal;
```

where the arguments to `output_proc` are as follows:

`context`	Pointer to the private client data supplied as input to POPSTORE_format_counters.
`data`	Formatted data to output. This string can not be null terminated.
`data_len`	Length in bytes of the data pointed at by `data`.
`is_eol`	When `is_eol` has a non-zero value, the `output_proc` procedure can want to output an end-of-line after this batch of formatted data.
`is_literal`	When `is_literal` has a non-zero value, the `output_proc` procedure should not apply any quoting to the formatted data. The formatted data is literal data which was contained within the formatting file.

Upon successful completion, `output_proc` should return the value `POPSTORE_SUCCESS`. In the event of an error, some value other than `POPSTORE_SUCCESS` should be returned. A user-requested abort can be signified by returning `POPSTORE_ABORT`.

RETURN VALUES

POPSTORE_SUCCESS	Normal, successful completion.
POPSTORE_BADARG	Bad value passed for the address of the output procedure, `output_proc`.
	Any error value returned by the output procedure, `output_proc`.

POPSTORE_format_dispose

Dispose of a formatting context.

FORMAT

```
int POPSTORE_format_dispose (format)
      POPSTORE_format_element *format;
```

ARGUMENTS

`format`
Formatting context to dispose of. Used for input only.

DESCRIPTION

POPSTORE_format_dispose should be called to dispose of a formatting context generated by `POPSTORE_format_read`. `POPSTORE_format_dispose` should be called once for each context generated and no longer needed.

RETURN VALUES

POPSTORE_SUCCESS	Normal, successful completion.

POPSTORE_format_forwarding

Format forwarding information.

FORMAT

```
int POPSTORE_format_forwarding (format, user, user_len,
                                 is_prefix, context,
                                 output_proc)
     POPSTORE_format_element  *format;
     char                     *user;
     int                       user_len;
     int                       is_prefix;
     void                     *context;
     int                     (*output_proc)();
```

ARGUMENTS

`format`
Pointer to a formatting context returned by a previous call to POPSTORE_format_read. Used for input only.

`user`
Name of the user to display the forwarding for. This string can not contain wild cards. The length of this string can not exceed POPSTORE_MAX_USER_LEN bytes. Used for input only.

`user_len`
Length in bytes of the username passed in `user`. Used for input Only.

`is_prefix`
Boolean flag with value 0 or 1 indicating whether or not to treat the specified username as a prefix. Used for input only.

`context`
Pointer to private client data to be passed to the client-supplied `output_proc` procedure. Used for input only.

`output_proc`
Address of a client-supplied subroutine to call to output formatted data. Used for input only.

DESCRIPTION

Although still supported, this subroutine is obsolete. Instead use the more general `POPSTORE_format_forwarding_d` subroutine.

RETURN VALUES

POPSTORE_SUCCESS	Normal, successful completion.
POPSTORE_BADARG	Bad value passed for the address of the output procedure, `output_proc`.

POPSTORE_TOOLONG	Username string is too long; it can not exceed a length of POPSTORE_MAX_USER_LEN bytes.
	Any error value returned by the output procedure, `output_proc`.

POPSTORE_format_forwarding_d

Format forwarding information.

FORMAT

```
int POPSTORE_format_forwarding_d (format, domain, domain_len,
                                  user, user_len, is_prefix,
                                  context, output_proc)
      POPSTORE_format_element  *format;
      char                     *domain;
      int                       domain_len;
      char                     *user;
      int                       user_len;
      int                       is_prefix;
      void                     *context;
      int                     (*output_proc)();
```

ARGUMENTS

`format`
Pointer to a formatting context returned by a previous call to POPSTORE_format_read. Used for input only.

`domain`
Name of the user domain associated with the user `user`. Supply a value of `NULL` to indicate the `default` domain. Used for input only.

`domain_len`
Length in bytes of the user domain name, `domain`. Supply a value of zero to indicate the `default` domain. Used for input only.

`user`
Name of the user to display the forwarding for. This string can not contain wild cards. The length of this string cannot exceed `POPSTORE_MAX_USER_LEN` bytes. Used for input only.

`user_len`
Length in bytes of the username passed in `user`. Used for input Only.

`is_prefix`
Boolean flag with value 0 or 1 indicating whether or not to treat the specified username as a prefix. Used for input only.

`context`
Pointer to private client data to be passed to the client-supplied `output_proc` procedure. Used for input only.

`output_proc`
Address of a client-supplied subroutine to call to output formatted data. Used for input only.

DESCRIPTION

To generate a formatted display of forwarding information, use `POPSTORE_format_forwarding`. The formatting context passed to this subroutine should be derived from a formatting file using substitution strings from Tables 4–10 and 4–11.

The supplied username is treated in a case-insensitive fashion and can not contain wild cards. An exact match will be done if `is_prefix` has the value 0. If `is_prefix` has a non-zero value, then the value of `user` will be treated as a prefix and all forwardings for usernames beginning with that value will be returned. For instance, if `user` has the value "D" and `is_prefix` is non-zero, then all forwardings for usernames beginning with the letter "D" will be returned. To return all of the forwardings, supply an empty string for `user`, the value 0 for `user_len`, and a non-zero value for `is_prefix`.

See the description of the `POPSTORE_format_counters` subroutine for a description of the `output_proc` procedure.

RETURN VALUES

POPSTORE_SUCCESS	Normal, successful completion.
POPSTORE_BADARG	Bad value passed for the address of the output procedure, `output_proc`.
POPSTORE_TOOLONG	Username string is too long; it can not exceed a length of POPSTORE_MAX_USER_LEN bytes.
	Any error value returned by the output procedure, `output_proc`.

POPSTORE_format_message

Format a stored message.

FORMAT

```
int POPSTORE_format_message (format, fspec, fpsec_len,
                             context, output_proc)
      POPSTORE_format_element  *format;
      char                     *fspec;
      int                       fspec_len;
      void                     *context;
      int                     (*output_proc)();
```

ARGUMENTS

`format`
Pointer to a formatting context returned by a previous call to `POPSTORE_format_read`. Used for input only.

`fspec`
Message file specification for the message file to display. Used for input only.

`fspec_len`
Length in bytes of the message file specification. Used for input only.

`context`
Pointer to private client data to be passed to the client-supplied `output_proc` procedure. Used for input only.

`output_proc`
Address of a client-supplied subroutine to call to output formatted data. Used for input only.

DESCRIPTION

Stored messages can be formatted for display with `POPSTORE_format_message`. The formatting context passed to this subroutine should be derived from a formatting file using substitution strings from Tables 4–10 and 4–22.

Whereas the `POPSTORE_message_` subroutines all require message indices such as the value 9 to reference the ninth message, `POPSTORE_format_message` requires a message file name to reference a message file. This is done because `POPSTORE_format_message` is typically used in situations where a user context is no longer available.

The message to display is indicated by the `fspec` argument. The value of that argument should be a message file name without directory path information. This file name can be derived from a user's message list: the nth message in a user's list of messages has the file name

```
user_context->messages[n-1].filename
```

where user_context is a pointer to a POPSTORE_user_context returned by the POPSTORE_user_begin_d subroutine.

See the description of the POPSTORE_format_counters subroutine for a description of the output_proc procedure.

RETURN VALUES

POPSTORE_SUCCESS	Normal, successful completion.
POPSTORE_BADARG	Bad value passed for the address of the output procedure, output_proc, or the message file specification, fspec.
POPSTORE_BADLENGTH	Bad length passed for the value of fspec_len.
POPSTORE_FILOPNERR	An error occurred while trying to open the message file.
POPSTORE_READERROR	An error occurred while trying to read the message file.
POPSTORE_SEEKERROR	An error occurred while seeking in the message file.
POPSTORE_TOOLONG	Username or file specification is too long. The username string can not exceed a length of POPSTORE_MAX_USER_LEN bytes; the file specification can not exceed a length of 1024 bytes.
POPSTORE_VMERROR	Insufficient virtual memory.
	Any error value returned by the output procedure, output_proc.

POPSTORE_format_messages

Format a user's list of stored messages.

FORMAT

```
int POPSTORE_format_messages (format, user_context, context,
                              output_proc)
     POPSTORE_format_element  *format;
     POPSTORE_user_context    *user_context;
     void                     *context;
     int                     (*output_proc)();
```

ARGUMENTS

format
Pointer to a formatting context returned by a previous call to POPSTORE_format_read. Used for input only.

user_context
User context for the user to display the messages for. Obtain this context by calling POPSTORE_user_begin_d. Used for input only.

context
Pointer to private client data to be passed to the client-supplied output_proc procedure. Used for input only.

output_proc
Address of a client-supplied subroutine to call to output formatted data. Used for input only.

DESCRIPTION

POPSTORE_format_messages can be used to format a user's list of stored messages. The formatting context passed to this subroutine should be derived from a formatting file using substitution strings from Tables 4–10 and 4–24.

See the description of the POPSTORE_format_counters subroutine for a description of the output_proc procedure.

RETURN VALUES

POPSTORE_SUCCESS	Normal, successful completion.
POPSTORE_BADARG	Bad value passed for the address of the output procedure, output_proc.
	Any error value returned by the output procedure, output_proc.

POPSTORE_format_profile

Format for display a user profile.

FORMAT

```
int POPSTORE_format_profile (format, user_context, flags,
                             context, output_proc)
     POPSTORE_format_element  *format;
     POPSTORE_user_context    *user_context;
     int                       flags;
     void                     *context;
     int                     (*output_proc)();
```

ARGUMENTS

`format`
Pointer to a formatting context returned by a previous call to `POPSTORE_format_read`. Used for input only.

`user_context`
User context to display. Obtain this context by calling `POPSTORE_user_begin_d`. Used for input only.

`flags`
Bit mask. Used for input only.

`context`
Pointer to private client data to be passed to the client-supplied `output_proc` procedure. Used for input only.

`output_proc`
Address of a client-supplied subroutine to call to output formatted data. Used for input only.

DESCRIPTION

POPSTORE_format_profile can be used to format for display a user profile. The formatting context passed to this subroutine should be derived from a formatting file using substitution strings from Tables 4–10 and 4–15.

The `flags` argument is used to control the handling of the `%first`, `%last`, `%!first`, and `%!last` substitution strings. When the lowest bit of `flags` is set, `%first` is true. When the second lowest bit is set, `%last` is true.

See the description of the `POPSTORE_format_counters` subroutine for a description of the `output_proc` procedure.

RETURN VALUES

POPSTORE_SUCCESS	Normal, successful completion.
POPSTORE_BADARG	Bad value passed for the address of the output procedure, `output_proc`.
	Any error value returned by the output procedure, `output_proc`.

POPSTORE_format_profiles

Obsolete subroutine: use the more general POPSTORE_format_profiles_d subroutine.

FORMAT

```
int POPSTORE_format_profiles (format, user, user_len,
                              context, output_proc)
      POPSTORE_format_element  *format;
      char                     *user;
      int                       user_len;
      void                     *context;
      int                     (*output_proc)();
```

ARGUMENTS

`format`
Pointer to a formatting context returned by a previous call to POPSTORE_format_read. Used for input only.

`user`
Name of the user to display profile information for. This string can contain wild cards. Used for input only.

`user_len`
Length in bytes of the username string, `user`. Length of this string cannot exceed `ALFA_SIZE` bytes. Used for input only.

`context`
Pointer to private client data to be passed to the client-supplied output_proc procedure. Used for input only.

`output_proc`
Address of a client-supplied subroutine to call to output formatted data. Used for input only.

DESCRIPTION

While use of this subroutine is still supported, it is now obsolete. Use the more general `POPSTORE_format_profiles_d` subroutine instead.

RETURN VALUES

POPSTORE_SUCCESS	Normal, successful completion.
POPSTORE_BADARG	Bad value passed for the address of the output procedure, `output_proc`.
POPSTORE_TOOLONG	Length of username string exceeds ALFA_SIZE bytes.
	Any error value returned by the output procedure, `output_proc`.

POPSTORE_format_profiles_d

Format for display a list of user profiles.

FORMAT

```
int POPSTORE_format_profiles_d (format, domain, domain_len,
                                group, group_len, user,
                                user_len, context,
                                output_proc)
     POPSTORE_format_element  *format;
     char                     *domain;
     int                       domain_len;
     char                     *group;
     int                       group_len;
     char                     *user;
     int                       user_len;
     void                     *context;
     int                     (*output_proc)();
```

ARGUMENTS

format
Pointer to a formatting context returned by a previous call to POPSTORE_format_read. Used for input only.

domain
Name of the user domain to use. Supply a value of NULL to indicate the default domain. Used for input only.

domain_len
Length in bytes of the user domain name, domain. Supply a value of zero to indicate the default domain. Used for input only.

group
Name of the management group and subgroups thereof to restrict the listing to. Supply a value of NULL if accounts from any group should be listed. This string cannot contain wild cards. Used for input only.

group_len
Length in bytes of the group name, group. Specify a value of zero if you do not want to restrict the listing to a particular group.

user
Name of the user to display profile information for. This string can contain wild cards. Used for input only.

user_len
Length in bytes of the username string, user. Length of this string can not exceed ALFA_SIZE bytes. Used for input only.

context
Pointer to private client data to be passed to the client-supplied output_proc procedure. Used for input only.

`output_proc`
Address of a client-supplied subroutine to call to output formatted data. Used for input only.

DESCRIPTION

Use POPSTORE_format_profiles_d to generate a listing of popstore user accounts. When `group_len` is zero, all accounts matching the username specification, `user`, will be formatted for display. When `group_len` is larger than zero, only those accounts within the specified management group and matching the username specification will be formatted for display. The username specification is treated as a case-insensitive string and can contain wild card characters. To list all accounts you can simply pass a null value for `user` or a zero value for `user_len` or both.

The formatting context passed to this subroutine should be derived from a formatting file using substitution strings from Tables 4–10 and 4–15.

See the description of the `POPSTORE_format_counters` subroutine for a description of the `output_proc` procedure.

RETURN VALUES

POPSTORE_SUCCESS	Normal, successful completion.
POPSTORE_BADARG	Bad value passed for the address of the output procedure, `output_proc`.
POPSTORE_TOOLONG	Length of username string exceeds ALFA_SIZE bytes.
	Any error value returned by the output procedure, `output_proc`.

POPSTORE_format_read

Read and parse a formatting file.

FORMAT

```
int POPSTORE_format_read (format, fname, fname_len, path,
                          path_len)
     POPSTORE_format_element **format;
     char                     *fname;
     int                       fname_len;
     char                     *path;
     int                       path_len;
```

ARGUMENTS

`format`
Pointer to a formatting context generated by reading and parsing the specified formatting template file. Dispose of the context by calling POPSTORE_format_dispose. Used for output only.

`fname`
Name of the formatting template file to read. Used for input only.

`fname_len`
Length in bytes of the formatting template file, `fspec`. Used for input only.

`path`
Full directory path to the directory containing the formatting template file. Used for input only.

`path_len`
Length in bytes of the directory path, `path`. Used for input only.

DESCRIPTION

Before calling any of the other `POPSTORE_format_` subroutines, you must first have processed a formatting template file with `POPSTORE_format_read`. This will generate a formatting context which can then be used with the other `POPSTORE_format_` subroutines. When finished using a formatting context, dispose of it with a call to `POPSTORE_format_dispose`.

The name of the formatting file is specified with the `fname` argument. The directory path leading to the file is specified with the `path` argument.[3]

Formatting files are described throughout Chapter 4; see Section 4.3.2 for basic information on formatting files. The particular substitution strings permitted in a formatting file will depend upon the intended usage of the formatting file; *i.e.*, depends upon which `POPSTORE_format_` subroutines it will subsequently be used with.

[3] These two parts of the full file path are specified independently so as to simplify the coding of servers which need to ensure that formatting files are read only from a specific directory tree and not from any user-specified location.

RETURN VALUES

POPSTORE_SUCCESS	Normal, successful completion.
POPSTORE_BADARG	File name argument, `fname`, is null, or file name length, `fname_len`, is zero.
POPSTORE_ILLFILE	Directory path and file name result in an illegal file specification; note that the directory path must be specified and must have a non-zero length.
POPSTORE_TOOLONG	Directory path and file name result in a file specification whose length exceeds 1024 bytes.
POPSTORE_FILOPNERR	Unable to open the formatting file.
POPSTORE_VMERROR	Insufficient virtual memory.

POPSTORE_init

Initialize the popstore API.

FORMAT

