OTN OpenMail Technical Note

B.06.00 Changes to the IMAP Server

Summary

The OpenMail IMAP server has been completely overhauled at B.06.00. In this OTN, we look at how existing functionality has been improved, for example, logging and searching, and investigate new features introduced, such as IMAP4rev1 commands and supported optional capabilities, troubleshooting tools and tweaks.

We will also look at the changes made to improve the performance and reliability of the server.

I have tended to focus on the two most popular IMAP clients, Netscape Messenger 4.5/6 and Outlook Express 5, but where appropriate, I have included information about other clients.

This OTN will form part of the OM360 (B.06.00 upgrade) course.

Readership

Readers are assumed to be OpenMail support engineers, who have attended an advanced OpenMail course (AE351 or H1805s) or are familiar with the material covered in these courses.

I have also assumed that readers are familiar with the existing OTN: *IMAP4 Support*.

Revision History

November 1999:	First issue
December 1999:	Comments incorporated

Comments Please!

I would welcome any comments you may have on this document. Please email them to joyce@pwd.hp.com.

Contents

Introduction

The changes to the OpenMail IMAP server at B.06.00 cover several areas:

- Improvements to existing functionality, in particular, logging, searching, message rendering and caching, security and multiple access of mailboxes.
- Support for new IMAP4rev1 functionality, such as the `STATUS` command, `FETCH` command extensions and Unicode support.
- Support for optional IMAP4rev1 functionality, such as `IDLE`, `NAMESPACE` and proprietary `CAPABILITIES`. Two useful OpenMail proprietary extensions are included for test situations.

In the first section, we shall look at how to setup tracing for IMAP connections. You can then use this as you work through the rest of the OTN, to get a better idea of what is happening in the background. I have also included information on a TCL tool, called Watcher, which gives you a useful window on the raw protocol being sent between client and IMAP server.

In the subsequent sections, we shall work through the changes in the order shown above, looking at how the IMAP server interacts with other parts of OpenMail, such as the UAL and item browser. Where we found a client behaves in an unusual or unreliable way, I have included the details. For instance, configuring some clients to be able to view Bulletin Boards is not intuitive, because of client limitations.

For specific client examples, I have tended to use Netscape Messenger 4.6 and Outlook Express 5, because they are the most widely used IMAP clients. For illustrating general IMAP sessions, I have used telnet'd IMAP sessions, i.e. `telnet <host> 143`.

Logging

Significant improvements have been made to the logging options available for IMAP in B.06.00. As we shall be using some of these options throughout this OTN to view new functionality, it is probably a good idea to start by looking at the various ways you can trace client, IMAP server and UAL activity.

The following new tweaks control logging:

- `IMAP_LOGLEVEL` - This controls the level of logging for the IMAP server process(es). Setting a logging level automatically starts logging of the IMAP server and IMAP UAL traffic.
- `IMAP_LOGFILE` - The name of the file(s) to hold IMAP server log messages.
- `IMAP_UAL_TRACE_LEVEL` - This controls the level of IMAP UAL logging.

We'll look at each tweak in more detail and consider some combinations of settings to show how we can use them all to tailor logging.

Note: As we will be looking at tailoring logging at system, client and user level, remember that `~/sys/general.cfg` is read once, when the IMAP daemon starts. `~/sys/client.cfg/*` and `~/sys/user.cfg/*` configuration files are read each time a new connection is made to the daemon.

In the section, "Watcher Connection Monitoring Tool" on page 10, I have documented a useful TCL program, which allows you to monitor IMAP conversations between client and server machines.

IMAP_LOGLEVEL

This option controls the logging of the IMAP server processes, `in.imap41d`. By default, logging of the IMAP server is turned off:

```
IMAP_LOGLEVEL=0
```

The following values (bit settings) define the logging level:

1 Log any unexpected UAL errors.

2 Log the IMAP commands after they have been parsed by the IMAP server (but before any action is taken to process the command) and the final response to the client when the command is complete.

8 Log the original IMAP commands, without the value of any literal strings, before they have been parsed by the IMAP server and all the IMAP server responses, both tagged and untagged. At this logging level, no interpretation is placed on the lines being logged, so the IMAP server cannot suppress information, such as the password in an IMAP `LOGIN` command. However, log files are normally created with permissions that prevent ordinary users from reading them.

These can be combined to give the required output. We will look at the different types of lines created with different log levels set.

All logged messages take the format:

```
<IMAP server process ID> <current time HH:MM:SS> <log message>
```

When very long log messages are split over more than one line, continuation lines begin with white space.

With the exception of the two lines below, the format of the <*`log message`*> part differs, depending on the log level set. The following two messages are logged whenever the log level is greater than zero and show when a particular instance of the IMAP server is starting up or shutting down:

```
IMAP4 Server <version> on <host> at <date>
IMAP4 Server exiting
```

On my server, the lines look like this:

```
9327 15:13:28 IMAP4 Server B.06.00.R0 on kuda at Fri Oct  1 15:13:28 1999
9327 15:13:32 IMAP4 server exiting.
```

When `IMAP_LOGLEVEL=1`, the log message records the text of any UAL error which the IMAP server was not expecting. This doesn't include UAL errors resulting from things like bad IMAP commands. The message format is:

```
UAL: <UAL error message>
```

For example:

```
9472 16:44:15 UAL: error 655 (0,0):
        The password supplied is incorrect
```

When `IMAP_LOGLEVEL=2`, the log messages include the parsed commands from the client and the final response to the client. The format is:

```
CMD: <tag> <parsed command>
RSP: <tag> <final response for the command>
```

For example:

```
9370 15:26:59 CMD: a1 LOGIN "admin" <password>
9370 15:27:03 RSP: a1 OK LOGIN completed, now connected to kuda.pwd.hp.com
9370 15:27:03 CMD: a2 SELECT inbox
9370 15:27:03 RSP: a2 OK [READ-WRITE] SELECT completed
9370 15:27:03 CMD: a3 FETCH 13 (RFC822)
9370 15:27:07 RSP: a3 OK FETCH completed
9370 15:27:07 CMD: a4 LOGOUT
9370 15:27:08 RSP: a4 OK LOGOUT completed
```

Note that the password in the client `LOGIN` line is suppressed by the IMAP server.