```
int POPSTORE_init (init_pmdf, sleep_proc, usage, usage_len)
      int    init_pmdf;
      void (*sleep_proc)();
      char  *usage;
      int    usage_len;
```

ARGUMENTS

`init_pmdf`
Boolean flag, 0 or 1, indicating whether or not to also call PMDF_initialize. Used for input only.

`sleep_proc`
Address of a procedure to call to sleep the process or thread. Used for input only.

`usage`
A short description identifying the usage being made by the popstore API client. Used for input only.

`usage_len`
Length in bytes of the usage description, `usage`. Used for input only.

DESCRIPTION

To use the popstore API, you must first initialize the API by calling `POPSTORE_init`. When through using the popstore API, call `POPSTORE_end`. When a call to `POPSTORE_init` fails, `POPSTORE_end` does not need to be called. However, `POPSTORE_end` can be called even when a call to `POPSTORE_init` has failed.

If PMDF has not already been initialized via the PMDF API subroutine PMDF_initialize, then you must specify a value of 1 for `init_pmdf`. Otherwise, specify a value of 0.

A brief description of the intended usage can be specified with the `usage` argument. This information will be passed to any site-supplied logging procedure as part of the `POPSTORE_LOG_START` logging call.

When the popstore API is used by a multi-threaded process, the address of a procedure to sleep a single thread should be supplied with the `sleep_proc` argument. The procedure takes the form

```
void sleep (centi_seconds)
       unsigned long centi_seconds;
```

where `centi_seconds` is the number of hundredths of seconds to sleep the thread for. When a null is passed for the value of `sleep_proc`, the popstore API will use a default sleep procedure.[4]

RETURN VALUES

POPSTORE_SUCCESS	Normal, successful completion.
POPSTORE_BADARG	Illegal value specified for the USERNAME_STYLE or USERNAME_CHARSET options; initialization failed.
POPSTORE_LOADERROR	Unable to load a site-supplied callable image; initialization failed.
POPSTORE_PMDFINITERR	Unable to initialize PMDF; initialization failed.
POPSTORE_READERROR	Unable to read the popstore option file; initialization failed.

[4] Presently, the sleep procedure is only used in the event that a file cannot be opened or created for some unknown reason. In such situations, the popstore will sleep for a brief period of time and attempt to open or create the file a second time.

POPSTORE_manage

Allow changing of the MANAGE usage flag for accounts.

FORMAT

```
int POPSTORE_manage (allow)
      int allow;
```

ARGUMENTS

allow
When non-zero, the MANAGE usage flag can be changed. Used for input only.

DESCRIPTION

By default, the popstore API subroutines do not allow manipulation of the MANAGE usage flag for popstore accounts. Specifically, the POPSTORE_user_data_set, POPSTORE_user_update, POPSTORE_command, and POPSTORE_command_d subroutines will not, by default, allow alterations to be made to that flag for any account. In order to enable the setting or clearing of that flag, POPSTORE_manage must be called with a non-zero value for the allow argument. To subsequently disable the ability to set or clear the MANAGE flag, call POPSTORE_manage with a zero value for allow. Note that even after POPSTORE_manage has been called, a program cannot alter the MANAGE flag, or any other aspect of a popstore account, unless it has sufficient privileges to read and write profile files.

Note that POPSTORE_manage does not return a POPSTORE_ status code. Instead, it returns a value indicating the state prior to calling POPSTORE_manage. If a value of 0 is returned, then previously the MANAGE flag could not be manipulated. If a value of 1 is returned, then previously the MANAGE flag could be manipulated. Thus, a subroutine which needs to briefly impose one state or another can make that imposition and then restore the prior state:

```
/* Change the current state */
old_state = POPSTORE_manage (new_state);

...

/* Restore the old state */
(void) POPSTORE_manage (old_state);
```

RETURN VALUES

0 or 1	The previous state prior to calling POPSTORE_manage.

POPSTORE_message_begin

Access a stored message for reading.

FORMAT

```
int POPSTORE_message_begin (user_context, message_index,
                            message_context, env_id,
                            env_id_len msg_id, msg_id_len)
      POPSTORE_user_context *user_context;
      int                    message_index;
      int                   *message_context;
      char                  *env_id;
      int                   *env_id_len;
      char                  *msg_id;
      int                   *msg_id_len;
```

ARGUMENTS

`user_context`
User context created by a previous call to `POPSTORE_user_begin_d`. Used for input only.

`message_index`
Index of the message to access. Used for input only.

`message_context`
Message context established by this call. Dispose of the context by calling `POPSTORE_message_end`. Used for output only.

`env_id`
Pointer to a string in which to return the value of the message's envelope identification field. Used for output only.

`env_id_len`
On input, the maximum length in bytes of the buffer to receive the message's envelope identification field, `env_id`. On output, the length in bytes of the returned envelope identification field. Used for input and output.

`msg_id`
Pointer to a string in which to return the value of the message's message identification field. Used for output only.

`msg_id_len`
On input, the maximum length in bytes of the buffer to receive the message identification field, `msg_id`. On output, the length in bytes of the returned message identification field. Used for input and output.

DESCRIPTION

To access a user's nth message for purposes of reading it, call POPSTORE_message_begin with `message_index` set to the value `n`. (The first message has index 1, the second index 2, and so on.) The message context returned can then be used with `POPSTORE_message_read` to read the message. When done reading the message, call `POPSTORE_message_end` to close the message file and end the context.

Optionally, the values of the message's envelope and message identification fields can be returned. To receive the value for the envelope identification, supply with `env_id` the address of a buffer to receive the value. On input, `env_id_len` should point to an integer whose value is the maximum size in bytes of that buffer. On output, the value of that integer will be changed to be the length in bytes of the returned envelope identification field. The same holds for the message identification field. To be ensured that that entire field values are returned and not truncated, the buffers should have lengths of at least `ALAFA_SIZE` bytes plus an additional byte to store a null terminator.

If you are not interested in obtaining the envelope identification, you can pass a null value for `env_id`. Likewise for the message identification. Note that if site-supplied logging subroutines are in use, the envelope identification for the message will not be logged when you supply a null value for `env_id`. Again, likewise for the message identification.

An example of using `POPSTORE_message_begin` is given in Example 12–8.

RETURN VALUES

POPSTORE_SUCCESS	Normal, successful completion.
POPSTORE_BADMSG	Underlying message file is corrupted or otherwise unreadable; message marked for deletion.
POPSTORE_BADMSGID	Message index is less than 1 or greater than `user_context->message_count`.
POPSTORE_DELETED	Message has been marked for deletion.
POPSTORE_FILOPNERR	Unable to open the underlying message file.
POPSTORE_NULLCONTEXT	`user_context` context is null.
POPSTORE_NULLMSGLIST	`user_context->messages` is null.
POPSTORE_NULLPROFILE	`user_context->profile` context is null.
POPSTORE_TOOLONG	File specification for the underlying message file is too long.

POPSTORE_message_end

Deaccess a stored message.

FORMAT

```
int POPSTORE_message_end (message_context);
      int message_context;
```

ARGUMENTS

`message_context`
Message context returned by a previous call to `POPSTORE_message_begin`. Used for input only.

DESCRIPTION

When finished reading a message, call `POPSTORE_message_end` to close the underlying message file and finish the message context. Note that it is not necessary to call any of the `POPSTORE_message_mark_` subroutines before `POPSTORE_message_end` as they can be called at any time.

RETURN VALUES

POPSTORE_SUCCESS	Normal, successful completion.
POPSTORE_FILCLSERR	Error occurred while closing the underlying message file.

POPSTORE_message_mark_delete

Mark a user's message copy for deletion.

FORMAT

```
int POPSTORE_message_mark_delete (user_context,
                                  message_index)
      POPSTORE_user_context *user_context;
      int                    message_index;
```

ARGUMENTS

`user_context`
User context created by a prior call to `POPSTORE_user_begin_d`. Used for input only.

`message_index`
Index of the message to mark for deletion. Used for input only.

DESCRIPTION

To mark a user's stored message for deletion, use `POPSTORE_message_mark_delete`. The index of the message to mark for deletion is specified with `message_index`. The first message has index 1, the second index 2, and so on.

Note that the message is not immediately deleted. It is retained until the user context is shut down with `POPSTORE_user_end`.[5] Note further that the underlying message file will only be deleted when all popstore recipients of the message have deleted the message.

RETURN VALUES

POPSTORE_SUCCESS	Normal, successful completion.
POPSTORE_BADMSGID	Message index is less than 1 or greater than `user_context->message_count`.
POPSTORE_NULLCONTEXT	`user_context` context is null.
POPSTORE_NULLMSGLIST	`user_context->messages` is null.
POPSTORE_NULLPROFILE	`user_context->profile` context is null.

[5] Such deferrals are necessary in order to implement the POP `RSET` command.

POPSTORE_message_mark_nodelete

Mark a user’s message to be retained.

FORMAT

```
int POPSTORE_message_mark_nodelete (user_context,
                                    message_index)
      POPSTORE_user_context *user_context;
      int                    message_index;
```

ARGUMENTS

user_context
User context created by a prior call to POPSTORE_user_begin_d. Used for input only.

message_index
Index of the message to mark to keep and not delete. Used for input only.

DESCRIPTION

To mark a user’s stored message to be retained and not deleted, call POPSTORE_message_mark_nodelete. The index of the message to retain is specified with message_index. The first message has index 1, the second index 2, and so on.

RETURN VALUES

POPSTORE_SUCCESS	Normal, successful completion.
POPSTORE_BADMSGID	Message index is less than 1 or greater than user_context->message_count.
POPSTORE_NULLCONTEXT	user_context context is null.
POPSTORE_NULLMSGLIST	user_context->messages is null.
POPSTORE_NULLPROFILE	user_context->profile context is null.

POPSTORE_message_mark_noread

Mark a user's message as being unread.

FORMAT

```
int POPSTORE_message_mark_noread (user_context,
                                  message_index)
      POPSTORE_user_context *user_context;
      int                    message_index;
```

ARGUMENTS

`user_context`
User context created by a prior call to `POPSTORE_user_begin_d`. Used for input only.

`message_index`
Index of the message to mark as unread. Used for input only.

DESCRIPTION

Use `POPSTORE_message_mark_noread` to mark a user's stored message as unread. The index of the message to mark is specified with `message_index`. The first message has index 1, the second index 2, and so on.

RETURN VALUES

POPSTORE_SUCCESS	Normal, successful completion.
POPSTORE_BADMSGID	Message index is less than 1 or greater than `user_context->message_count`.
POPSTORE_NULLCONTEXT	`user_context` context is null.
POPSTORE_NULLMSGLIST	`user_context->messages` is null.
POPSTORE_NULLPROFILE	`user_context->profile` context is null.

POPSTORE_message_mark_read

Mark a user's message as read.

FORMAT

```
int POPSTORE_message_mark_read (user_context, message_index)
      POPSTORE_user_context *user_context;
      int                    message_index;
```

ARGUMENTS

`user_context`
User context created by a prior call to `POPSTORE_user_begin_d`. Used for input only.

`message_index`
Index of the message to mark as read. Used for input only.

DESCRIPTION

Use `POPSTORE_message_mark_read` to mark a user's stored message as being read. The index of the message to mark is specified with `message_index`. The first message has index 1, the second index 2, and so on.

An example of using `POPSTORE_message_mark_read` is given in Example 12–8.

RETURN VALUES

POPSTORE_SUCCESS	Normal, successful completion.
POPSTORE_BADMSGID	Message index is less than 1 or greater than `user_context->message_count`.
POPSTORE_NULLCONTEXT	`user_context` context is null.
POPSTORE_NULLMSGLIST	`user_context->messages` is null.
POPSTORE_NULLPROFILE	`user_context->profile` context is null.

POPSTORE_message_read

Sequentially read a message from disk.

FORMAT

```
int POPSTORE_message_read (message_context, buffer,
                           buffer_len, bytes_read)
      int   message_context;
      char *buffer;
      int   buffer_len;
      int  *bytes_read;
```

ARGUMENTS

`message_context`
Message context returned by a previous call to `POPSTORE_message_begin`. Used for input only.

`buffer`
Pointer to a buffer into which to read message data. Used for output only.

`buffer_len`
Maximum length in bytes of the buffer. Used for input only.

`bytes_read`
On output, the number of bytes read and stored in the buffer. Used for output only.

DESCRIPTION

A message accessed with a call to `POPSTORE_message_begin` can be sequentially read with `POPSTORE_message_read`. `POPSTORE_message_read` will read up to `buflen-1` bytes of data, storing them in the buffer pointed at by `buffer` and terminating the data with a null. `POPSTORE_message_read` should be repeatedly called until either POPSTORE_EOM or an error is returned. Note that when `POPSTORE_EOM` is returned, data can also have been returned as indicated by a non-zero value for `bytes_read`.

Note that the data returned by `POPSTORE_message_read` will have embedded CRLF pairs marking the end of message records. Moreover, the data will be "dot stuffed" as per the POP protocol.

An example of using `POPSTORE_message_read` is given in Example 12–8.

RETURN VALUES

POPSTORE_SUCCESS	Normal, successful completion.
POPSTORE_EOM	End of message reached; normal, successful completion.
POPSTORE_READERROR	Error reading message file.

POPSTORE_message_return

Return a message to its originator.

FORMAT

```
int POPSTORE_message_return (user_context, message_index,
                             reason, reason_len)
     POPSTORE_user_context *user_context;
     int                    message_index;
     char                  *reason;
     int                    reason_len;
```

ARGUMENTS

`user_context`
User context returned by a previous call to `POPSTORE_user_begin_d`. Used for input only.

`message_index`
Index of the message to return. Used for input only.

`reason`
Pointer to a string containing a brief explanation explaining why the message is being returned. Used for input only.

`reason_len`
Length in bytes of the string pointed at by `reason`. Used for input only.

DESCRIPTION

Call `POPSTORE_message_return` to return a user's message to its originator. The reason the message is being returned can be indicated with the `reason` argument. If a null value is specified for `reason` or a zero length for `reason_len`, then the reason "Unable to deliver: recipient has not downloaded this message after X days" will be given.

The index of the message to return is specified with `message_index` where the first message has index value 1, the second index value 2, and so on.

RETURN VALUES

POPSTORE_SUCCESS	Normal, successful completion.
POPSTORE_BADMSG	Underlying message file is corrupted or otherwise unreadable.
POPSTORE_BADMSGID	Message index is less than 1 or greater than `user_context->message_count`.
POPSTORE_DELETED	Message has been marked for deletion.
POPSTORE_FILOPNERR	Unable to open the underlying message file.
POPSTORE_INSUFPRIV	The process lacks SYSLCK privilege (OpenVMS only).
POPSTORE_LCKOPNERR	Unable to obtain lock information for the profile file.

POPSTORE_LOCKERROR	Unable to obtain a lock for the profile file.
POPSTORE_NULLCONTEXT	`user_context` context is null.
POPSTORE_NULLMSGLIST	`user_context->messages` is null.
POPSTORE_NULLPROFILE	`user_context->profile` context is null.
POPSTORE_TOOLONG	File specification for the underlying message file is too long.

POPSTORE_user_begin

Obsolete subroutine: use the more general POPSTORE_user_begin_d subroutine.

FORMAT

```
int POPSTORE_user_begin (user_context, user, user_len,
                         do_accounting, usage, usage_len)
      POPSTORE_user_context **user_context;
      char                   *user;
      int                     user_len;
      int                     do_accounting;
      char                   *usage;
      int                     usage_len;
```

ARGUMENTS

user_context
User context returned by a successful call to this subroutine. Used for output only.

user
Name of the popstore user to obtain a user context for. Used for input only.

user_len
Length in bytes of the username string, user. Used for input only.

do_accounting
Boolean flag indicating whether or not to update accounting information for this user when the user context is disposed of. Used for input only.

usage
A short description identifying the usage being made by the caller. This description is passed to site-supplied logging subroutines for logging purposes. Used for input only.

usage_len
Length in bytes of the usage description, usage. Used for input only.

DESCRIPTION

While this subroutine is still supported, it is now obsolete. Use the more general POPSTORE_user_begin_d subroutine.

RETURN VALUES

POPSTORE_SUCCESS	Normal, successful completion.
POPSTORE_FILOPNERR	Unable to open the profile file for reading.
POPSTORE_INSUFPRIV	Insufficient privileges to access the profile file, or, on OpenVMS, the process lacks SYSLCK privilege.
POPSTORE_LCKOPNERR	Unable to obtain lock information for the profile file.
POPSTORE_LOCKERROR	Unable to obtain a lock for the profile file.

POPSTORE_NOSUCHUSR	No such user account.
POPSTORE_READERROR	An error while attempting to read data from the profile file.
POPSTORE_TOOLONG	The resulting profile file name is too long.
POPSTORE_VMERROR	Insufficient virtual memory.

POPSTORE_user_begin_d

Access a user account.

FORMAT

```
int POPSTORE_user_begin_d (user_context, domain, domain_len,
                           user, user_len, do_accounting,
                           usage, usage_len)
      POPSTORE_user_context **user_context;
      char                   *domain;
      int                     domain_len;
      char                   *user;
      int                     user_len;
      int                     do_accounting;
      char                   *usage;
      int                     usage_len;
```

ARGUMENTS

`domain`
Name of the user domain to use. Supply a value of NULL to indicate the `default` domain. Used for input only.

`domain_len`
Length in bytes of the user domain name, `domain`. Supply a value of zero to indicate the `default` domain. Used for input only.

`user_context`
User context returned by a successful call to this subroutine. Used for output only.

`user`
Name of the popstore user to obtain a user context for. Used for input only.

`user_len`
Length in bytes of the username string, `user`. Used for input only.

`do_accounting`
Boolean flag indicating whether or not to update accounting information for this user when the user context is disposed of. Used for input only.

`usage`
A short description identifying the usage being made by the caller. This description is passed to site-supplied logging subroutines for logging purposes. Used for input only.

`usage_len`
Length in bytes of the usage description, `usage`. Used for input only.

DESCRIPTION

User accounts are accessed with the POPSTORE_user_begin_d subroutine. On input, provide the username associated with the account you are interested in and an optional usage string describing the purpose for the access. On output, a pointer to a POPSTORE_user_context structure is returned. That structure will contain the profile information for the account as well as the list of stored messages for the account. See Section 12.11.3 for a description of that structure. When you are finished using the context, dispose of it with a call to POPSTORE_user_end. This call is important as it is during the call that accounting information is recorded and any manipulations of messages actually carried out (*e.g.,* stored messages deleted).

Fields in the returned POPSTORE_user_context must not be changed. Fields which you want to be changed should be changed by calling the appropriate API subroutines. For instance, POPSTORE_user_update to change profile fields, the POPSTORE_message_mark_ subroutines to change the disposition of a stored message, *etc.*

When accessing an account, you must indicate whether or not accounting information should be recorded for the access. This indication is expressed with the do_accounting argument. Specify for that argument a value of POPSTORE_ACCOUNTING if accounting information should be recorded; otherwise, specify a value of POPSTORE_NOACCOUNTING. Accounting should only be recorded when the access is being done on the behalf of the actual user (*i.e.,* is a billable event). For example, a POP server when accessing the account for a remote user would request accounting. A program doing account management would not. Accounting operations entail recording the time of attempted access and deaccess and updating the total connect time.

The optional usage description, specified with the usage and usage_len arguments, are passed to any site-supplied logging subroutines. The popstore does not itself make any use of those strings.

RETURN VALUES

POPSTORE_SUCCESS	Normal, successful completion.
POPSTORE_FILOPNERR	Unable to open the profile file for reading.
POPSTORE_INSUFPRIV	Insufficient privileges to access the profile file, or, on OpenVMS, the process lacks SYSLCK privilege.
POPSTORE_LCKOPNERR	Unable to obtain lock information for the profile file.
POPSTORE_LOCKERROR	Unable to obtain a lock for the profile file.
POPSTORE_NOSUCHUSR	No such user account.
POPSTORE_READERROR	An error while attempting to read data from the profile file.
POPSTORE_TOOLONG	The resulting profile file name is too long.
POPSTORE_VMERROR	Insufficient virtual memory.

POPSTORE_user_billing

Obsolete subroutine: use the more general POPSTORE_user_billing_d subroutine.

FORMAT

```
int POPSTORE_user_billing (user, user_len, last_billing,
                            user_data)
      char                *user;
      int                  user_len;
      time_t               last_billing;
      POPSTORE_user_data  *user_data;
```

ARGUMENTS

`user`
Name of the user to perform billing operations for. Used for input only.

`user_len`
Length in bytes of the username string, `user`. Used for input only.

`last_billing`
End time to use for this billing cycle. Used for input only.

`user_data`
User profile information containing the generated billing information. Used for output only.

DESCRIPTION

While still supported, this subroutine is now obsolete. Instead use the more general `POPSTORE_user_billing_d` subroutine.

RETURN VALUES

POPSTORE_SUCCESS	Normal, successful completion.
POPSTORE_FILOPNERR	Unable to open the profile file for reading.
POPSTORE_INSUFPRIV	Insufficient privileges to access the profile file, or, on OpenVMS, the process lacks SYSLCK privilege.
POPSTORE_LCKOPNERR	Unable to obtain lock information for the profile file.
POPSTORE_LOCKERROR	Unable to obtain a lock for the profile file.
POPSTORE_NOSUCHUSR	No such user account.
POPSTORE_READERROR	An error while attempting to read data from the profile file.
POPSTORE_TOOLONG	The resulting profile file name is too long.
POPSTORE_VMERROR	Insufficient virtual memory.

POPSTORE_user_billing_d

Perform billing operations on an account.

FORMAT

```
int POPSTORE_user_billing_d (domain, domain_len, user,
                             user_len, last_billing,
                             user_data)
      char               *domain;
      int                 domain_len;
      char               *user;
      int                 user_len;
      time_t              last_billing;
      POPSTORE_user_data *user_data;
```

ARGUMENTS

`domain`
Name of the user domain to use. Supply a value of NULL to indicate the `default` domain. Used for input only.

`domain_len`
Length in bytes of the user domain name, `domain`. Supply a value of zero to indicate the `default` domain. Used for input only.

`user`
Name of the user to perform billing operations for. Used for input only.

`user_len`
Length in bytes of the username string, `user`. Used for input only.

`last_billing`
End time to use for this billing cycle. Used for input only.

`user_data`
User profile information containing the generated billing information. Used for output only.

DESCRIPTION

Programs wanting to perform billing operations should use the `POPSTORE_user_billing_d` subroutine in conjunction with the `POPSTORE_user_list_d` subroutine. That latter subroutine provides a convenient way to obtain the name of each popstore account and, with the name in hand, invoke `POPSTORE_user_billing_d` to perform billing operations for that account.

The `POPSTORE_user_billing_d` subroutine performs “atomic” billing operations. Specifically, `POPSTORE_user_billing_d` does the following:

1. Locks the user profile file.
2. Reads the profile file into memory and copies it to the structure pointed at with the `user_data` argument.

3. Computes the accumulated block days used for the currently stored message and copies this information to `user_data->past_block_days`.
4. Clears the total connect and past block days profile fields, and sets the last billing profile and message list fields to the value of the `last_billing` argument.
5. Writes the updated user profile to disk.
6. Unlocks the profile file.
7. Returns the profile data including the total connect time and computed block days in the structure pointed at by the `user_data` argument.

In the returned profile data,

- The value of the last billing field, `user_data->last_billing`, will be the time of the prior billing, not this current billing.
- The value of the total connect time field, `user_data->total_connect`, will be the total connect time accumulated between the times `user_data->last_billing` and `last_billing`.
- The value of the past block days, `user_data->past_block_days`, will be that accumulated between the times `user_data->last_billing` and `last_billing`. This includes the storage used for any messages deleted or returned during that time as well as the storage used for any messages presently being stored.

Note that the roundoff associated with computing the past block days is kept in the user profile file. The returned profile data has the roundoff field set to zero.

RETURN VALUES

POPSTORE_SUCCESS	Normal, successful completion.
POPSTORE_FILOPNERR	Unable to open the profile file for reading.
POPSTORE_INSUFPRIV	Insufficient privileges to access the profile file, or, on OpenVMS, the process lacks SYSLCK privilege.
POPSTORE_LCKOPNERR	Unable to obtain lock information for the profile file.
POPSTORE_LOCKERROR	Unable to obtain a lock for the profile file.
POPSTORE_NOSUCHUSR	No such user account.
POPSTORE_READERROR	An error while attempting to read data from the profile file.
POPSTORE_TOOLONG	The resulting profile file name is too long.
POPSTORE_VMERROR	Insufficient virtual memory.

POPSTORE_user_copy

Obsolete subroutine: use the more general POPSTORE_user_copy_d subroutine.

FORMAT

```
int POPSTORE_user_copy (old_user, old_user_len, new_user,
                        new_user_len, new_password,
                        new_password_len, do_rename)
      char *old_user;
      int   old_user_len;
      char *new_user;
      int   new_user_len;
      char *new_password;
      int   new_password_len;
      int   do_rename;
```

ARGUMENTS

`old_user`
Name of the popstore user to copy or rename. Used for input only.

`old_user_len`
Length in bytes of the string, `old_user`. Used for input only.

`new_user`
Name of the new popstore user to copy or rename the `old_user` account to. Used for input only.

`new_user_len`
Length in bytes of the string, `new_user`. Used for input only.

`new_password`
Optional argument specifying a new, plain text password to use for the new or renamed user profile. Used for input only.

`new_password_len`
Length in bytes of the new, plain text password string, `new_password`. Used for input only.

`do_rename`
Boolean flag indicating whether to do a rename or a copy operation. Used for input only.

DESCRIPTION

While this subroutine is still supported, it is now obsolete. Use the more general `POPSTORE_user_copy_d` subroutine.

RETURN VALUES

POPSTORE_SUCCESS	Normal, successful completion.
POPSTORE_BADLENGTH	The value of `new_user_len` is less than 1.
POPSTORE_FILOPNERR	Unable to open the old profile file for reading or create the new profile file for writing.
POPSTORE_INSUFPRIV	The calling process lacks SYSLCK privilege (OpenVMS only).
POPSTORE_LCKOPNERR	Unable to obtain lock information for one of the profile files.
POPSTORE_LOCKERROR	Unable to obtain a lock for one of the profile files.
POPSTORE_NOTINIT	The API has not yet been initialized.
POPSTORE_READERROR	An error while attempting to read data from the old profile file.
POPSTORE_SEEKERROR	an error occurred while attempting to seek to a position in one of the profile files.
POPSTORE_TOOLONG	The length of the `new_user` string exceeds POPSTORE_ MAX_USER_LEN bytes or is otherwise too long; the length of the `new_password` string exceeds a length of POPSTORE_MAX_PWD_LEN bytes; or a resulting profile file name is too long.
POPSTORE_USEREXISTS	An account with the name `new_user` already exists.
POPSTORE_VMERROR	Insufficient virtual memory to perform requested operation.
POPSTORE_WRITERROR	An error occurred while writing data to the new profile file.

POPSTORE_user_copy_d

Copy or rename an existing user account.

FORMAT

```
int POPSTORE_user_copy_d (old_domain, old_domain_len,
                          old_user, old_user_len,
                          new_domain, new_domain_len,
                          new_user, new_user_len,
                          new_password, new_password_len,
                          do_rename)
      char *old_domain;
      int   old_domain_len;
      char *old_user;
      int   old_user_len;
      char *new_domain;
      int   new_domain_len;
      char *new_user;
      int   new_user_len;
      char *new_password;
      int   new_password_len;
      int   do_rename;
```

ARGUMENTS

`old_domain`
Name of the user domain for the old user account. To use the `default` domain, supply a value of `NULL`. Used for input only.

`old_domain_len`
Length in bytes of the user domain name, `old_domain`. To use the `default` domain, supply a value of zero. Used for input only.

`old_user`
Name of the popstore user to copy or rename. Used for input only.

`old_user_len`
Length in bytes of the string, `old_user`. Used for input only.

`new_domain`
Name of the user domain for the new user account. To use the `default` domain, supply a value of `NULL`. Used for input only.

`new_domain_len`
Length in bytes of the user domain name, `new_domain`. To use the `default` domain, supply a value of zero. Used for input only.

`new_user`
Name of the new popstore user to copy or rename the `old_user` account to. Used for input only.

`new_user_len`
Length in bytes of the string, `new_user`. Used for input only.

new_password
Optional argument specifying a new, plain text password to use for the new or renamed user profile. Used for input only.

new_password_len
Length in bytes of the new, plain text password string, new_password. Used for input only.

do_rename
Boolean flag indicating whether to do a rename or a copy operation. Used for input only.

DESCRIPTION

To duplicate an account or change the username associated with an existing account, use the POPSTORE_user_copy_d subroutine. A copy operation is performed when do_rename has a value of zero; otherwise, a change of username (*i.e.*, rename) operation is performed.

When an account is copied, do_rename=0, the message list and accounting information is not copied to the new account. The name of the account to copy is specified with old_user and the name of the new account to copy to is given by new_user. If an account already exists with the name new_user, an error is returned and no copy is made.

When an account is renamed, do_rename=1, the entire user account is copied over to the new account and then the old account is deleted. The stored message list and accounting information from the old account is copied over to the new account.

An account copy operation is typically used to create new accounts: so as to effect default settings in the new account, the default account is copied to create the new account. Non-default settings (*e.g.*, the owner field) are then made with POPSTORE_user_update. Account rename operations a usually done for one reason only: to change the username associated with an account. As an example of performing a copy operation, see Example 12–2.

Optionally, the password for the new or renamed account can be set. To set a new password, supply the new plain text password and length for the new_password and new_password_len arguments. If you do not want to set or change the password, supply a null value for new_password.

When an error occurs, the new account is not created or the old account renamed or deleted.

RETURN VALUES

POPSTORE_SUCCESS	Normal, successful completion.
POPSTORE_BADLENGTH	The value of new_user_len is less than 1.
POPSTORE_FILOPNERR	Unable to open the old profile file for reading or create the new profile file for writing.
POPSTORE_INSUFPRIV	The calling process lacks SYSLCK privilege (OpenVMS only).

POPSTORE_LCKOPNERR	Unable to obtain lock information for one of the profile files.
POPSTORE_LOCKERROR	Unable to obtain a lock for one of the profile files.
POPSTORE_NOTINIT	The API has not yet been initialized.
POPSTORE_READERROR	An error while attempting to read data from the old profile file.
POPSTORE_SEEKERROR	an error occurred while attempting to seek to a position in one of the profile files.
POPSTORE_TOOLONG	The length of the `new_user` string exceeds POPSTORE_ MAX_USER_LEN bytes or is otherwise too long; the length of the `new_password` string exceeds a length of POPSTORE_MAX_PWD_LEN bytes; or a resulting profile file name is too long.
POPSTORE_USEREXISTS	An account with the name `new_user` already exists.
POPSTORE_VMERROR	Insufficient virtual memory to perform requested operation.
POPSTORE_WRITERROR	An error occurred while writing data to the new profile file.

POPSTORE_user_create

Create a new popstore account.

FORMAT

```
int POPSTORE_user_create (user_context)
     POPSTORE_user_context **user_context;
```

ARGUMENTS

`user_context`
Address of a pointer to a `POPSTORE_user_context` structure created by a prior call to `POPSTORE_user_create_set`. Used for input and output.

DESCRIPTION

Once a `POPSTORE_user_context` structure has been initialized with calls to `POPSTORE_user_create_set`, a user account is then created with `POPSTORE_user_create`. `POPSTORE_user_create` will create a new profile file and user database entry containing the information contained in the `POPSTORE_user_context` structure. Once the account is created, it can then begin receiving mail and be accessed by POP clients.

When the account is created, the last billing field for the account is set to the creation time for the profile file.

An example of using `POPSTORE_user_create` and `POPSTORE_user_create_set` is given in Example 12–1.

RETURN VALUES

POPSTORE_SUCCESS	Normal, successful completion.
POPSTORE_DBBADLENGTH	New user record would not fit in the user database. New account not created. In practice, this error should never occur.
POPSTORE_DBERR	An unknown error occurred while attempting to add the new user record to the user database. New account not created. In practice, this error should never occur.
POPSTORE_DBOPNERR	Unable to open the user database. New account not created.
POPSTORE_DBPUTERR	Unable to add the new user record to the user database. New account not created.
POPSTORE_DIRCRTERR	Unable to create a directory in the required directory path. New account not created.
POPSTORE_FILCLSERR	An error occurred while closing the new profile file. New account was created.
POPSTORE_FILOPNERR	Unable to create the new profile file for writing. New account not created.
POPSTORE_ILLUSRNAM	Username field has zero length. New account not created.

POPSTORE_INSUFPRIV	The process lacks SYSLCK privilege (OpenVMS only). New account not created.
POPSTORE_LCKOPNERR	Unable to obtain lock information for the profile file. New account not created.
POPSTORE_LOCKERROR	Unable to obtain a lock for the profile file. New account not created.
POPSTORE_NOPRIVILEGE	MANAGE flag was set but xxx.
POPSTORE_NOSUCHUSER	Username contains illegal characters or is too long. New account not created.
POPSTORE_NULLCONTEXT	**user_context** argument was NULL. New account not created.
POPSTORE_TOOLONG	Resulting file name for the profile file is too long. This error usually should not occur. New account not created.
POPSTORE_USEREXISTS	Username conflicts with an already existing account. New account not created.
POPSTORE_WRITERROR	An error occured while attempting to write the new profile file. New profile file deleted; new account not created.

POPSTORE_user_create_dispose

Dispose of a context created by POPSTORE_user_create_set.

FORMAT