If a command could not be parsed, then the `CMD:` line will usually be absent. This line reflects what the IMAP server interpreted the command to mean, not what was actually sent, and in some circumstances the command may not be syntactically correct.

When `IMAP_LOGLEVEL=8`, the log messages record the original, unparsed IMAP commands (apart from the value of any literal strings) and all the IMAP server responses. The lines take the format:

```
C: <literal client request>
R: <literal server response>
```

For example:

```
9407 15:29:19 C: a1 login admin admin
9407 15:29:23 S: a1 OK LOGIN completed, now connected to kuda.pwd.hp.com
9407 15:29:23 C: a2 select inbox
9407 15:29:23 S: * 16 EXISTS
* 0 RECENT
* OK [UNSEEN 16] is the first unread message
* OK [UIDVALIDITY 1] UIDVALIDITY value
* FLAGS (\Answered \Flagged \Deleted \Seen \Draft $MdnSent)
* OK [PERMANENTFLAGS (\Answered \Flagged \Deleted \Seen \Draft $MdnSent)]
flags will stay set
9407 15:29:23 S: a2 OK [READ-WRITE] SELECT completed
9407 15:29:23 C: a3 fetch 13 (rfc822)
9407 15:29:26 S: * 13 FETCH (RFC822 {1663}
9407 15:29:26 S: )
9407 15:29:26 S: a3 OK FETCH completed
9407 15:29:26 C: a4 logout
9407 15:29:27 S: * BYE OpenMail IMAP Server logging out
9407 15:29:27 S: a4 OK LOGOUT completed
```

Note that this time the password in the client `LOGIN` line is not suppressed. Also, the whole message returned as a literal string in response to the command `fetch 13 (rfc822)` is not logged.

Where To Set IMAP_LOGLEVEL

This tweak can be set for the whole system (in `~/sys/general.cfg`) or for a particular client machine (in `~/sys/client.cfg/<`*`client hostname`*`>`) or for a particular OpenMail user (in `~/sys/user.cfg/<`*`OM numeric user ID`*`>`).

The IMAP log file is created when logging is first turned on. However, the level of logging for the system or client can vary from the level for an individual user. So, for example, if you set the log level to 3 on a system-wide or per-client basis, then you could set it to 8 (to record the literal protocol) in the per-user configuration file. Changing log levels like this allows a complete protocol trace to be recorded without actually recording authentication details.

Remember, if you set the tweak in `general.cfg`, you will need to restart the IMAP server for it to take effect. If it is set for a client or user, it is not necessary to restart the IMAP server.

Effect on UAL Tracing

If the tweak, `IMAP_UAL_TRACE_LEVEL`, is not defined, then setting `IMAP_LOGLEVEL` to a non-zero level will automatically set the `IMAP_UAL_TRACE_LEVEL` to 8.

Compatibility With B.05.10

If `IMAP_LOGLEVEL` is not defined and the old B.05.10 option, `UAL_IMAP4_TRACE=TRUE`, then the log level is set to 3 for compatibility. (This will probably be removed in a future release).

IMAP_LOGFILE

If logging has been enabled for the IMAP server, using the `IMAP_LOGLEVEL` option, this tweak defines the name of the log file(s) and the scope of the logging contained in the log file, that is, whether the log file pertains to a particular client, user and/or IMAP process. Substitutions can be used in the log filename and the option can be set on a per session, per client machine or per user basis. If the specified log file does not exist, it will be created. If it does exist, new log messages will be appended to it.

The default setting is

```
IMAP_LOGFILE=~/tmp/imap.%h
```

where

~ represents the OpenMail directory, usually `/var/opt/openmail`

`%h` is replaced by the fully qualified client hostname, for example, `kuda.pwd.hp.com`.

So, if `IMAP_LOGLEVEL` is set in `general.cfg` to enable IMAP logging across the system and the default setting of `IMAP_LOGFILE` is used, all IMAP communications with the client host machine, `kuda.pwd.hp.com`, will be logged in the file, `/var/opt/openmail/tmp/imap.kuda.pwd.hp.com`.

Note that several IMAP server processes can all log to the same log file without corrupting it. The example above is quite likely to include the logging from several different IMAP server processes.

Other substitutions that can be made are `%p` and `%u`, where

`%p` is replaced with the PID of the IMAP server process, `in.imap41d`

`%u` is replaced with the OpenMail numeric user ID. If logging starts before the IMAP connection is authenticated, `%u` will be replaced with zero instead of the OpenMail user ID.

For example, let's say you want to capture detailed logging for individual users on the client machine, `kuda.pwd.hp.com`, so in `~openmail/sys/client.cfg/kuda.pwd.hp.com`, you set

```
IMAP_LOGLEVEL=8
IMAP_LOGFILE=~/tmp/imap.%u
```

Instead of seeing individual log files, `~/tmp/imap.<`*`OpenMail ID`*`>`, you will find one file, `~/tmp/imap.0`, containing the detailed logging for all the users on the client machine. This is because logging has already been enabled (at client level) before the users login (i.e. before the users are authenticated).

To log the individual users, you can either

- Set `IMAP_LOGFILE` and `IMAP_LOGLEVEL=0` in `general.cfg` and
- Set `IMAP_LOGLEVEL=8` in the users' configuration files in `~/sys/user.cfg`.

or

- Set the log level and the log file options in the users' configuration files in `~/sys/user.cfg`.

If our users have the OpenMail IDs, 102 and 103, we could set

```
IMAP_LOGLEVEL=8
IMAP_LOGFILE=~/tmp/imap.%u
```

in `~openmail/sys/user.cfg/102` and `~openmail/sys/user.cfg/103`. We would then find the files `~/tmp/imap.102` and `~/tmp/imap.103` automatically created, because logging is turned on at the correct level for these options to take effect.