```
int POPSTORE_user_create_dispose (user_context)
      POPSTORE_user_context **user_context;
```

ARGUMENTS

`user_context`
Address of a pointer to a `POPSTORE_user_context` structure created by a prior call to `POPSTORE_user_create_set`. Used for input and output.

DESCRIPTION

Should you want to cancel creating a user account, then call `POPSTORE_user_create_dispose` to dispose of the context created by prior calls to `POPSTORE_user_create_set`.

RETURN VALUES

POPSTORE_SUCCESS	Normal, successful completion.

POPSTORE_user_create_set

Set the value of a field in a `POPSTORE_user_context` structure.

FORMAT

```
int POPSTORE_user_create_set (user_context, operation, ...)
     POPSTORE_user_context **user_context;
     int                    operation;
```

ARGUMENTS

`user_context`
Address of a pointer to a POPSTORE_user_context structure in which to set the specified items. Used for input and output.

`operation`
Operation to perform indicating which field in the data structure to set. Used for input only.

DESCRIPTION

POPSTORE_user_create_set is used to set fields in a `POPSTORE_user_context` structure which will then subsequently be used with `POPSTORE_user_create` to create a new popstore account. `POPSTORE_user_create_set` cannot be used to change values for an existing account; use `POPSTORE_user_update` for such purposes. An example of using `POPSTORE_user_create` and `POPSTORE_user_create_set` is given in Example 12–1.

`POPSTORE_user_create_set` is used by calling it repeatedly to set the values of different fields in a `POPSTORE_user_context` structure which is the core structure representing a user account. When setting fields for a new account, set the **user_context** variable to `NULL` prior to the first call to `POPSTORE_user_create_set`; *e.g.*,

```
POPSTORE_user_context *user_context;

user_context = NULL;
istat = POPSTORE_user_create_set (&user_context, POPSTORE_SET_USERNAME,
                                  username, strlen (username));
istat = POPSTORE_user_create_set (&user_context, POPSTORE_SET_PASSWORD,
                                  password, strlen (password));
```

On the first call, a context will be allocated and initialized and its address returned in **user_context**. This context is then used with subsequent calls for this particular account. When `POPSTORE_user_create` is called, the account is then created and the context deallocated. To create another new account, again set the **user_context** to `NULL` prior to calling `POPSTORE_user_create_set`. If you want to abort creating an account, use `POPSTORE_user_create_dispose` to dispose of a context created by `POPSTORE_user_create_set`.

The `operation` argument specifies the operation to perform. The accepted values for `operation` are listed in Table 12–4. Depending upon the selected operation, one or two additional arguments can need to be specified when calling `POPSTORE_user_create_set`; refer to Table 12–4 for specific details. Note that the `MANAGE` usage flag cannot be set or cleared without first calling `POPSTORE_manage` to authorize such activity.

Table 12–4 POPSTORE_user_update Operations

Item codes requiring no additional call arguments	
Operation	**Usage**
POPSTORE_SET_FLAGS_DELETE	Set the DELETE usage flag.
POPSTORE_SET_FLAGS_DISMAIL	Set the DISMAIL usage flag.
POPSTORE_SET_FLAGS_DISUSER	Set the DISUSER usage flag.
POPSTORE_SET_FLAGS_LOCKPWD	Set the LOCKPWD usage flag.
POPSTORE_SET_FLAGS_MANAGE	Set the MANAGE usage flag; requires a prior call to POPSTORE_manage.
POPSTORE_SET_FLAGS_MIGRATED	Set the MIGRATED usage flag.
POPSTORE_SET_FLAGS_PWD_ ELSEWHERE	Set the PWD_ELSEWHERE usage flag.
POPSTORE_SET_FLAGS_NO_DELETE	Clear the DELETE usage flag.
POPSTORE_SET_FLAGS_NO_DISMAIL	Clear the DISMAIL usage flag.
POPSTORE_SET_FLAGS_NO_DISUSER	Clear the DISUSER usage flag.
POPSTORE_SET_FLAGS_NO_LOCKPWD	Clear the LOCKPWD usage flag.
POPSTORE_SET_FLAGS_NO_MANAGE	Clear the MANAGE usage flag; requires a prior call to POPSTORE_manage.
POPSTORE_SET_FLAGS_NO_MIGRATED	Clear the MIGRATED usage flag.
POPSTORE_SET_FLAGS_NO_PWD_ ELSEWHERE	Clear the PWD_ELSEWHERE usage flag.
POPSTORE_SET_STORE_TYPE_IMAP	Set the message store type to be IMAP.
POPSTORE_SET_STORE_TYPE_NATIVE	Set the message store type to be NATIVE.
POPSTORE_SET_STORE_TYPE_POP	Set the message store type to be POP.

Table 12–4 (Cont.) POPSTORE_user_update Operations

Item codes requiring one additional call argument of type time_t

Item code	Usage
POPSTORE_SET_LAST_BILLING	Set the value for the last billing time field. The value of this field is a C time_t type expressing the number of seconds since 1 January 1970 and is passed as the third argument. Note that POPSTORE_user_add will override this field with the current time.
POPSTORE_SET_LAST_CONNECT	Set the value of the last connect time field. The value of this field is a C time_t type expressing the number of seconds since 1 January 1970 and is passed as the third argument. Use a value of zero to indicate no prior connections have been made.
POPSTORE_SET_LAST_DISCONNECT	Set the value of the last disconnect time field. The value of this field is a C time_t type expressing the number of seconds since 1 January 1970 and is passed as the third argument. Use a value of zero to indicate that no prior connections have been made.
POPSTORE_SET_LAST_PWD_CHANGE	Set the value of the last password change time field. The value of this field is a C time_t type expressing the number of seconds since 1 January 1970 and is passed as the third argument. Use a value of zero to indicate that the account password is pre-expired.

Item codes requiring one additional call argument of type uint32

Item code	Usage
POPSTORE_SET_OVERDRAFT	Set the overdraft storage quota. The value of this field is measured in units of bytes.
POPSTORE_SET_PAST_BLOCK_DAYS	Set the past block days accounting field. The value of this field is measured in units of block days where a block is 1024 bytes.
POPSTORE_SET_QUOTA	Set the primary storage quota.
POPSTORE_SET_RECEIVED_BYTES	Set the cumulative count of received message bytes.
POPSTORE_SET_RECEIVED_MESSAGES	Set the cumulative count of received messages.
POPSTORE_SET_TOTAL_CONNECT	Set the elapsed total connect time. The value of this field is measured in units of seconds.
POPSTORE_SET_TOTAL_CONNECTIONS	Set the elapsed count of total connections time.

Table 12–4 (Cont.) POPSTORE_user_update Operations

Item codes requiring two additional call arguments of types int, char *	
Item code	**Usage**
POPSTORE_SET_GROUP_NAME	Set the value of the group field. The third argument to POPSTORE_user_data_set should be the length in bytes of the string value and the fourth argument a pointer to the string value. The length of the string value can not exceed POPSTORE_MAX_GROUP_LEN bytes.
POPSTORE_SET_OWNER	Set the value of the owner field. The third argument to POPSTORE_user_data_set should be the length in bytes of the string value and the fourth argument a pointer to the string value. The length of the string value can not exceed POPSTORE_MAX_OWN_LEN bytes.
POPSTORE_SET_PASSWORD	Set the value of the password field. The third argument to POPSTORE_user_data_set should be the length in bytes of the string value and the fourth argument a pointer to the string value. The password should be supplied in plain text form; it will automatically be encrypted. The length of the string value can not exceed POPSTORE_MAX_PWD_LEN bytes.
POPSTORE_SET_PRIVATE	Set the value of the site-defined private data field. The third argument to POPSTORE_user_data_set should be the length in bytes of the string value and the fourth argument a pointer to the string value. The length of the string value can not exceed POPSTORE_MAX_PRIV_LEN bytes.
POPSTORE_SET_USERNAME	Set the value of the username field. The third argument to POPSTORE_user_data_set should be the length in bytes of the string value and the fourth argument a pointer to the string value. The length of the string value can not exceed POPSTORE_MAX_USER_LEN bytes.

RETURN VALUES

POPSTORE_SUCCESS	Normal, successful completion.
POPSTORE_BADARG	Unrecognized value specified for `operation`.
POPSTORE_BADLENGTH	Negative value specified for a length argument.
POPSTORE_NOPRIVILEGE	An attempt was made to set or clear the MANAGE flag without prior authorization via a POPSTORE_manage call.

POPSTORE_TOOLONG	String value exceeds maximum allowed length.
POPSTORE_VMERROR	Insufficient virtual memory. Operation failed; no context created.

POPSTORE_user_delete

Obsolete subroutine: use the more general POPSTORE_user_delete_d subroutine.

FORMAT

```
int POPSTORE_user_delete (user, user_len, do_return)
      char *user;
      int   user_len;
      int   do_return;
```

ARGUMENTS

`user`
Username for the popstore account to delete. Used for input only.

`user_len`
Length in bytes of the username string, `user`. Used for input only.

`do_return`
Boolean flag indicating whether or not unread mail should be silently deleted or returned as unread. Used for input only.

DESCRIPTION

While this subroutine is still supported, it is now obsolete. Use the more general POPSTORE_user_delete_d subroutine.

RETURN VALUES

POPSTORE_SUCCESS	Normal, successful completion.
POPSTORE_FILDELERR	Unable to delete the underlying profile file.
POPSTORE_FILOPNERR	Unable to open the profile file for reading.
POPSTORE_INSUFPRIV	Insufficient privileges to access the profile file, or, on OpenVMS, the process lacks SYSLCK privilege.
POPSTORE_LCKOPNERR	Unable to obtain lock information for the profile file.
POPSTORE_LOCKERROR	Unable to obtain a lock for the profile file.
POPSTORE_NOSUCHUSR	No such user account.
POPSTORE_READERROR	An error while attempting to read data from the profile file.
POPSTORE_TOOLONG	The resulting profile file name is too long.
POPSTORE_VMERROR	Insufficient virtual memory.

POPSTORE_user_delete_d

Delete a user account.

FORMAT

```
int POPSTORE_user_delete_d (domain, domain_len, user,
                            user_len, do_return)
      char *domain;
      int   domain_len;
      char *user;
      int   user_len;
      int   do_return;
```

ARGUMENTS

domain
Name of the user domain to use. Supply a value of NULL to indicate the default domain. Used for input only.

domain_len
Length in bytes of the user domain name, domain. Supply a value of zero to indicate the default domain. Used for input only.

user
Username for the popstore account to delete. Used for input only.

user_len
Length in bytes of the username string, user. Used for input only.

do_return
Boolean flag indicating whether or not unread mail should be silently deleted or returned as unread. Used for input only.

DESCRIPTION

The POPSTORE_user_delete_d subroutine is used to delete a user's account and, optionally, to return any unread messages stored for that user. The user argument specifies the name of the account to delete. When do_return has a non-zero value, any unread messages stored for the user are returned as unread to their originators.

RETURN VALUES

POPSTORE_SUCCESS	Normal, successful completion.
POPSTORE_FILDELERR	Unable to delete the underlying profile file.
POPSTORE_FILOPNERR	Unable to open the profile file for reading.
POPSTORE_INSUFPRIV	Insufficient privileges to access the profile file, or, on OpenVMS, the process lacks SYSLCK privilege.
POPSTORE_LCKOPNERR	Unable to obtain lock information for the profile file.

POPSTORE_LOCKERROR	Unable to obtain a lock for the profile file.
POPSTORE_NOSUCHUSR	No such user account.
POPSTORE_READERROR	An error while attempting to read data from the profile file.
POPSTORE_TOOLONG	The resulting profile file name is too long.
POPSTORE_VMERROR	Insufficient virtual memory.

POPSTORE_user_end

Deaccess a user account.

FORMAT

```
int POPSTORE_user_end (user_context)
     POPSTORE_user_context *user_context;
```

ARGUMENTS

`user_context`
User context to dispose of. Used for input only.

DESCRIPTION

When finished with a user context, it must be disposed of by calling POPSTORE_user_end. Not only does this call dispose of allocated resources, it also deletes any messages marked for deletion, and updates accounting information in the user's profile if requested by the `do_accounting` argument of POPSTORE_user_begin_d.

RETURN VALUES

POPSTORE_SUCCESS	Normal, successful completion.
POPSTORE_FILOPNERR	An error occurred while trying to re-open the user's profile file for updating.
POPSTORE_INSUFPRIV	The calling process lacks SYSLCK privilege (OpenVMS only).
POPSTORE_LCKOPNERR	Unable to obtain lock information for the user's profile file.
POPSTORE_LOCKERROR	Unable to obtain a lock for the user's profile file.
POPSTORE_READERROR	An error occurred while attempting to read data from the user's profile file.
POPSTORE_SEEKERROR	An error occurred while attempting to seek to a position in the user's profile files.
POPSTORE_VMERROR	Insufficient virtual memory to perform requested operation.
POPSTORE_WRITERROR	An error occurred while writing data to the user's profile file.

POPSTORE_user_exists

Obsolete subroutine: use the more general POPSTORE_user_exists_d subroutine.

FORMAT

```
int POPSTORE_user_exists (user, user_len)
      char *user;
      int   user_len;
```

ARGUMENTS

`user`
Name of the user to check for the existence of. Used for input only.

`user_len`
Length in bytes of the username string, `user`. Used for input only.

DESCRIPTION

While this subroutine is still supported, it is now obsolete. Use the more general `POPSTORE_user_exists_d` subroutine.

RETURN VALUES

POPSTORE_USEREXISTS	Username corresponds to an existing popstore account.
POPSTORE_NOSUCHUSER	No matching account or the process lacks privileges to access the profile directories.

POPSTORE_user_exists_d

See if a username specifies a valid account.

FORMAT

```
int POPSTORE_user_exists_d (domain, domain_len, user,
                            user_len)
      char *domain;
      int   domain_len;
      char *user;
      int   user_len;
```

ARGUMENTS

`domain`
Name of the user domain to use. Supply a value of NULL to indicate the `default` domain. Used for input only.

`domain_len`
Length in bytes of the user domain name, `domain`. Supply a value of zero to indicate the `default` domain. Used for input only.

`user`
Name of the user to check for the existence of. Used for input only.

`user_len`
Length in bytes of the username string, `user`. Used for input only.

DESCRIPTION

To see if a username corresponds to an existing popstore account, call POPSTORE_user_exists with the username in question specified with the `user` argument. If the name corresponds to an existing account, the value POPSTORE_USEREXISTS is returned. Otherwise, POPSTORE_NOSUCHUSER is returned. Note that if the process lacks privileges to access the popstore profile directories, POPSTORE_NOSUCHUSER will be returned.

RETURN VALUES

POPSTORE_USEREXISTS	Username corresponds to an existing popstore account.
POPSTORE_NOSUCHUSER	No matching account or the process lacks privileges to access the profile directories.

POPSTORE_user_list

Obsolete subroutine: use the more general `POPSTORE_user_list_d` subroutine.

FORMAT

```
int POPSTORE_user_list (list_context, pattern, pattern_len,
                        user, user_len, max_user_len)
      POPSTORE_list_context **list_context;
      char                   *pattern;
      int                     pattern_len;
      char                   *user;
      int                    *user_len;
      int                     max_user_len;
```

ARGUMENTS

`user_context`
List context maintained by POPSTORE_user_list. Used for input and output.

`pattern`
Optional username pattern. Used for input only.

`pattern_len`
Length in bytes of the optional username pattern, `pattern`. Used for input only.

`user`
Buffer to receive the returned username. Length of this buffer should be at least POPSTORE_MAX_USER_LEN+1 bytes long. Used for output only.

`user_len`
Length in bytes of the username returned in `user`. Used for output only.

`max_user_len`
Maximum length in bytes of the buffer, `user`. Used for input only.

DESCRIPTION

Although still supported, `POPSTORE_user_list` is obsolete. Instead use the more general `POPSTORE_user_list_d` subroutine.

RETURN VALUES

POPSTORE_SUCCESS	Normal, successful completion.
POPSTORE_BADARG	Bad value supplied for the `list_context`, `user`, `user_len`, or `max_user_len` argument.
POPSTORE_DBBADLENGTH	Retrieved a record from the database which has an unexpected length. In practice, this error should never occur.
POPSTORE_DBERR	An unknown error occurred while attempting to read from the user database. In practice, this error should never occur.

POPSTORE_DBOPNERR	Unable to open the user database.
POPSTORE_DBMISSING	User database does not exist; create it with the X-BUILD-USER-DB command of the command-line popstore management utility.
POPSTORE_EOM	No more usernames to return
POPSTORE_FILOPNERR	Unable to open the profile path file. See Section 14.2 for details.
POPSTORE_INSUFPRIV	Process has insufficient privileges to access the user database.
POPSTORE_VMERROR	Insufficient virtual memory.

POPSTORE_user_list_d

Return one-by-one the usernames associated with each account.

FORMAT