More than one substitution can be specified. For example, setting

```
IMAP_LOGFILE=~/temp/imap4-%u.%h
```

will create log files, like `imap4-102.kuda.pwd.hp.com`, in the OpenMail `temp` directory. Each log file will contain logs for all the sessions of a particular user connecting from a particular client machine.

Note: Where you set the `IMAP_LOGFILE` option is not as important as where you set the `IMAP_LOGLEVEL` option.

As with `IMAP_LOGLEVEL`, it is recommended that you restart the IMAP daemon after changing `IMAP_LOGFILE`.

In B.05.10, the option `UAL_IMAP4_LOGFILE` defined the log file but it did not support subsitutions. The log file name always had the process ID of the current IMAP server appended to it.

IMAP_UAL_TRACE_LEVEL

This option allows you to turn up the UAL logging level for IMAP connections without affecting the level set for other UAL connections. When you enable IMAP server logging (using `IMAP_LOGLEVEL`), `IMAP_UAL_TRACE_LEVEL` is automatically set to 8.

The levels you can set are the same as for `UAL_TRACE_LEVEL` and the default setting is

```
IMAP_UAL_TRACE_LEVEL=0
```

that is, it is not enabled.

This option can be set for the whole system or for a particular client machine. It cannot be set for a particular user, because UAL logging has to be enabled before the user's OpenMail ID is known.

To ensure all connections use the new setting, it is advisable to restart the Remote Client Interface (rci).

Lab

In this lab, you need two OpenMail users. They will both be running IMAP sessions from the same client machine. For this it is probably easiest just to use telnet IMAP sessions.

Set IMAP logging so that:

1. Detailed logging is enabled for both users and separate log files are created for each user.

   Answer:

   Make sure IMAP logging is not enabled in `general.cfg` or for the client host machine, in the `client.cfg` directory.

   Set up a user configuration file for each user in `~/sys/user.cfg` and in each, put the lines:

   ```
   IMAP_LOGLEVEL=8
   IMAP_LOGFILE=~/tmp/imap.%u
   ```

   Restart the IMAP daemon. If you changed the `general.cfg` file, you will also need to restart the remote connection interface.

   Start a telnet session for each user.

   You should find user specific files in `~/tmp`, for example, `imap.102` and `imap.103`.

2. Enable summary logging for all the IMAP connections on the system but turn up IMAP logging for one user. (Start IMAP sessions for the two users). What is the log file called?

   Answer:

   In `general.cfg`, set:

   ```
   IMAP_LOGLEVEL=3
   ```

   In one user configuration file leave IMAP logging turned up. Rename the other user configuration file to disable it.

   Restart the IMAP daemon and the remote connection interface.

   Start a telnet session for each of the two users.

   You should find a client specific log file in `~openmail/tmp`, for example, `imap.kuda.pwd.hp.com`. This should contain `CMD` and `RSP` entries for user sessions other than the one with logging turned up. For this user, you should see `C:` and `S:` entries after login.

3. We want to see, separated out, the logging for individual IMAP processes for one user's session. What needs to be configured?

   Answer:

   Make sure IMAP logging is not enabled in `general.cfg` or for the client host machine, in the `client.cfg` directory.

   Set up one user configuration file in `~/sys/user.cfg` and put in the lines:

   ```
   IMAP_LOGLEVEL=8
   IMAP_LOGFILE=~/tmp/imap.%u.%p
   ```

   Restart the IMAP daemon. If you changed the `general.cfg` file, you will also need to restart the remote connection interface.

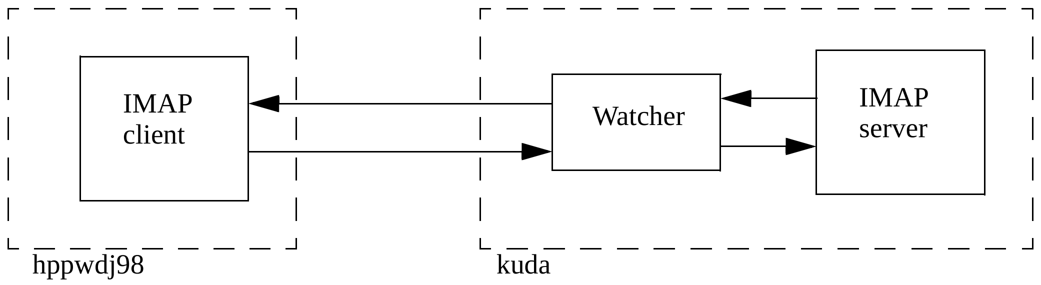
   Start a telnet session for the user. This may be more interesting if you start a Netscape session for this user, as you will see more log files created.

   You should find log files in `~openmail/tmp` with filenames like, `imap.102.2345` and `imap.102.5678`, where the first number is the OpenMail user ID and the second number is the PID of an `in.imap41d` (IMAP server) process.

Watcher Connection Monitoring Tool

Watcher is a TCL script, which sits between the client and server and provides you with a graphical window on all the raw protocol/data exchanged between the two. Although it is used here with IMAP connections, it can actually be used to monitor other communication connections; SMTP, for example. You can run Watcher on a Unix machine or a PC - wherever you can install TCL.

The diagram below shows how Watcher just acts as a relay between client and server. I have Watcher and the IMAP server running on kuda and the IMAP client running on hppwdj98.



I have included the Watcher script in “Watcher TCL Script” on page 60 of this OTN. However, it will be made available from the Contributed Tools section of the OpenMail website.