```
int POPSTORE_user_list_d (list_context, pattern, pattern_len,
                          domain, domain_len, group,
                          group_len, user, user_len,
                          max_user_len)
      POPSTORE_list_context **list_context;
      char                    *pattern;
      int                      pattern_len;
      char                    *domain;
      int                      domain_len;
      char                     group;
      int                      group_len;
      char                    *user;
      int                     *user_len;
      int                      max_user_len;
```

ARGUMENTS

`user_context`
List context maintained by POPSTORE_user_list_d. Used for input and output.

`pattern`
Optional username pattern. Used for input only.

`pattern_len`
Length in bytes of the optional username pattern, `pattern`. Used for input only.

`domain`
Name of the user domain to use. Supply a value of NULL to indicate the `default` domain. Used for input only.

`domain_len`
Length in bytes of the user domain name, `domain`. Supply a value of zero to indicate the `default` domain. Used for input only.

`group`
Optional name of a management group to confine the listing to. Used for input only.

`group_len`
Length in bytes of the optional group name, `group`. Used for input only.

`user`
Buffer to receive the returned username. Length of this buffer should be at least POPSTORE_MAX_USER_LEN+1 bytes long. Used for output only.

`user_len`
Length in bytes of the username returned in `user`. Used for output only.

`max_user_len`
Maximum length in bytes of the buffer, `user`. Used for input only.

DESCRIPTION

POPSTORE_user_list_d can be used to obtain one at a time the usernames associated with each popstore account. To call POPSTORE_user_list_d, you must have a variable of type "POPSTORE_list_context *" available, the address of which you specify with the `list_context` argument. Prior to calling POPSTORE_user_list_d for the first time, set the value of that pointer variable to null. That will signify to POPSTORE_user_list_d that this is the start of a listing. POPSTORE_user_list_d will create a context and point your pointer variable at that context. Repeatedly call POPSTORE_user_list_d to obtain each username. When there are no more usernames to return, the context will be deallocated and POPSTORE_EOM returned. Sample code is provided in Example 12–3

The optional `group` and `group_len` arguments can be used to restrict the listing to just those accounts contained within the specified management group and any subgroups thereof. The group name specification can not contain wild card characters. If you do not want to restrict a listing, then specify null and 0 for these two parameters.

The optional `pattern` and `pattern_len` arguments can be used to specify a pattern which the returned usernames must match. The pattern can contain wild card characters. For example, to list all accounts with usernames starting with the letter "z" supply the string value "z*" and length 2 for these two arguments. To return all usernames, instead supply the values null and 0.

To prematurely end a listing, call POPSTORE_user_list_abort so as to properly dispose of the listing context.

While POPSTORE_user_list_d can be used to produce lists of user accounts, often using POPSTORE_format_profiles is more appropriate. POPSTORE_user_list_d is intended for situations where code needs to be able to obtain, one at a time, the usernames for all or some accounts. For instance, to periodically bill all user accounts with POPSTORE_user_billing_d.

RETURN VALUES

POPSTORE_SUCCESS	Normal, successful completion.
POPSTORE_BADARG	Bad value supplied for the `list_context`, `user`, `user_len`, or `max_user_len` argument.
POPSTORE_DBBADLENGTH	Retrieved a record from the database which has an unexpected length. In practice, this error should never occur.
POPSTORE_DBERR	An unknown error occurred while attempting to read from the user database. In practice, this error should never occur.
POPSTORE_DBOPNERR	Unable to open the user database.

POPSTORE_DBMISSING	User database does not exist; create it with the X-BUILD-USER-DB command of the command-line popstore management utility.
POPSTORE_EOM	No more usernames to return
POPSTORE_FILOPNERR	Unable to open the profile path file. See Section 14.2 for details.
POPSTORE_INSUFPRIV	Process has insufficient privileges to access the user database.
POPSTORE_TOOLONG	Length of specified group name is too long.
POPSTORE_VMERROR	Insufficient virtual memory.

POPSTORE_user_list_abort

Prematurely dispose of a list context generated by POPSTORE_user_list_d or POPSTORE_user_list.

FORMAT

```
int POPSTORE_user_list_abort (list_context)
      POPSTORE_list_context **list_context;
```

ARGUMENTS

`user_context`
List context generated by POPSTORE_user_list_d or POPSTORE_user_list. Used for input and output.

DESCRIPTION

List contexts generated by POPSTORE_user_list_d and POPSTORE_user_list will normally be disposed of automatically when there are no more usernames to be listed. To prematurely dispose of a list context, call POPSTORE_user_list_abort.

RETURN VALUES

POPSTORE_SUCCESS	Normal, successful completion.

POPSTORE_user_pw_change

Obsolete subroutine: use the more general `POPSTORE_user_pw_change_d` subroutine.

FORMAT

```
int POPSTORE_user_pw_change (user, user_len, new_password, new_pas
                             errmsg, errmsg_len, errmsg_max)
      char *user;
      int   user_len;
      char *new_password;
      int   new_password_len;
      char *errmsg;
      int  *errmsg_len;
      int   errmsg_max;
```

ARGUMENTS

`user`
Name of the user to change the password for. Used for input only.

`user_len`
Length of the username string, `user`. Used for input only.

`new_password`
New password to use for the account. Used for input only.

`new_password_len`
Length in bytes of the new password string, `new_password`. Used for input only.

`errmsg`
Address of character array that `POPSTORE_user_pw_change` can put an error message into if the password change fails.

`errmsg_len`
Address of an integer that `POPSTORE_user_pw_change` can write the length of the error message that it put into `errmsg`.

`errmsg_max`
The size of the array that `errmsg` points to.

DESCRIPTION

While still supported, this subroutine is now obsolete. Use the more general `POPSTORE_user_pw_change_d` subroutine.

RETURN VALUES

POPSTORE_SUCCESS	Normal, successful completion.
POPSTORE_FILOPNERR	Unable to open the profile file for reading.

POPSTORE_INSUFPRIV	Insufficient privileges to access the profile file, or, on OpenVMS, the process lacks SYSLCK privilege.
POPSTORE_LCKOPNERR	Unable to obtain lock information for the profile file.
POPSTORE_LOCKERROR	Unable to obtain a lock for the profile file.
POPSTORE_NOSUCHUSR	No such user account.
POPSTORE_READERROR	An error while attempting to read data from the profile file.
POPSTORE_TOOLONG	The resulting profile file name is too long.
POPSTORE_VMERROR	Insufficient virtual memory.
POPSTORE_WRITERROR	An error occurred while writing data to the user’s profile file.

POPSTORE_user_pw_change_d

Change a user’s password.

FORMAT

```
int POPSTORE_user_pw_change_d (domain, domain_len, user,
                               user_len, new_password, new_passwor
                               errmsg, errmsg_len, errmsg_max)
      char *domain;
      int   domain_len;
      char *user;
      int   user_len;
      char *new_password;
      int   new_password_len;
      char *errmsg;
      int  *errmsg_len;
      int   errmsg_max;
```

ARGUMENTS

`domain`
Name of the user domain to use. Supply a value of NULL to indicate the `default` domain. Used for input only.

`domain_len`
Length in bytes of the user domain name, `domain`. Supply a value of zero to indicate the `default` domain. Used for input only.

`user`
Name of the user to change the password for. Used for input only.

`user_len`
Length of the username string, `user`. Used for input only.

`new_password`
New password to use for the account. Used for input only.

`new_password_len`
Length in bytes of the new password string, `new_password`. Used for input only.

`errmsg`
Address of character array that `POPSTORE_user_pw_change` can put an error message into if the password change fails.

`errmsg_len`
Address of an integer that `POPSTORE_user_pw_change` can write the length of the error message that it put into `errmsg`.

`errmsg_max`
The size of the array that `errmsg` points to.

DESCRIPTION

A user's password can be changed with POPSTORE_user_pw_change_d. As input, supply the name of the user to effect the change for as well as the new, plain text password to use, and the user domain. The length of the new password can not exceed POPSTORE_MAX_PWD_LEN bytes.

Note that if you already have a user context from POPSTORE_user_begin_d, then you can call POPSTORE_user_update to change the password.

RETURN VALUES

POPSTORE_SUCCESS	Normal, successful completion.
POPSTORE_FILOPNERR	Unable to open the profile file for reading.
POPSTORE_INSUFPRIV	Insufficient privileges to access the profile file, or, on OpenVMS, the process lacks SYSLCK privilege.
POPSTORE_LCKOPNERR	Unable to obtain lock information for the profile file.
POPSTORE_LOCKERROR	Unable to obtain a lock for the profile file.
POPSTORE_NOSUCHUSR	No such user account.
POPSTORE_READERROR	An error while attempting to read data from the profile file.
POPSTORE_TOOLONG	The resulting profile file name is too long.
POPSTORE_VMERROR	Insufficient virtual memory.
POPSTORE_WRITERROR	An error occurred while writing data to the user's profile file.

POPSTORE_user_pw_check

Perform an authentication check.

FORMAT

```
int POPSTORE_user_pw_check (user_context, password,
                            password_len, auth_type,
                            challenge, challenge_len,
                            response, response_len)
      POPSTORE_user_context *user_context;
      char                  *password;
      int                    password_len;
      int                    auth_mechanism;
      char                  *challenge;
      int                    challenge_len;
      char                  *response;
      int                    response_len;
```

ARGUMENTS

`user_context`
User context to authenticate the password against. Used for input only.

`password`
Plain text password to authenticate against the user context. Used for input only.

`password_len`
Length in bytes of the plain text password string, `password`. Used for input only.

`auth_mechanism`
Authentication mechanism to use to check the password. Used for input only.

`challenge`
Challenge string presented as part of a challenge-response authentication mechanism. Used for input only.

`challenge_len`
Length in bytes of the challenge string, `challenge`. Used for input only.

`response`
Response to the challenge. Used for input only.

`response_len`
Length in bytes of the response string, `response`. Used for input only.

DESCRIPTION

To authenticate a user against the information contained in a user profile, call POPSTORE_user_pw_check supplying the authentication mechanism to use and the authentication credentials:

`auth_mechanism`	**Required authentication credentials**
POPSTORE_AUTH_MECH_PLAIN	Password supplied in the clear; supply the `password` and `password_len` arguments; challenge and response arguments are ignored.

If the authentication credentials are verified to be correct and the account is not marked DISUSER, a status of POPSTORE_SUCCESS is returned. Otherwise, a status of POPSTORE_DISUSERD or POPSTORE_NOMATCH is returned.

Note that the value of the field `user_context->profile->password` is encrypted and as such applications cannot directory validate popstore passwords.

RETURN VALUES

POPSTORE_SUCCESS	Authentication successful.
POPSTORE_DISUSERD	Account is marked DISUSER; authentication failed.
POPSTORE_NOMATCH	Authentication failed; invalid password.
POPSTORE_NULLCONTEXT	`user_context` argument had a null value.
POPSTORE_NULLPROFILE	`user_context->profile` is null.

POPSTORE_user_update

Update a field in an existing account.

FORMAT

```
int POPSTORE_user_update (user_context, item_list
                          errmsg, errmsg_len, errmsg_max)
      POPSTORE_user_context *user_context;
      POPSTORE_item_list    *item_list;
      char *errmsg;
      int  *errmsg_len;
      int   errmsg_max;
```

ARGUMENTS

`user_context`
User context to update. Used for input and output.

`item_list`
Item list describing the updates to perform. Used for input only.

`errmsg`
Address of character array that `POPSTORE_user_update` can put an error message into if the password change fails.

`errmsg_len`
Address of an integer that `POPSTORE_user_update` can write the length of the error message that it put into `errmsg`.

`errmsg_max`
The size of the array that `errmsg` points to.

DESCRIPTION

One or more fields in a popstore profile file can be changed by calling POPSTORE_user_update. The fields to update and the new values to use are described using an array of item descriptors, each of which specifies an action and provides the information needed to perform that action. The list of item descriptors is terminated with an entry with the POPSTORE_SET_END item code. The list of item descriptors is referred to as an item list.

When the item list is successfully processed, both the on-disk and in-memory profile is updated and the POPSTORE_SUCCESS item code is returned.[6] Should an error occur while processing the item list, no updates are made to the underlying user profile file, and POPSTORE_user_end returns an error status. Depending upon the nature of the error, the in-memory profile data can have been changed.

Note that the username field can not be changed with POPSTORE_user_update. Use POPSTORE_user_copy_d with `do_rename`=1 to change the username field for

[6] The in-memory profile is that pointed at by `user_context->profile`.

an existing account. Moreover, the MANAGE usage flag can not be set or cleared without first calling POPSTORE_manage to authorize such activity.

The `item_list` argument is a pointer to an array of one or more item descriptors whose type is given by the `POPSTORE_item_list` structure shown below:

```
typedef struct {
  int   item_code;
  void *item_address;
  int   item_length;
} POPSTORE_item_list;
```

The interpretation of the three fields in an item descriptor is as follows:

POPSTORE_item_list item descriptor fields

`item_code`
A signed, longword integer containing a user-supplied symbolic code specifying an action to be taken by PMDF_user_update. A description of each item code follows this list of item descriptor fields.

`item_address`
A pointer to the value to use in updating the profile field indicated by the associated `item_code`. Not all actions require that a value be supplied for the `item_address` field.

`item_length`
A signed, longword integer containing the user-supplied length of the data pointed at by `item_address`. Not all actions require that an `item_length` be specified.

The `item_code` values accepted by POPSTORE_user_update are as follows:

POPSTORE_user_update item codes

POPSTORE_SET_CHAIN
This item code can be used to direct the item list processing to another item list. The value of the `item_address` field should be a pointer to another POPSTORE_item_list structure. The `item_length` field is ignored.

POPSTORE_SET_END
The end of the item list is indicated by this item code. The `item_address` and `item_length` fields are ignored.

POPSTORE_SET_FLAGS_DELETE
Set the DELETE usage flag. The `item_address` and `item_length` fields are ignored.

POPSTORE_SET_FLAGS_DISMAIL
Set the DISMAIL usage flag. The `item_address` and `item_length` fields are ignored.

POPSTORE_SET_FLAGS_DISUSER
Set the DISUSER usage flag. The `item_address` and `item_length` fields are ignored.

POPSTORE_SET_FLAGS_LOCKPWD
Set the LOCKPWD usage flag. The `item_address` and `item_length` fields are ignored.

POPSTORE_SET_FLAGS_MANAGE
Set the MANAGE usage flag. The `item_address` and `item_length` fields are ignored. A prior call to POPSTORE_manage must be made in order to permit setting the MANAGE flag.

POPSTORE_SET_FLAGS_NO_DELETE
Clear the DELETE usage flag. The `item_address` and `item_length` fields are ignored.

POPSTORE_SET_FLAGS_NO_DISMAIL
Clear the DISMAIL usage flag. The `item_address` and `item_length` fields are ignored.

POPSTORE_SET_FLAGS_NO_DISUSER
Clear the DISUSER usage flag. The `item_address` and `item_length` fields are ignored.

POPSTORE_SET_FLAGS_NO_LOCKPWD
Clear the LOCKPWD usage flag. The `item_address` and `item_length` fields are ignored.

POPSTORE_SET_FLAGS_NO_MANAGE
Clear the MANAGE usage flag. The `item_address` and `item_length` fields are ignored. A prior call to POPSTORE_manage must be made in order to permit clearing the MANAGE flag.

POPSTORE_SET_GROUP_NAME
Set the value of the group field. The value of the `item_address` field must point to a string value. The value of the `item_length` field must be the length in bytes of the string value. That length can not exceed POPSTORE_MAX_GROUP_LEN bytes.

POPSTORE_SET_LAST_BILLING
Set the value for the last billing time field. The value of the `item_address` field must point to a `time_t` value expressing the last billing time as the number of seconds since 1 January 1970. The `item_length` field is ignored.

POPSTORE_SET_LAST_CONNECT
Set the value of the last connect time field. The value of the `item_address` field must point to a `time_t` value expressing the last connect time as the number of seconds since 1 January 1970. The `item_length` field is ignored.

POPSTORE_SET_LAST_DISCONNECT
Set the value of the last disconnect time field. The value of the `item_address` field must point to a `time_t` value expressing the last disconnect time as the number of seconds since 1 January 1970. The `item_length` field is ignored.

POPSTORE_SET_LAST_PWD_CHANGE

Set the value of the last password change time field. The value of the `item_address` field must point to a `time_t` value expressing the last password change time as the number of seconds since 1 January 1970. Use a value of zero to indicate that the account password is pre-expired. The `item_length` field is ignored.

POPSTORE_SET_MESSAGE_COUNT

Set the value of the message count field. If the value, V, is less than the count of currently stored messages, M, then the first M-V stored messages will be deleted. The value of the `item_address` field must point to a `uint32` value denoting the new message count. The `item_length` field is ignored.

POPSTORE_SET_NOOP

This signifies an item descriptor which should be ignored. The `item_address` and `item_length` fields are ignored.

POPSTORE_SET_OVERDRAFT

Set the overdraft storage quota. The value of the `item_address` field must point to a `uint32` value expressing the overdraft storage quota in units of bytes. The `item_length` field is ignored.

POPSTORE_SET_OWNER

Set the value of the owner field. The value of the `item_address` field must point to a string value. The value of the `item_length` field must be the length in bytes of the string value. That length can not exceed POPSTORE_MAX_OWN_LEN bytes.

POPSTORE_SET_PASSWORD

Set the value of the password field. The value of the `item_address` field must point to a string value. The value of the `item_length` field must be the length in bytes of the string value. That length can not exceed POPSTORE_MAX_PWD_LEN bytes.

POPSTORE_SET_PAST_BLOCK_DAYS

Set the past block days accounting field. The value of the `item_address` field must point to a `uint32` value expressing the new past block days value to use. The `item_length` field is ignored.

Note that when the past block days field is set, the past block days remainder field is set to a value of zero.

POPSTORE_SET_PRIVATE

Set the value of the site-defined private data field. The value of the `item_address` field must point to a string value. The value of the `item_length` field must be the length in bytes of the string value. That length can not exceed POPSTORE_MAX_PRIV_LEN bytes.

POPSTORE_SET_QUOTA

Set the primary storage quota. The value of the `item_address` field must point to a `uint32` value expressing the primary storage quota in units of bytes. The `item_length` field is ignored.

POPSTORE_SET_TOTAL_CONNECT

Set the elapsed total connect time. The value of the `item_address` field must point to a `uint32` value expressing the elapsed total connect time in units of seconds. The `item_length` field is ignored.

RETURN VALUES

POPSTORE_SUCCESS	Normal, successful completion.
POPSTORE_BADITMCOD	Unrecognized item code specified. No changes made to the on-disk or in-memory user information.
POPSTORE_BADLENGTH	Length of a string value is less than zero. No changes made to the on-disk or in-memory user information.
POPSTORE_CMDBADVAL	Attempt made to set the message count to a value exceeding the count of currently stored messages. No changes made to the on-disk or in-memory user information.
POPSTORE_FILOPNERR	Unable to open the profile file for reading and writing. No changes made to the on-disk or in-memory user information.
POPSTORE_INSUFPRIV	The process lacks SYSLCK privilege (OpenVMS only). No changes made to the on-disk or in-memory user information.
POPSTORE_LCKOPNERR	Unable to obtain lock information for the profile file. No changes made to the on-disk or in-memory user information.
POPSTORE_LOCKERROR	Unable to obtain a lock for the profile file. No changes made to the on-disk or in-memory user information.
POPSTORE_NOPRIVILEGE	Process is not allowed to set or clear the MANAGE usage flag without a prior call to POPSTORE_manage. No changes made to the on-disk or in-memory user information.
POPSTORE_NULLCONTEXT	`user_context` is null. No changes made to the on-disk or in-memory user information.
POPSTORE_NULLPROFILE	`user_context->profile` context is null. No changes made to the on-disk or in-memory user information.
POPSTORE_PWDLOCKED	Cannot change the password; password is locked. No changes made to the on-disk or in-memory user information.
POPSTORE_READERROR	An error occurred while reading data from the profile file. No changes made to the on-disk or in-memory user information.
POPSTORE_SEEKERROR	An error occurred while seeking to a location in the profile file. No changes made to the on-disk or in-memory user information.
POPSTORE_TOOLONG	Length of string value exceeds permitted length. No changes made to the on-disk or in-memory user information.
POPSTORE_WRITERROR	An error occurred while writing data to the profile file. No changes made to the on-disk profile information; in-memory user information has been changed.

13 Logging

There are two logging interfaces for the popstore. The first is the normal PMDF logging interface as described in Section 13.1. This interface should suit the needs of most sites.

For sites needing more sophisticated logging facilities, there is a subroutine level interface. When enabled, the popstore dynamically loads and links to a site-supplied subroutine and then calls that subroutine each time an event is to be logged. The loggable events are summarized below and discussed in Section 13.2:

- Account creation, modification, and deletion
- Account access (login, password verification calls, logout)
- Message storage, access, and deletion
- Access to the popstore API (servers, management utilities, third-party applications, *etc.*)

Section 13.2.3 shows some sample event logs depicting various activities, recorded through this subroutine interface.

13.1 PMDF-style Logging

Normal PMDF logging facilities can be used to track messages into the popstore and POP3 client access to the store. The former is activated with the `logging` channel keyword; the latter by enabling logging in the POP3 server itself. Both will produce log file entries in the PMDF log file, `mail.log_current`, found in the PMDF log directory.

To use the `logging` channel keyword, edit the PMDF configuration file, `popstore.chans` found in the PMDF table directory, `/pmdf/table/` on UNIX and NT systems and `PMDF_TABLE:` on OpenVMS. In that file, add the `logging` keyword to the end of the line beginning with `holdexquota` so that that line reads

```
holdexquota description "popstore channel" logging
```

Then save the file. After saving the file, recompile your configuration if using a compiled configuration. Also, if on OpenVMS, reinstall the configuration after recompiling. If using the multi-threaded SMTP server, restart it with the PMDF RESTART SMTP command.

See the *PMDF System Manager's Guide* for further information on the `logging` channel keyword and the format of the PMDF log file.

Use the LOGGING option of the POP3 server to enable logging in that server. See the POP3 server documentation in the *PMDF System Manager's Guide* for details on enabling that option.

13.2 The Site-Supplied Logging Interface

Sites needing very detailed popstore activity logging can obtain such detail by providing a subroutine for the popstore to call. The subroutine is provided to the popstore as a shareable image and made known to the popstore via the LOG_ACTIVITY option described in Section 3.3. By default, any “loggable” event will be passed to the site supplied logging routine. Optionally, the LOGGING_ACTIVITY_MASK option can be used to select which events are logged.

The site-supplied subroutine must have the name `log_activity` and takes the form

```
#ifndef __VMS
#  include "/pmdf/com/popstore.h"
#else
#  include "PMDF_COM:POPSTORE.H"
#endif

void log_activity (*log_id, log_subid, log_type,
                   *log_data, log_len)
        char   *log_id;
        uint32  log_subid;
        int32   log_type;
        void   *log_data;
        int32   log_len;
```

where the parameters have the following interpretations:

`log_id`
Character string of length at most 20 bytes which remains the same amongst a related sequence of popstore activities bounded by `POPSTORE_init()` and `POPSTORE_end()` calls.

`log_subid`
Thirty-two bit unsigned integer identifying a related sequence of popstore activities bounded by `POPSTORE_user_begin()` and `POPSTORE_user_end()` calls.

`log_type`
The type of activity being logged. See Section 13.2.2 for a description of the different values for this parameter.

`log_data`
A pointer to the information to be logged. The nature of the data and its organization varies with the type of activity being logged. Refer to Section 13.2.2 for details.

`log_len`
The length in bytes of the information to be logged.

The intent behind the `log_id` and `log_subid` parameters is to provide a means of grouping related activity threads together. For instance, if logging activity to a file, then start each record of the file with the values of the `log_id` and `log_subid` parameters. The log file can then be sorted to produce a file in which related events are grouped together within the (sorted) file.

Each call the popstore makes to the LOG_ACTIVITY subroutine reports an event which the subroutine can then log or discard as it sees fit. The subroutine does not need to be reentrant or thread-safe: within a given process context, the popstore will serialize its calls to the subroutine. However, several processes can simultaneously call the same subroutine. Consequently, if the subroutine, for instance, appends records to a log file, the subroutine must ensure that the writes to that file support simultaneous writers.

In the file `popstore_log_activity.c`, a sample `log_activity` subroutine is provided. This file is in the `/pmdf/doc/examples` directory on UNIX and NT systems and, on OpenVMS systems, the directory `PMDF_ROOT:[DOC.EXAMPLES]`.

13.2.1 Linking a Shared Library

Solaris Systems

On Solaris systems, linking a C program into a shared library is accomplished with the command:

```
# cc -mt -KPIC -G -h filename.so -o filename.so filename.c
```

where `filename.c` is the name of the program to compile link and `filename.so` is the name of the shared library to create.

Linux Systems

On Linux systems, linking a C program into a shared library is accomplished with the command:

```
# gcc -shared -o filename.so filename.c
```

where `filename.c` is the name of the program to compile and link and `filename.so` is the name of the shared library to create.

OpenVMS Systems

On OpenVMS VAX systems, the link command should be of the form

```
$ DEFINE/SYSTEM/EXECUTIVE_MODE POP_LOG_ACTIVITY -
_$            disk:[device]filename.EXE
$ LINK/SHAREABLE=POP_LOG_ACTIVITY object-file-spec,SYS$INPUT:/OPT
UNIVERSAL=log_activity
CTRL/Z
$ INSTALL CREATE POP_LOG_ACTIVITY
```

and on OpenVMS Alpha or I64 systems,

```
$ DEFINE/SYSTEM/EXECUTIVE_MODE POP_LOG_ACTIVITY -
_$            disk:[device]filename.EXE
$ LINK/SHAREABLE=POP_LOG_ACTIVITY object-file-spec,SYS$INPUT:/OPT
SYMBOL_VECTOR=(log_activity=PROCEDURE)
CTRL/Z
$ INSTALL CREATE POP_LOG_ACTIVITY
```

where `object-file-spec` is the name of the object file to be linked to produce the resulting shared image `disk:[device]filename.EXE`. The choice of logical name is arbitrary. Use whatever name you see fit and then use the `LOG_ACTIVITY` option to tell the popstore the logical name to use.

When relinking the image, be sure to issue the command

```
$ INSTALL REPLACE POP_LOG_ACTIVITY
```

so that the new image is instead used.

13.2.2 Logging Data Types

The value of the `log_type` parameter indicates the type of activity being logged and the type of data referenced by the `log_data` pointer. The symbolic names of the values are defined in the `popstore.h` header file and are summarized in Table 13–1. They are fully discussed below.

Table 13–1 Summary of Logging Data Types

Symbolic name	Value	log_data data type
POPSTORE_LOG_START	1	char *
POPSTORE_LOG_END	2	NULL
POPSTORE_LOG_LOGIN_START	3	POPSTORE_user_log *
POPSTORE_LOG_LOGIN_PW_MATCH	4	POPSTORE_user_log *
POPSTORE_LOG_LOGIN_PW_NOMATCH	5	POPSTORE_user_log *
POPSTORE_LOG_LOGIN_END	6	POPSTORE_user_log *
POPSTORE_LOG_MSG_STORE	7	POPSTORE_message_log *
POPSTORE_LOG_MSG_OPEN	8	POPSTORE_message_log *
POPSTORE_LOG_MSG_DELETE	9	POPSTORE_message_log *
POPSTORE_LOG_USER_CREATE	10	POPSTORE_user_log *
POPSTORE_LOG_USER_DELETE	11	POPSTORE_user_log *
POPSTORE_LOG_USER_MODIFY	12	POPSTORE_user_log *

By default, the site-supplied logging routine is called for every event. This corresponds to a setting of `LOG_ACTIVITY_MASK=-1`. Each bit in the value of `LOG_ACTIVITY_MASK` indicates whether a particular event should be logged (and hence when it has a value of `-1` all events are logged). You can therefore control which events are logged by specifying a non-default value for the `LOG_ACTIVITY_MASK` option as shown in Table 13–2 below. In that table, bit 0 is the lowest (first) bit.

Table 13–2 LOG_ACTIVITY_MASK Bit Values

Symbolic name	Bit to set
`POPSTORE_LOG_START`	Always logged
`POPSTORE_LOG_END`	Always logged
`POPSTORE_LOG_LOGIN_START`	0
`POPSTORE_LOG_LOGIN_PW_MATCH`	1
`POPSTORE_LOG_LOGIN_PW_NOMATCH`	2
`POPSTORE_LOG_LOGIN_END`	3
`POPSTORE_LOG_MSG_STORE`	4
`POPSTORE_LOG_MSG_OPEN`	5
`POPSTORE_LOG_MSG_DELETE`	6
`POPSTORE_LOG_USER_CREATE`	7
`POPSTORE_LOG_USER_DELETE`	8
`POPSTORE_LOG_USER_MODIFY`	9

Each of these logging data types are described below.

POPSTORE_LOG_START

When `log_type` has the value `POPSTORE_LOG_START`, `log_data` is a pointer to a character string.

This logging type arises when the `POPSTORE_init()` subroutine is called and signifies the initialization of the popstore API by a popstore client program. `log_data` points to a string identifying the client program and information about it. Standard popstore subroutines provide the following usage strings:

00:cgi:*user@server-host* (*server-ip-addr+server-tcp-port*):
client@*client-host* (*client-ip-addr+client-tcp-port*)
The indicated user on the indicated host is running the popstore's management HTTP CGI and is processing a request from the indicated host.

01:user_cgi:*user@server-host* (*server-ip-addr+server-tcp-port*):
client@*client-host* (*client-ip-addr+client-tcp-port*)
The indicated user on the indicated host is running the popstore's user-mode informational HTTP CGI and is processing a request from the indicated host.

02:master:*user@host*:*channel*
The indicated user on the indicated host is running the inbound popstore delivery channel under the indicated channel name, *channel*.

03:pop3::
This indicates that the POP3 server has initialized the popstore. At this point, the POP3 server does not have useful information to record. However, as it processes connections

from POP3 clients, it will provide more detailed information. See the description of the `POPSTORE_LOG_LOGIN_START` logging type.

04:popmgr:*user@host*
The indicated user on the indicated host is running the command line management utility. The username is the user's login username on the host and not their popstore username.

05:poppassd::
This indicates that the poppassd server has initialized the popstore. At this point, the poppassd server does not have useful information to record. However, as it processes connections from POP3 clients, it will provide more detailed information. See the description of the `POPSTORE_LOG_LOGIN_START` logging type.

06:return:*user@host*:*channel*
The indicated user on the indicated host is running the popstore message bouncer under the indicated channel name, *channel*.

07:mbxmove:*user@host*
The indicated user on the indicated host is running the mail box migration utility to migrate one or more mail boxes to the popstore.

08:pwd_cgi:*user@server-host* (*server-ip-addr+server-tcp-port*):
client@*client-host* (*client-ip-addr+client-tcp-port*)
The indicated user on the indicated host is running the popstore's user-mode password HTTP CGI and is processing a request from the indicated host.

Site-supplied and third-party popstore clients can use other usage strings. Usage strings beginning with "00" through "20" are reserved for use by Process Software.

POPSTORE_LOG_END

When `log_type` has the value `POPSTORE_LOG_END`, `log_data` has the value `NULL` and `log_len` has the value `0`.

This logging type arises when the subroutine `POPSTORE_end()` is called and signifies the end of use of the popstore by a popstore client program.

POPSTORE_LOG_LOGIN_START

When `log_type` has the value `POPSTORE_LOG_LOGIN_START`, `log_data` is a pointer to a `POPSTORE_user_log` structure. See Section 13.2.4 for a description of that structure.

This logging type arises when the subroutine `POPSTORE_user_begin()` is called to begin a user context. The fields of the logging data will be as follows:

Field name	Description
`stat`	Set to the value `POPSTORE_SUCCESS`
`profile`	Set to the value `NULL`

Field name	Description
`username`	As supplied by the caller of `POPSTORE_user_begin()`
`ulen`	As supplied by the caller of `POPSTORE_user_begin()`
`usage`	As supplied by the caller of `POPSTORE_user_begin()`
`usagelen`	As supplied by the caller of `POPSTORE_user_begin()`

The caller supplied fields should be treated as suspect. For instance, the ulen field can have an incorrect value. Standard popstore subroutines provide the following usage strings

00:cgi:*user*@*server-host* (*server-ip-addr*+*server-tcp-port*):
client@*client-host* (*client-ip-addr*+*client-tcp-port*)
The popstore's management HTTP CGI running on the host *server-host* is processing a request from the indicated host, *client-host*.

01:user_cgi:*user*@*server-host* (*server-ip-addr*+*server-tcp-port*):
client@*client-host* (*client-ip-addr*+*client-tcp-port*)
The popstore's user-mode HTTP CGI running on the host *server-host* is processing a request from the indicated host, *client-host*.

03:pop3:*server-ip-addr*+*server-tcp-port*:*client-ip-addr*+*client-tcp-port*
A POP3 client running on the host with IP address *client-ip-addr* has made a connection to the POP3 server at the IP address *server-ip-addr*.

05:poppassd:*user*@*server-host* (*server-ip-addr*+*server-tcp-port*):
client@*client-host*(*client-ip-addr*+*client-tcp-port*)
A POP3 client running on the host *client-host* has made a connection to the poppassd server on the host *server-host*.

08:pwd_cgi:*user*@*server-host* (*server-ip-addr*+*server-tcp-port*):
client@*client-host* (*client-ip-addr*+*client-tcp-port*)
The popstore's user-mode password HTTP CGI running on the host *server-host* is processing a request from the indicated host, *client-host*.

This logging type also arises when an error is encountered within the API subroutine `POPSTORE_user_begin()`. In that case, fields of the logging data will be as shown below:

Field name	Description
`stat`	Set to the popstore error code value indicating the error which occurred
`profile`	Set to the value `NULL`
`username`	As supplied by the caller of `POPSTORE_user_begin()`
`ulen`	As supplied by the caller of `POPSTORE_user_begin()`

Field name	Description
`usage`	Set to the value `NULL`
`usagelen`	Set to the value `0`

POPSTORE_LOG_LOGIN_PW_MATCH
POPSTORE_LOG_LOGIN_PW_NOMATCH

When `log_type` has the either the value `POPSTORE_LOG_LOGIN_PW_MATCH` or the value `POPSTORE_LOG_LOGIN_PW_NOMATCH`, `log_data` is a pointer to a `POPSTORE_user_log` structure. See Section 13.2.4 for a description of that structure.

This logging type arises when the subroutine `POPSTORE_user_pw_check()` has been called to validate a popstore username and password pair. The first type occurs when the password or challenge response is correct; the second type when the password or challenge response is incorrect.

The logging data, which is of type `POPSTORE_user_log`, will have the field values indicated below:

Field name	Description
`stat`	Return value which will be returned by `POPSTORE_user_pw_check()` to the caller
`profile`	Pointer to the popstore user context
`username`	Username associated with the user context
`ulen`	Length of the username associated with the popstore user context
`usage`	Set to the value `NULL`
`usagelen`	Set to the value `0`

POPSTORE_LOG_LOGIN_END

When `log_type` has the value `POPSTORE_LOG_LOGIN_END`, `log_data` is a pointer to a `POPSTORE_user_log` structure. See Section 13.2.4 for a description of that structure.

This logging type arises when the subroutine `POPSTORE_user_end()` has been called to end a popstore user context. The fields of the logging data will be as follows:

Field name	Description
`stat`	Return value which will be returned by `POPSTORE_user_end()` to the caller
`profile`	Pointer to the popstore user context being ended
`username`	Username associated with the user context
`ulen`	Length of the username associated with the popstore user context
`usage`	Set to the value `NULL`
`usagelen`	Set to the value `0`

POPSTORE_LOG_MSG_STORE

When `log_type` has the value `POPSTORE_LOG_MSG_STORE`, `log_data` is a pointer to a `POPSTORE_message_store_log` structure. See Section 13.2.6 for a description of that structure.

This logging type arises when the inbound message delivery subroutine stores a message into the popstore. The contents of the logging data are as described in Section 13.2.6.

POPSTORE_LOG_MSG_OPEN

When `log_type` has the value `POPSTORE_LOG_MSG_OPEN`, `log_data` is a pointer to a `POPSTORE_message_log` structure. See Section 13.2.5 for a description of that structure.

This logging type arises when a stored message file is accessed with the API subroutine `POPSTORE_message_begin()`. The contents of the logging data are as described in Section 13.2.5.

POPSTORE_LOG_MSG_DELETE

When `log_type` has the value `POPSTORE_LOG_MSG_DELETE`, `log_data` is a pointer to a `POPSTORE_message_log` structure. See Section 13.2.5 for a description of that structure.

This logging type arises when a stored message file is deleted for one of its recipients. The contents of the logging data are as described in Section 13.2.5 with the exception that the four envelope and message id fields will have the values `NULL` and `0`.

POPSTORE_LOG_USER_CREATE

When `log_type` has the value `POPSTORE_LOG_USER_CREATE`, `log_data` is a pointer to a `POPSTORE_user_log` structure. See Section 13.2.4 for a description of that structure.

This logging type arises when the subroutine `POPSTORE_user_add()` has been called to create a new user account. The fields of the log data will be as follows:

Field name	Description
`stat`	Return value which will be returned by `POPSTORE_user_add()` to the caller
`profile`	Pointer to the user context describing the new account
`username`	Username associated with the user context
`ulen`	Length of the username associated with the user context
`usage`	Set to the value `NULL`
`usagelen`	Set to the value `0`

POPSTORE_LOG_USER_DELETE

When `log_type` has the value `POPSTORE_LOG_USER_DELETE`, `log_data` is a pointer to a `POPSTORE_user_log` structure. See Section 13.2.4 for a description of that structure.

This logging type arises when the subroutine `POPSTORE_user_delete()` has been called to delete a user account. The fields of the log data will be as follows:

Field name	Description
`stat`	Return value which will be returned by `POPSTORE_user_add()` to the caller

Field name	Description
`profile`	Pointer to the user context describing the account being deleted
`username`	Username associated with the user context
`ulen`	Length of the username associated with the user context
`usage`	Set to the value `NULL`
`usagelen`	Set to the value `0`

POPSTORE_LOG_USER_MODIFY

When `log_type` has the value `POPSTORE_LOG_USER_MODIFY`, `log_data` is a pointer to a `POPSTORE_user_log` structure. See Section 13.2.4 for a description of that structure.

This logging type arises when the subroutine `POPSTORE_user_update()` has been called to modify a user account. The fields of the logging data will be as follows:

Field name	Description
`stat`	Return value which will be returned by `POPSTORE_user_update()` to the caller
`profile`	Pointer to an array of two user contexts: the first context describes the original settings for the account, the second context describes the new settings for the account
`username`	Username associated with the user context
`ulen`	Length of the username associated with the user context
`usage`	Set to the value `NULL`
`usagelen`	Set to the value `0`

13.2.3 Logging Samples

The following samples illustrate some of the logging possibilities realized through the logging interface. In these examples, the site-supplied `log_activity` subroutine outputs records containing the `log_id` and `log_subid` values separated by a dot and then followed by the date and time. That is then followed by the value of the `log_type` parameter which is then followed by information pertinent to the type of data being logged.

In the first example, Example 13–1, the logging samples correspond to the UNIX login user `bob` issuing the commands