TCL scripts are interpreted using the TCL library, available from *www.scriptics.com*. If TCL is installed on your machine, you just need to make sure that the TCL library, `/opt/tcl/bin`, is added to `PATH`. If the library exists on another machine, you need to create a symbolic link first, then add the TTCLcl interpreter library to your `PATH`. For example, the TCL library I want to use is on another machine, called `fred`, so first I create a symbolic link to the TCL directories, `/opt/tcl`, on `fred`:

```
ln -s /net/fred/opt/tcl /opt
ls /opt/tcl
```

Then I add the TCL library, `/opt/tcl/bin`, to my `PATH`:

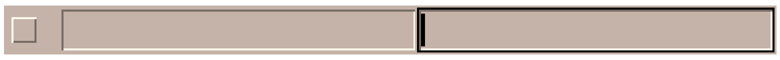
```
PATH=$PATH:/opt/tcl/bin
```

Configuring Watcher

To start Watcher, you simply type:

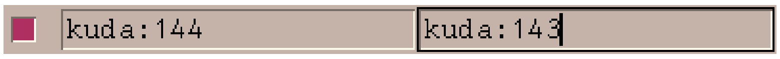
```
watcher &
```

and the following box will appear:



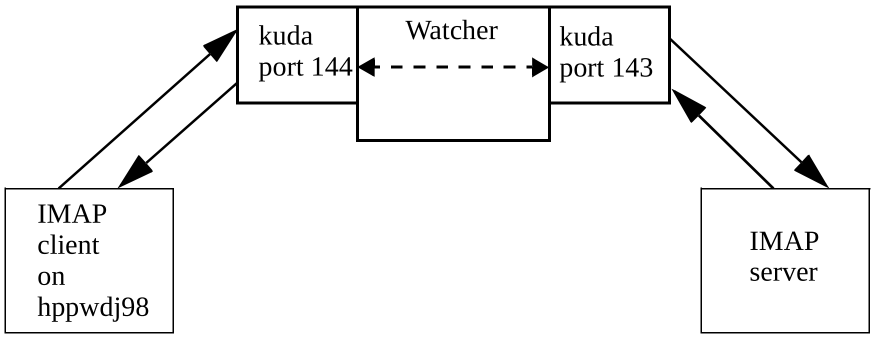
On the left is a button you click on to start monitoring. You can tab between the following two fields. In the left hand field, you enter details of the client side connection with which Watcher will communicate. In the right hand field, you enter details of the server side connection with which Watcher will communicate. Watcher then just displays and relays traffic between these two addresses.

For example:



kuda:144 kuda:143

Let's look at what this sets up in the diagram below:



Watcher will take traffic arriving on port 144 on kuda, display it and relay it to port 143 on kuda. Going the other way, Watcher will take traffic arriving on port 143 and relay it to port 144.

In the above set up, I am running Outlook Express 5 on the host machine, hppwdj98. In Outlook Express (Account Properties window, Servers tab), I set kuda as the Incoming Mail (IMAP) server. Under the Advanced tab, I configured 144 as the IMAP server port.

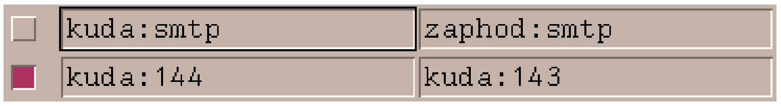
We could have configured the connections to be "watched" from the command line, when we started Watcher:

```
watcher kuda:144 kuda:143 &
```

The following example shows how you can configure Watcher from the command line to monitor an IMAP connection on kuda and an SMTP connection between kuda and zaphod:

```
watcher kuda:143 localhost:143 kuda:smtp zaphod:smtp
```

The resulting window would look like this:

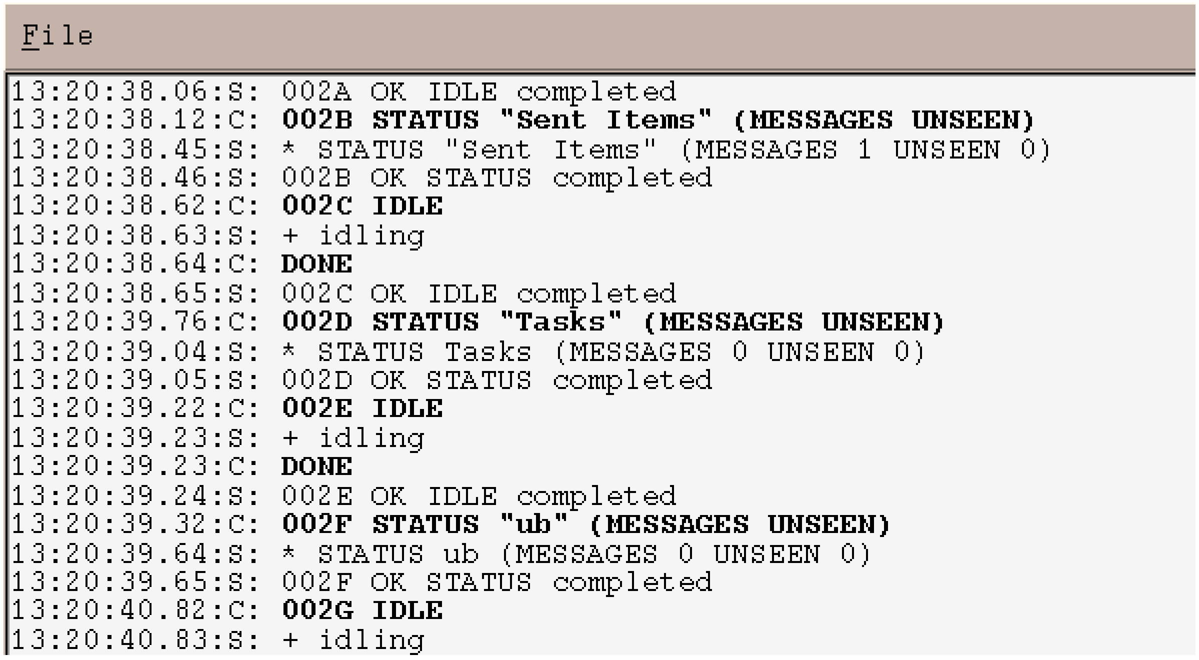


☐	kuda:smtp	zaphod:smtp
■	kuda:144	kuda:143

You initiate monitoring for each connection independently by clicking the button to the left of the connection you want to monitor.