```
# pmdf popstore
popstore> modify jdoe -quota=20480
popstore> quit
```

Example 13–1 Logging of a user profile modification

```
X9TC0008T6.1 11:30:11 - POPSTORE_LOG_START:
X9TC0008T6.1 11:30:11 -    usage = 04:popmgr:bob@gate.plastic.com
X9TC0008T6.1 11:30:15 - POPSTORE_LOG_USER_MODIFY: user="jdoe"
X9TC0008T6.1 11:30:15 -   quota:10240 -> 20480
X9TC0008T6.1 11:30:15 - POPSTORE_LOG_END
```

Example 13–2 Logging of a POP3 client downloading and deleting a message

```
27I80009GF.0 11:30:16 - POPSTORE_LOG_START:
27I80009GF.0 11:30:16 -   usage = "03:pop3::"
27I80009GF.0 11:31:45 - POPSTORE_LOG_LOGIN_START:
27I80009GF.0 11:31:45 -   usage = "03:pop3:192.160.0.1+14080:192.160.0.5+12839"
27I80009GF.0 11:31:45 -   user  = "jdoe"
27I80009GF.0 11:31:45 - POPSTORE_LOG_LOGIN_PW_MATCH: user="jdoe"
27I80009GF.0 11:32:20 - POPSTORE_LOG_MSG_OPEN
27I80009GF.0 11:32:20 -   UIDL        = !!!"01IDYVUZOFEO0008IH0
27I80009GF.0 11:32:20 -   envelope ID = 01IDYTN79MU2000882@gate.plastic.com
27I80009GF.0 11:32:20 -   message ID  = <01IDEQPFAOH@foo.albany.edu>
27I80009GF.0 11:32:21 - POPSTORE_LOG_MSG_DELETE:
27I80009GF.0 11:32:21 -   user        = "jdoe"
27I80009GF.0 11:32:21 -   UIDL        = !!!"01IDYVUZOFEO0008IH0
27I80009GF.0 11:32:21 - POPSTORE_LOG_LOGIN_END: user="jdoe"
27I80009GF.0 11:32:22 - POPSTORE_LOG_END
```

Example 13–3 Logging of the storage of a message

```
GMU8000B5V.0 16:12:39 - POPSTORE_LOG_START:
GMU8000B5V.0 16:12:39 -   usage = 02:master:SYSTEM@gate.plastic.com:popstore
GMU8000B5V.0 16:12:43 - 2 recipient, 2218 byte message from a@example.com stored
GMU8000B5V.0 16:12:43 -   envelope ID = 01IDZ5RYTUKU000882@gate.plastic.com
GMU8000B5V.0 16:12:43 -   message ID  = <97008157_132305@emout1.mail.aol.com>
GMU8000B5V.0 16:12:43 -   filename    = 01IDZ5S85JTU000B5V0
GMU8000B5V.0 16:12:43 -   1 !!!! asmith
GMU8000B5V.0 16:12:43 -   1 !!!" jdoe
GMU8000B5V.0 16:12:45 - POPSTORE_LOG_END
```

The next example, Example 13–2, corresponds to the popstore user `jdoe` reading a mail message and then deleting it with their POP3 client.

The final example, Example 13–3, shows the inbound delivery agent delivering a message for two recipients to the popstore.

13.2.4 POPSTORE_user_log Structure

The POPSTORE_user_log structure is used to log events associated with the processing of user accounts. The C language declaration of the structure is provided by the popstore.h header file and shown below:

```
typedef struct {
  int32               stat;
  POPSTORE_user_data *profile;
  char               *username;
  int32               ulen;
  char               *usage;
  int32               usagelen;
} POPSTORE_user_log;
```

The usage of the fields in this structure varies with the value of the log_type parameter. See Section 13.2.2 for details. Note that the username and usage strings can not be NULL terminated.

13.2.5 POPSTORE_message_log Structure

The POPSTORE_message_log structure is used to log events associated with the processing of stored message files. The C language declaration of the structure is provided by the popstore.h header file and repeated below:

```
typedef struct {
  char  *user;
  int32  user_len;
  char  *uidl;
  int32  uidl_len;
  char  *env_id;
  int32  env_id_len;
  char  *msg_id;
  int32  msg_id_len;
} POPSTORE_message_log;
```

The interpretation of these fields are as follows:

user
user_len
The username and length in bytes of the username associated with the popstore account for which the message operation is being performed. This information can be omitted in which case the fields will have the values NULL and 0. The username string can not be NULL terminated.

uidl
uidl_len
UIDL and length in bytes of the UIDL referencing the message. The UIDL string can not be NULL terminated.

env_id
env_id_len
The envelope identification and length in bytes of the envelope identification associated with the message. This information can not be provided, in which case the fields will have the values NULL and 0. The envelope identification string can not be NULL terminated.

msg_id
msg_id_len
The value of the message's RFC822 message-id header line and length in bytes of that value. This information can not be provided, in which case the fields will have the values NULL and 0. The message-id string can not be NULL terminated.

13.2.6 POPSTORE_message_store_log Structure

The POPSTORE_message_store_log structure is used to log the delivery a message to the message store. The C language declaration of the structure is provided by the popstore.h header file and repeated below:

```
typedef struct {
  uint32                   version;
  uint32                   size;
  time_t                   creation_date;
  uint32                   recipient_count;
  uint32                   reference_count;
  char                    *env_from;
  int32                    env_from_len;
  char                    *env_id;
  int32                    env_id_len;
  char                    *msg_id;
  int32                    msg_id_len;
  char                    *filename;
  int32                    filename_len;
  char                     channel[40];
  POPSTORE_recipient_list *users;
} POPSTORE_message_store_log;
```

The interpretation of these fields are as follows:

version
Message file format version used for the message file.

size
Length in bytes of the stored message content.

creation_date
Creation date and time for the message file as measured in seconds since 0:00:00.00 on 1 January 1970.

recipient_count
Count of popstore envelope recipients for the message.

reference_count
Count of active envelope recipients for the message; that is, the number of popstore accounts with references to the message. This number will be less than or equal to the recipient_count. Usually it will be equal to the recipient_count, but it can be less if some sort of problem arose making delivery to a recipient impossible.

env_from
env_from_len
The message's envelope From: address and length of that address. The env_from field need not be NULL terminated.

env_id
env_id_len
The message's envelope id and length of that id. The env_id field need not be NULL terminated.

msg_id
msg_id_len
The message's RFC822 message-id and length of that id. The msg_id field need not be NULL terminated.

filename
filename_len
The name of the file in which the message is stored. The filename field need not be NULL terminated.

channel
The name of the channel which delivered this message. The channel field is space (0x20 hex) padded and is not NULL terminated.

users
Pointer to an array of POPSTORE_recipient_list structures. The number of entries in the array is given by the reference_count field. The POPSTORE_recipient_list structure is described in Section 13.2.7.

13.2.7 POPSTORE_recipient_list Structure

The POPSTORE_recipient_list structure is used to log the envelope recipient list for a message stored in the popstore. The C language declaration of the structure is provided by the popstore.h header file and repeated below:

```
typedef struct {
  char  *user;
  int    user_len;
  char  uidl[4];
  int    status;
} POPSTORE_recipient_list;
```

The interpretation of these fields are as follows:

`user`
`user_len`
Popstore username and length of that username. The `username` field can not be `NULL` terminated.

`uidl`
The first four characters of the UIDL for this user's instance of the message.

`status`
Status of this recipient:

Value	Interpretation
1	Message was successfully delivered to this recipient.
2	Message was not delivered to this recipient; temporary error; will retry later.
3	Message was not delivered to this recipient; no such recipient.
4	Message was not delivered to this recipient; recipient is over quota.
5	Message was not delivered to this recipient; recipient is marked DISMAIL.

14 Miscellaneous Subroutines

Sites can override five aspects of the popstore and MessageStore by providing callable subroutines:

- the algorithm used to compute elapsed connect time,
- the algorithm used to compute past block days, a measure of disk space usage over time,
- the location in the file system of user profile files,
- the location in the file system of stored message files, and
- the reasonableness checks done on the new password during a password change operation.

The first two subroutines are only of interest to sites which bill users for connect time or disk storage and want to change how the popstore and MessageStore computes those quantities; see Section 14.1 for details. The next two subroutines are of interest to sites who want to spread the popstore/msgstore user or message files across multiple disks as described in Section 14.2.[1]

The last subroutine is of interest to sites who want to increase the security of popstore/msgstore passwords for example by comparing the proposed password to a dictionary or to a site-maintained history of previous passwords. See Section 14.3.

14.1 Computation Subroutines

By default, the popstore/msgstore uses straightforward algorithms to compute elapsed connect times and disk storage over time:

```
elapsed_time := end_time - start_time
storage := (end_time - start_time) / (86400 seconds/hour) *
                size / (1024 bytes/block)
```

where the quantities above have the following interpretations:

elapsed_time
Elapsed time measured in units of seconds.

end_time
Ending time measured in units of seconds elapsed since 1 January 1970 at 0:00:00.0.

start_time
Starting time measured in units of seconds elapsed since 1 January 1970 at 0:00:00.0.

[1] Note that on UNIX systems, this can be done using symbolic links; the popstore/msgstore will correctly follow hard and symbolic links.

storage
Amount of storage as measured in units of block days with 1 block equal to 1024 bytes.

size
Size of the stored object as measured in units of bytes.

Sites wanting to use different algorithms can do so by supplying executable subroutines via shared images. The subroutine to compute the elapsed time must have the name `compute_connect` and be of the form

```
#include <time.h>
#ifdef __VMS
#  include "pmdf_com:popstore.h"
#else
#  include "/pmdf/include/popstore.h"
#endif

uint32 compute_connect (start_time, end_time)
  time_t start_time;
  time_t end_time;
```

The calling arguments for `compute_connect` are as described below and the subroutine must return the elapsed time, as measured in seconds, between the starting and ending times:

start_time
Starting time measured in units of seconds elapsed since 1 January 1970 at 0:00:00.0. Used for input only.

end_time
Ending time measured in units of seconds elapsed since 1 January 1970 at 0:00:00.0. Used for input only.

A C code realization of the default algorithm used by the popstore/msgstore to compute connect time is given in Example 14–1.

Example 14–1 Default `compute_connect` Subroutine

```
#include <time.h>
#ifdef __VMS
#  include "pmdf_com:popstore.h"
#else
#  include "/pmdf/include/popstore.h"
#endif

uint32 compute_connect (start_time, end_time)
  time_t start_time;
  time_t end_time;
{
  return ((uint32)difftime (end_time, start_time));
}
```

The subroutine to compute storage must have the name `compute_block_days` and be of the form