Watcher Transcript Windows

A new window, like the one below, is started for each new connection established.



```
13:20:38.06:S: 002A OK IDLE completed
13:20:38.12:C: 002B STATUS "Sent Items" (MESSAGES UNSEEN)
13:20:38.45:S: * STATUS "Sent Items" (MESSAGES 1 UNSEEN 0)
13:20:38.46:S: 002B OK STATUS completed
13:20:38.62:C: 002C IDLE
13:20:38.63:S: + idling
13:20:38.64:C: DONE
13:20:38.65:S: 002C OK IDLE completed
13:20:39.76:C: 002D STATUS "Tasks" (MESSAGES UNSEEN)
13:20:39.04:S: * STATUS Tasks (MESSAGES 0 UNSEEN 0)
13:20:39.05:S: 002D OK STATUS completed
13:20:39.22:C: 002E IDLE
13:20:39.23:S: + idling
13:20:39.23:C: DONE
13:20:39.24:S: 002E OK IDLE completed
13:20:39.32:C: 002F STATUS "ub" (MESSAGES UNSEEN)
13:20:39.64:S: * STATUS ub (MESSAGES 0 UNSEEN 0)
13:20:39.65:S: 002F OK STATUS completed
13:20:40.82:C: 002G IDLE
13:20:40.83:S: + idling
```

The window displays the client/server conversation over that particular connection. Each line takes the form:

`<`*`timestamp`*`>:C|S: <`*`dialogue`*`>`

The timestamp, given in hours, minutes, seconds and centi-seconds, is Watcher's and is not linked to the time on the servers being monitored.

`C` is for Client side of the dialogue and `S` is for the Server side. The client dialogue is displayed in bold.

In the window above, the `*** EOF ***` lines show that this temporary connection is now closed. Dialogue will continue to be displayed in the main connection window and new windows will appear as new temporary connections are established. In IMAP sessions particularly, this is helpful, because you can see easily how many new connections a client makes within a session, and what the new connections are used for.

Saving Connection Dialogues

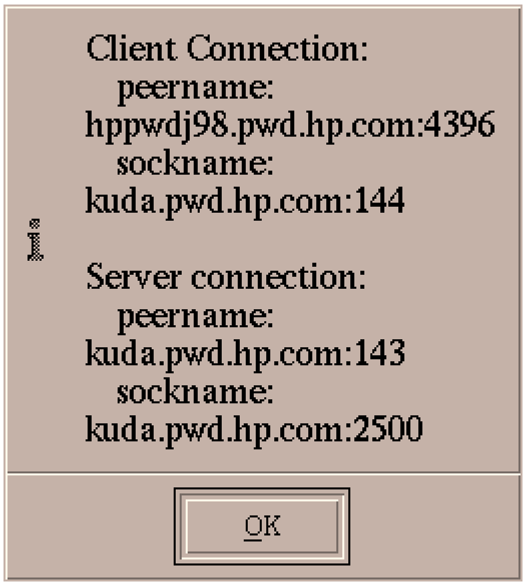
If you click File in the top left of a transcript window, you can opt to save in a file:

- The whole transcript displayed in the window
- Just the server side of the dialogue
- Just the client side of the dialogue

By default, the filename will be the `<`*`title of the window`*`>.scr`, for example, `watch0.scr`.

More Connection Configuration Details

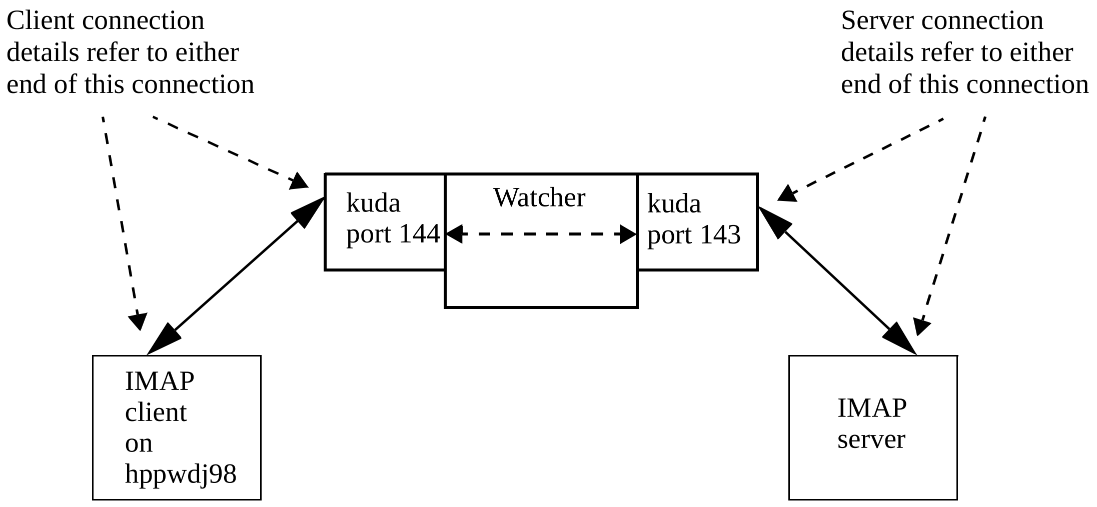
If you press Help in the top right of a transcript window, a dialogue box like the one below will appear.



Client Connection:
peername:
hppwdj98.pwd.hp.com:4396
sockname:
kuda.pwd.hp.com:144

Server connection:
peername:
kuda.pwd.hp.com:143
sockname:
kuda.pwd.hp.com:2500

This gives further details about the particular client/server connection to which this window relates. Both ends of a socket connection need to be identified, so an arbitrary number, assigned by the operating system, is given for the ends we have not specified, i.e. `hppwdj98.pwd.hp.com:4396` and `kuda.pwd.hp.com:2500`. What the `peername` and `sockname` refer to is probably best described using the diagram we had earlier:



The important thing to note with the Help dialogue box is that it shows the client host machine to which this dialogue refers. For instance, the Watcher configuration we set up:

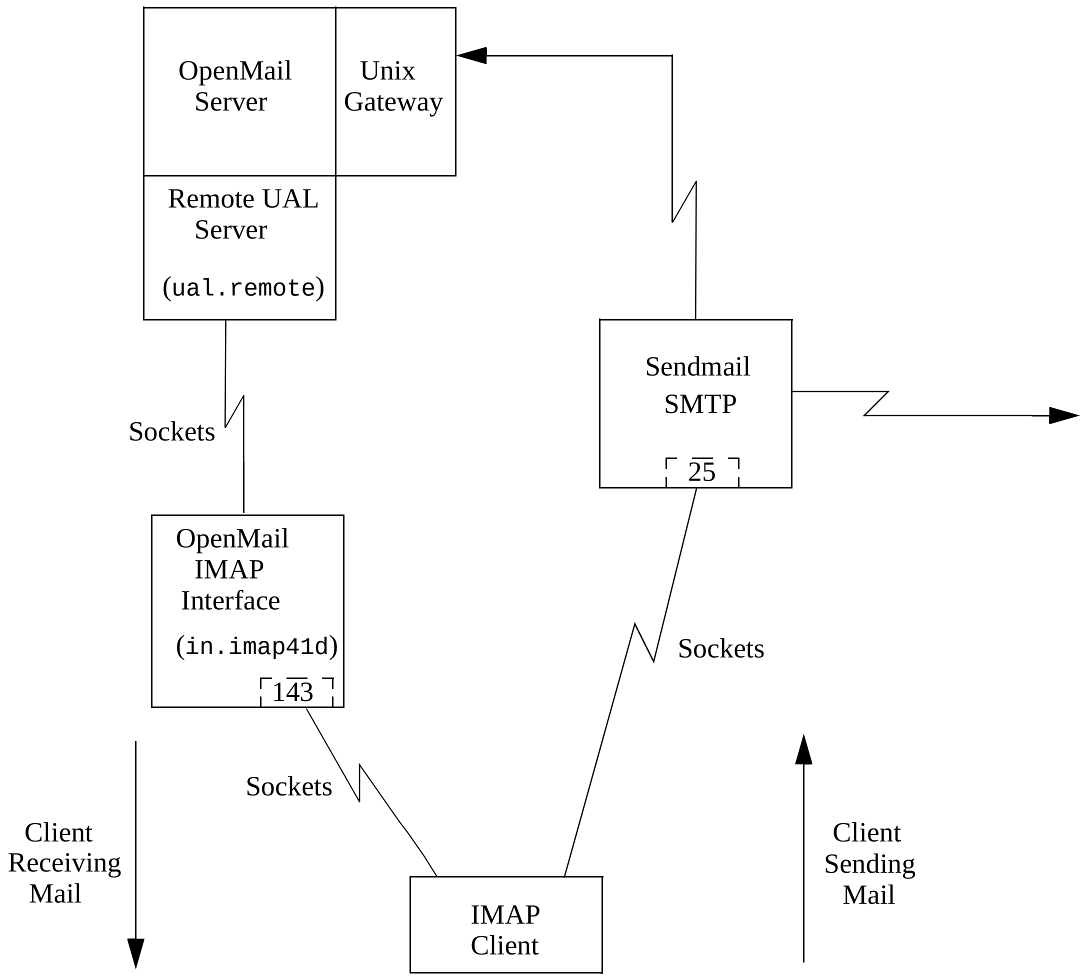
```
watcher kuda:144 kuda:143 &
```

would monitor both my Outlook Express session from hppwdj98 and a telnet session to port 144 on kuda. I could then use the Help dialogue information to find out which dialogue belonged to which session.

Improvements To The OpenMail IMAP Implementation

The following diagram of the OpenMail IMAP architecture should already be familiar to you - it is from the original *IMAP4 OTN*.

Figure 1 OpenMail IMAP Architecture



At B.06.00, the IMAP server, `in.imap41d`, is now a standalone daemon, which means that it can be started and stopped with `omon` and `omoff`, e.g. `omon -a imap`.

Although clients continue to make new connections (and consequently, new `in.imap41d` and `ual.remote` processes) during a session, now new copies of the running IMAP server are created, as opposed to new processes being started from scratch (as happened under inet). Consequently, the response time for a new connection has been improved.

Note: A Netscape Messenger session can have up to 7 concurrent connections - a total of 21 processes, which can be a significant problem on heavily loaded systems. If you are likely to have over 1,200 concurrent Netscape Messenger (4.5 or later) IMAP sessions, you will need to limit the number of new connections that Netscape clients can make. How to do this is described in the section, “Limiting The Number Of New Processes Netscape Can Start In A Session” on page 58.

The basic architecture of OpenMail IMAP support has not changed much at B.06.00 but there have been changes to the way the processes involved interact. As we shall see, these changes give significant improvements in performance and reliability.

Probably the three most important changes are the use of (virtual) memory mapped I/O, single slot caching by the IMAP server and the use of server push notifications:

- Large amounts of data are transferred from the IMAP server to the IMAP client, the use of memory mapped I/O both improves performance and greatly reduces the load on the server machine.

  If the UAL server and the IMAP server reside on the same machine, then files are transferred directly from the Message Store instead of being copied across the net using `ual_recvfile` and `ual_sendfile`.

- IMAP clients are increasingly "paging" data or asking for information about the message before fetching it. For example, the Jornada typically fetches a single message over about half a dozen consecutive `FETCH` requests, Netscape Messenger 4.5 asks for the body structure before fetching the parts of the message that it is able to display, older versions of Messenger have a tendency to fetch a message twice (just in case).

  This makes the implementation of single level caching very effective. The OpenMail IMAP server caches basic information about all the messages in the current mailbox in its memory and maps all of the current message into virtual memory.

  Extending the partial fetching of messages is an important feature of IMAP4rev1. To optimize this feature in the OpenMail IMAP server, new temporary object files, with the index name `imapBodyStrc`, are used to cache the message header and structural information about a message. These object files exploit the ability of the item browser to render parts of a message, instead of the whole message.