```
#include <time.h>
#ifdef __VMS
#  include "pmdf_com:popstore.h"
#else
#  include "/pmdf/include/popstore.h"
#endif

void compute_block_days (start_time, end_time, size, result,
                         remainder)
  time_t  start_time;
  time_t  end_time;
  uint32  size;
  uint32 *result;
  uint32 *remainder;
```

The input arguments to the subroutine are described below. On output, the subroutine must return in `*result` the storage as measured in units of block days. In addition, the subroutine must return in `*remainder` any roundoff, as measured in byte seconds.

start_time
Starting time measured in units of seconds elapsed since 1 January 1970 at 0:00:00.0. Used for input only.

end_time
Ending time measured in units of seconds elapsed since 1 January 1970 at 0:00:00.0. Used for input only.

result
Amount of storage as measured in units of block days with 1 block equal to 1024 bytes. Used for output only.

remainder
Roundoff in storage as measured in units of byte seconds. Used for input and output. On input, this will be the roundoff left over from a previous computation and which should be incorporated into this new computation. On output, this should be set to the roundoff resulting from computing the result.

The default subroutine used by the popstore/msgstore is shown in Example 14–2.

The `COMPUTE_CONNECT` and `COMPUTE_BLOCK_DAYS` options must be used to point the popstore/msgstore at the shared image or images containing the `compute_connect` and `compute_block_days` subroutines. See Section 3.4 for details. When linking the subroutines into shared images, use link commands of the forms shown in Section 13.2.1. The subroutines can be tested with the `TEST` command of the command line management utility.

Example 14–2 Default `compute_block_days` Subroutine

```
#include <time.h>
#ifdef __VMS
#  include "pmdf_com:popstore.h"
#else
#  include "/pmdf/include/popstore.h"
#endif

void compute_block_days (start_time, end_time, size, result,
                         remainder)
  time_t  start_time;
  time_t  end_time;
  uint32  size;
  uint32 *result;
  uint32 *remainder;
{
  double blocks, days, value;

  days       = difftime (end_time, start_time) / 86400.0;
  blocks     = size / 1024.0;
  value      = (days * blocks) + (*remainder / (86400.0 * 1024.0));
  *result    = (uint32)value;
  *remainder = (uint32)((value - (uint32)value) * 86400.0 * 1024.0);
}
```

14.2 File Locations

By default, the popstore/msgstore stores all message files on the same disk as described in Section 1.4. Likewise for account profile files as described in Section 1.3.9. On UNIX systems, links — symbolic or hard — can be used to relocate subdirectories in these trees thereby allowing files to be spread across any number of disks. The popstore/msgstore will automatically handle such links; no special configuration of the popstore/msgstore is required to accomodate links.

Sites wanting to relocate the entire message or profile directory tree to another disk should do so using either the UNIX PMDF tailor file, the NT registry, or the OpenVMS popstore/msgstore logicals. On UNIX and NT systems, manually move the tree in question and change the corresponding `PMDF_POPSTORE_MESSAGES` or `PMDF_POPSTORE_PROFILES` entry in the `/etc/pmdf_tailor` file (UNIX) or registry (NT). On OpenVMS systems, manually move the directory tree in question and then change the corresponding definition of the `PMDF_POPSTORE_MESSAGES` or `PMDF_POPSTORE_PROFILES` logical. Changing the values of those logicals is best done by first seeing how they are defined in the `pmdf_startup.com` procedure in the `SYS$STARTUP:` directory. Then, create a `pmdf_site_startup.com` command procedure which redefines the logicals, pointing to the correct disk and directory. Place that command procedure in the `PMDF_COM:` directory. It will then be seen and executed by PMDF each time you boot your system.

Now, some sites can want to actually spread message or profile files across more than one disk. Again, UNIX systems can use symbolic links to accomplish this. However, OpenVMS systems lack such a mechanism. Consequently, an alternate mechanism for relocating files is provided for all platforms. This mechanism involves the use of site-supplied subroutines made available to the popstore/msgstore via shared images. When the popstore/msgstore needs to access a message or a profile file, it first generates a default path to the file and then checks for a site-supplied subroutine. If the subroutine does not exist, the default path is used. If the subroutine does exist, it is called with the file path and filename in question. The subroutine can then change the path to the file; the popstore/msgstore will use the changed path to access the file. In addition to supplying subroutines, text files listing the path to each directory tree must also be supplied. These text files are used when the popstore/msgstore must search for a message or profile file.

Note: Only use this mechanism when actually spreading a profile or message file directory tree across more than one disk. Using this mechanism when merely moving an entire profile or message file directory tree to another disk is not worth the effort required and can be accomplished more easily using previously described mechanisms.

The name of the subroutine to map message file filenames is `map_message_filename`; the name of the subroutine to map account profile filenames is `map_profile_filename`. Their existence and location are made known to the popstore/msgstore using the MAP_ MESSAGE_FILENAME and MAP_PROFILE_FILENAME options documented in Sections 3.4 and 3.3, respectively. Sites can supply one or both of these subroutines. When linking the subroutines into shared images, use link commands of the forms given in Section 13.2.1.

The subroutines take the form

```
#ifdef __VMS
#  include "pmdf_com:popstore.h"
#else
#  include "/pmdf/include/popstore.h"
#endif

void map_message_filename (version, def_name, def_len, def_path_len,
                           new_name, new_len, max_new_len, new_path_len)
  uint32 version
  char   *def_name;
  int     def_len;
  int     def_path_len;
  char   *new_name;
  int    *new_len;
  int     max_new_len;
  int    *new_path_len;

void map_profile_filename (version, def_name, def_len, def_path_len,
                           new_name, new_len, max_new_len, new_path_len)
  uint32 version
  char   *def_name;
  int     def_len;
  int     def_path_len;
  char   *new_name;
  int    *new_len;
  int     max_new_len;
  int    *new_path_len;
```

The input and output arguments to the subroutines are as follows:

version
For map_message_filename, this argument is the current value of the MESSAGE_FILENAME_VERSION option. For map_profile_filename, this argument is the current value of the PROFILE_FILENAME_VERSION option. Used for input only.

def_name
The default file specification including the full path and filename. This string is NULL terminated. Used for input only.

def_len
The length in bytes of the default file specification. The length does not include any NULL terminator. Used for input only.

def_path_len
The length in bytes of the path specification in the default file specification. That is, the first def_path_len bytes of def_name is the path specification for the file. Used for input only.

new_name
The new file specification which should be used. This must include both the full path and filename for the file and must be NULL terminated. Note that the filename portion of the file specification must not be altered; only the path specification portion of the file specification can be changed. The default directory name in the path should not be removed—it is part of the popstore's user domain naming system. default is the default user domain. Used for output only.

new_len
The length in bytes, not including the NULL terminator, of the new file specification. Used for output only.

max_new_len
Maximum length in bytes of new_name, not including a trailing NULL terminator. The name passed back in new_name can not exceed this length. Used for input only. Note that the following relationship will always hold:
def_path_len <= 252 bytes <= max_new_len

new_path_len
The length in bytes of the path specification portion of the new file specification. Used for output only.

The site-supplied subroutine is passed the default file specification and must return on output a new file specification. If no change is to be made to the file specification, then the input arguments should simply be copied to the output arguments. If a change is made, the resulting file specification *must* have the same filename. The subroutine can only change the path portion of the file specification. (On OpenVMS systems, this is the device and directory portion of the specification.) Moreover, for profile file names, the default directory name should be left untouched. That name is part of the popstore's user domain naming system.

Note that for message filenames, the last character in the filename portion of the specification indicates the value of the MESSAGE_FILENAME_VERSION option which was in force when the message file in question was initially created. The default value for that option is 0. When implementing a map_message_filename subroutine, it is a good idea to increment that value to 1. Then your subroutine can distinguish between files generated when no algorithm was in place and files generated when an algorithm was first used. Yes, there's no immediate value in doing this. However, should you then change your algorithm, you will then want to distinguish between your new algorithm and your old algorithm. When you change your algorithm, then again increment the version number. You can then distinguish between the no algorithm case, version `0`, the original algorithm case, version `1`, and the revised algorithm case, version `2`.

The `command line management` utility includes a `TEST` command which should be used to test `map_message_filename` and `map_profile_filename` subroutines. Use that utility to test your subroutines before integrating them into the popstore with the `MAP_MESSAGE_FILENAME` and `MAP_PROFILE_FILENAME` options. Moreover, when you use that `TEST` command, the utility will tell you how each message or profile filename would be mapped by your subroutine. That information is provided for each message or user account currently used by the popstore. You can use that information to help relocate each popstore file to the correct new location required by your subroutine.

When a `map_message_filename` subroutine is provided, you must also supply a corresponding `popstore_message_paths` file. That file is a world readable text file which should be placed in the PMDF table directory. For each top-level directory tree used to store message files, a path to that tree must be given in the text file. One path per line in the file. Similarly, when a `map_profile_filename` subroutine is provided, a corresponding `popstore_profile_paths` file must be provided in the PMDF table directory. These files are then used by the popstore when it must conduct scans for message or profile files. Scans for profile files are only done to rebuild the user database or to test a `map_profile_filename` subroutine. However, scans for message files happen regularly when the popstore's message bouncer runs. An example file is given in Example 14–4.

A sample `map_profile_filename` is shown in Example 14–3. That sample subroutine implements, for UNIX systems, a directory tree split as depicted below:

Default filename	**New filename**
/pmdf/user/default/a/*	/disk0/profiles/default/a/*
...	...
/pmdf/user/default/m/*	/disk0/profiles/default/m/*
/pmdf/user/default/n/*	/disk1/profiles/default/n/*
...	...
/pmdf/user/default/z/*	/disk1/profiles/default/z/*
/pmdf/user/default/0/*	/disk2/profiles/default/0/*
...	...
/pmdf/user/default/9/*	/disk2/profiles/default/9/*

For example, the profile file normally stored in `/pmdf/user/default/r/o/b/rob` is relocated to `/disk1/profiles/default/r/o/b/rob`. The `popstore_profile_paths` file corresponding to this mapping is shown in Example 14–4.

Example 14–3 UNIX `map_profile_filename` Sample Subroutine

```
#include <string.h>
#include "/pmdf/include/popstore.h"

void map_profile_filename (uint32 version, char *def_name,
                           int def_len, int def_path_len,
                           char *new_name, int *new_len,
                           int max_new_len, int *new_path_len)
{
  char c;
  /*
   *  We assume that def_name will be of the form
   *  /pmdf/user/domain/x/y/z/filename.  This assumption is
   *  governed by the value of the option PMDF_POPSTORE_PROFILES
   *  in the /etc/pmdf_tailor file.
   */
  strcpy (new_name, "/disk#/profiles/");
  c = def_name[def_path_len-6];
  if ('a' <= c && c <= 'm') new_name[5] = '0';
  else if ('n' <= c && c <= 'z') new_name[5] = '1';
  else new_name[5] = '2';
  strcat (&new_name[16], &def_name[11]);
  *new_path_len = def_path_len + 5;
  *new_len = strlen (new_name);
}
```

Example 14–4 UNIX `/pmdf/table/popstore_profile_paths` Sample File

```
/disk0/profiles/
/disk1/profiles/
/disk2/profiles/
```

A similar example for OpenVMS systems is shown in Examples 14–5 and 14–6. That sample subroutine implements, the directory tree split as depicted below:

Default filename	New filename
PMDF_POPSTORE_PROFILES:[DEFAULT.A...]*.;	DISK0:[PROFILES.DEFAULT.A...]*.;
...	...
PMDF_POPSTORE_PROFILES:[DEFAULT.M...]*.;	DISK0:[PROFILES.DEFAULT.M...]*.;
PMDF_POPSTORE_PROFILES:[DEFAULT.N...]*.;	DISK1:[PROFILES.DEFAULT.N...]*.;
...	...
PMDF_POPSTORE_PROFILES:[DEFAULT.Z...]*.;	DISK1:[PROFILES.DEFAULT.Z...]*.;
PMDF_POPSTORE_PROFILES:[DEFAULT.0...]*.;	DISK2:[PROFILES.DEFAULT.0...]*.;

Default filename	New filename
...	...
PMDF_POPSTORE_PROFILES:[DEFAULT.9...]*.;	DISK2:[PROFILES.DEFAULT.9...]*.;

For example, the profile file normally stored in

```
PMDF_POPSTORE_PROFILES:[DEFAULT.R.O.B]rob.;
```

is relocated to

```
DISK1:[PROFILES.DEFAULT.R.O.B]rob.
```

The `popstore_profile_paths` file corresponding to this mapping is shown in Example 14–6.

Example 14–5 OpenVMS `map_profile_filename` Sample Subroutine

```
#include <string.h>
#include "pmdf_com:popstore.h"

void map_profile_filename (uint32 version, char *def_name,
                           int def_len, int def_path_len,
                           char *new_name, int *new_len,
                           int max_new_len, int *new_path_len)
{
  char c;
   *  We assume that def_name will be of the form
   *  PMDF_POPSTORE_PROFILES:[DEFAULT.X.Y.Z]filename.
   */
  strcpy (new_name, "DISK#:[PROFILES.DEFAULT.X.Y.Z]");
  c = def_name[def_path_len-6];
  if ('a' <= c && c <= 'm') new_name[4] = '0';
  else if ('n' <= c && c <= 'z') new_name[4] = '1';
  else new_name[4] = '2';
  strcat (&new_name[16], &def_name[24]);
  *new_path_len = def_path_len + 8;
  *new_len = strlen (new_name);
}
```

Example 14–6 OpenVMS `PMDF_TABLE:popstore_profile_paths.` Sample File

```
DISK0:[PROFILES...]*.;*
DISK1:[PROFILES...]*.;*
DISK2:[PROFILES...]*.;*
```

14.3 Subroutine To Validate A Password

The popstore/msgstore has support for certain reasonableness checks on a proposed password when a user or administrator attempts to change an account's password. The `validate_password` subroutine can be specified by sites who wish to add additional validation, such as a dictionary or history check.

The `validate_password` subroutine's existence and location are made known to the popstore/msgstore using the `VALIDATE_PASSWORD` option documented in Section 3.3. When linking this subroutine into a shared image, use link commands of the forms given in Section 13.2.1.

The subroutine takes the form

```
#ifdef __VMS
#include "pmdf_com:popstore.h"
#else
#include "/pmdf/include/popstore.h"
#endif

int validate_password (password, password_len, username, username_len,
                       errmsg, errmsg_len, errmsg_max)
    char *password;
    int password_len;
    char *username;
    int username_len;
    char *errmsg;
    int *errmsg_len;
    int errmsg_max;
```

The input and output arguments to the subroutines are as follows:

password
The new password. Used for input only.

password_len
The length in bytes of the new password, not including any `NULL` terminator. Used for input only.

username
The username of the account whose password is being changed. Used for input only.

username_len
The length in bytes of the username, not including any `NULL` terminator. Used for input only.

errmsg
Address of a character array that `validate_password` should put an error message into if the password validation fails. The length of the message must not be longer than `errmsg_max` bytes. Used for output only.

errmsg_len
The length in bytes of the error message that `validate_password` put into the `errmsg` array. Set this to 0 if no error message was placed there. Used for output only.

errmsg_max
The size in bytes of the character array that `errmsg` points to. Used for input only.

The validate_password subroutine should return one of the following statuses:

POPSTORE_SUCCESS
If the password passed all validation checks and was accepted.

POPSTORE_PWDNOTOK
If the password failed a validation check and was rejected.

A sample `validate_password` routine is shown in Example 14–7 and can be found at, on OpenVMS:

```
PMDF_ROOT:[DOC.EXAMPLES]POPSTORE_VALIDATE_PASSWORD.C
```

And on UNIX:

```
/pmdf/doc/examples/popstore_validate_password.c
```

Example 14–7 `validate_password` **Sample Subroutine**

```
#ifdef __VMS
#include "pmdf_com:popstore.h"
#else
#include "/pmdf/include/popstore.h"
#endif
#include <string.h>

int validate_password (char *password, int password_len,
                       char *username, int username_len,
                       char *errmsg, int *errmsg_len, int errmsg_max)
{
#define MSG "Password rejected by validate_password"

    (*errmsg_len) = 0;
    errmsg[0] = '\0';

    /* for example, reject a password of "invalid", otherwise accept */
    if ((password_len == 7) &&
        (password[0] == 'i') &&
        (password[1] == 'n') &&
        (password[2] == 'v') &&
        (password[3] == 'a') &&
        (password[4] == 'l') &&
        (password[5] == 'i') &&
        (password[6] == 'd'))
    {
        (*errmsg_len) = (strlen(MSG)>errmsg_max ? errmsg_max : strlen(MSG));
        strncpy (errmsg, MSG, (*errmsg_len));
        return (POPSTORE_PWDNOTOK);
    }
    else
        return (POPSTORE_SUCCESS);
}
```

Index