- The B.06.00 IMAP server makes extensive use of server-push notifications, which means that, when idle, it does not need to contact its associated UAL server. This mechanism is made visible to the IMAP client through the IDLE extension (see “IDLE” on page 46).

  Server push notifications therefore offer significant performance improvements for those clients which support them (namely Outlook Express and Outlook 98). The UAL server can simply sleep until new mail arrives and the most the IMAP server has to do is to tell the client that nothing has changed.

  At B.05.10, each time a client polled for updated information both the IMAP and the UAL processes were woken up; in many instances, only to report that nothing had changed. A lot of clients polling every few minutes was a significant overhead on the server.

Internet Acknowledgements (MDNs and DSNs) are now supported on OpenMail IMAP and the Internet Gateway. This is covered in another B.06.00 upgrade OTN, *Changes To The Internet Gateway*. Tweaks for supporting the generation of Internet Acks within IMAP are covered in “Support For Internet Acks” on page 31.

What The IMAP Server Caches In Its Memory

The following details for each message in the current mailbox are cached in the IMAP server’s memory data structures:

- The OpenMail container sequence number for this message. This maps onto the IMAP message UID.

  When an IMAP session is initialised, the IMAP server sends an `INIT` command like the following:

  ```
  UAL Server Command Trace at Wed Oct 20 16:41:36 1999
  -------------------------------------------------------
  INIT Command: UA Id=’IMAP4 Server B.06.00.R0’, Debug Level=2, Debug
  File=’’
  INIT Command: Flags=0x20020, CharSet=’UTF8’, Service Level=6
  ```

  The flag, `0x20000`, requests that items in a container be uniquely numbered the first time the container is opened. It also causes subsequently new items to be given a number. These sequence numbers map onto the IMAP UID numbers.

  What the UAL actually requests (in a `LIST` command) is a "Bulletin Board Sequence Number", though here it has nothing to do with Bulletin Boards. Bulletin Board sequence numbers can be used to number items in any container. These sequence numbers always begin at 1 for the first item in the container, and increase by 1 for each item added. If items are deleted, then, unlike the normal item reference, these numbers are never used again in the same container.

- The direct reference of the message in the message store.
- `INTERNALDATE` - the OpenMail delivery date, that is, the time the message was put in this folder.
- Various flags including `\Deleted`, `\Answered` etc. The IMAP server keeps account of previous flag settings, so that it knows what has changed.

Caching Rendered Messages

In addition, when a message is first fetched from the message store, it is converted back into MIME format ("rendered") by the item browser and stored in a UAL temporary file. The IMAP server maps this file into virtual memory. This message remains current until another is fetched.

This memory mapping makes paging large messages very efficient. It works in the same way as fetching part of a message, which we look at in the section "Fetch Part Of a Message" on page 35.

The imapBodyStrc Object File

The item browser also creates and automatically maintains an `imapBodyStrc` object file for each fetched message. This temporary object file provides a cache for the message header and structural information about the message body.

The old object files, `imapHdr`, which held the message header, and `imapMsg`, which held the header and rendered message, are no longer created (except in an IMAP `APPEND` request, which we shall look at later).

Note: The `IMAP_CACHE_RFC_HEADER` and `IMAP_CACHE_RFC_MESSAGE` tweaks associated with the old object files are now ignored. In previous releases, these tweaks could be used to turn off the creation of cached object files.

What is the imapBodyStrc object file used for?

The prime reason for this file is to optimize the ability of the item browser to render part of a message instead of having to render the complete message every time. This ability is increasingly used by modern clients, such as, Jornada, Netscape 4.6 and Mulberry.

What is in the imapBodyStrc object file?

The information in this file is roughly equivalent to that returned by

```
FETCH n (RFC822.SIZE BODY[HEADER] BODYSTRUCTURE)
```

One way to see if the object file exists is to use the `lo` (List Object) option in `omcontain`.

- Start a telnet IMAP session and fetch a message bodystructure, to force the item browser to create an `imapBodyStrc` file for that message:

```
root@kuda[] #telnet kuda 143
Trying...
Connected to kuda.pwd.hp.com.
Escape character is '^]'.
* OK OpenMail IMAP server B.06.00.R0 ready on kuda
a1 login admin admin
a1 OK LOGIN completed, now connected to kuda.pwd.hp.com
a2 select inbox
* 22 EXISTS
* 0 RECENT
* OK [UIDVALIDITY 1] UIDVALIDITY value
* FLAGS (\Answered \Flagged \Deleted \Seen \Draft $MdnSent)
* OK [PERMANENTFLAGS (\Answered \Flagged \Deleted \Seen \Draft
$MdnSent)] flags will stay set
a2 OK [READ-WRITE] SELECT completed
a3 fetch 22 bodystructure
* 22 FETCH (BODYSTRUCTURE ("TEXT" "PLAIN" ("charset" "iso-8859-1") NIL
NIL "QUOTED-PRINTABLE" 23 2 NIL ("inline" ("filename" "a")) NIL))
a3 OK FETCH completed
```

- Start `omcontain` and open the intray of this user:

```
Option?u
User:admin

User Name = Mr Admin /canberra,class
User Number = 102
Ctr Handle = 3001
Done

Option?o
Ctr Handle (3001):
Entry Number:1
Ctr Handle = 3002
Done
```

- Find the message you fetched in the telnet session and list the object file associated with it by entring the option `lo` and the number of the message in response to `Entry Number`:

```
Option?lo
Ctr Handle (3002):
Entry Number:29
Record No.     0
Index name  = imapBodyStrc      Item Number = 4284
Filename    = ~/data/000000p/00006o1
Flags          = 0      Domain         = 2
SubDomain      = 25     FileId         = 6913
Object Flags   = 3      File Extension =
Done
```

The value of the Object Flags is:

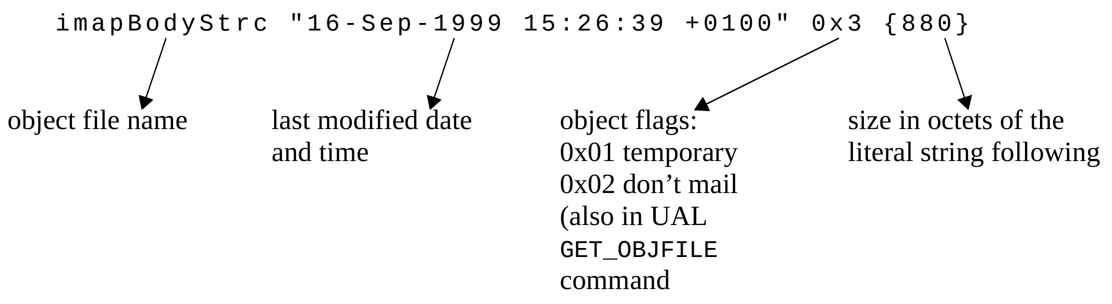
0x01 temporary

0x02 don’t mail

We can actually view the contents of `imapBodyStrc` object files using an `X-OMTEST` extension command, `om.objfile` (see “X-OMTEST” on page 53). Here is an example of the `imapBodyStrc` object file associated with message 1 in my mailbox:

```
a5 fetch 1 om.objfile (imapBodyStrc)
* 1 FETCH (OM.OBJFILE ((imapBodyStrc "16-Sep-1999 15:26:39 +0100" 0x3
{880}
OMHEADER (VERSION 1.2 FLAGS "Q 0 0 0 NIL Q 0 Q 0 NIL NIL F 0 0 1")
RFC822.HEADER {429}
Date: Thu, 16 Sep 1999 15:24:11 +0100
Message-Id: <H000006600000a85.0937491849.kuda.pwd.hp.com@MHS>
Subject: 2nd rtf test
MIME-Version: 1.0
Return-Path: <Admin_Mr/canberra_class@kuda.pwd.hp.com>
From: Admin_Mr/canberra_class@kuda.pwd.hp.com
To: Admin_Mr/canberra_class@kuda.pwd.hp.com
Content-Type: application/rtf; name="BDY.RTF"
Content-Disposition: attachment; filename="BDY.RTF"
Content-Transfer-Encoding: base64

ENV (DATE "Thu, 16 Sep 1999 15:24:11 +0100" SUBJECT "2nd rtf test" FROM
(Admin_Mr/canberra_class@kuda.pwd.hp.com) TO (Admin_Mr/
canberra_class@kuda.pwd.hp.com) MSGID "<H000006600000a85.0937491849.kuda.
pwd.hp.com@MHS>") BODY ((PATH 1 TYPE APPLICATION STYPE RTF TPARMS (NAME
BDY.RTF) ENC BASE64 SIZE 654 DISP ATTACHMENT DPARMS (FILENAME BDY.RTF)))
SIZE 1083 LINES 20)))
a5 OK FETCH completed
```

The first 2 lines of the response contain item browser information. The first line contains the following information:



```
imapBodyStrc "16-Sep-1999 15:26:39 +0100" 0x3 {880}
```

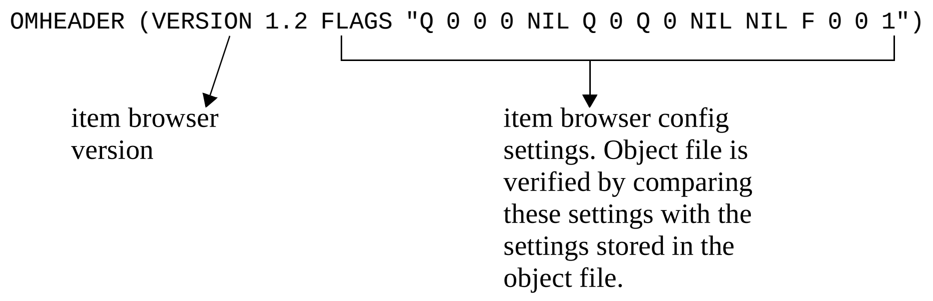
object file name

last modified date and time

object flags:
0x01 temporary
0x02 don't mail
(also in UAL `GET_OBJFILE` command

size in octets of the literal string following

The second line is the item browser header:



```
OMHEADER (VERSION 1.2 FLAGS "Q 0 0 0 NIL Q 0 Q 0 NIL NIL F 0 0 1")
```

item browser version

item browser config settings. Object file is verified by comparing these settings with the settings stored in the object file.

The next part is the message header as a literal string (size 429 octets):

```
RFC822.HEADER {429}
Date: Thu, 16 Sep 1999 15:24:11 +0100
Message-Id: <H000006600000a85.0937491849.kuda.pwd.hp.com@MHS>
Subject: 2nd rtf test
MIME-Version: 1.0
Return-Path: <Admin_Mr/canberra_class@kuda.pwd.hp.com>
From: Admin_Mr/canberra_class@kuda.pwd.hp.com
To: Admin_Mr/canberra_class@kuda.pwd.hp.com
Content-Type: application/rtf; name="BDY.RTF"
Content-Disposition: attachment; filename="BDY.RTF"
Content-Transfer-Encoding: base64
```

This is then followed by envelope information held by the item browser. I have reformatted this part to make it more readable:

```
ENV (                                -- envelope, for "FETCH n ENVELOPE"
    DATE "Thu, 16 Sep 1999 15:24:11 +0100" -- same as RFC822 Date: header
    SUBJECT "2nd rtf test"           -- message subject
    FROM (                           -- list of FROM addresses
        Admin_Mr/canberra_class@kuda.pwd.hp.com
    )
    TO (                             -- list of TO addresses
        Admin_Mr/canberra_class@kuda.pwd.hp.com
    )
    MSGID "<H000006600000a85.0937491849.kuda.pwd.hp.com@MHS>" --message id
)
BODY (                               -- body, for "FETCH n BODYSTRUCTURE"
    (
        PATH 1                       -- browser path to message part
```

```
                TYPE APPLICATION              -- MIME type
                STYPE RTF                     -- MIME subtype
                TPARMS (                      -- content-type parameters
                    NAME BDY.RTF
                )
                ENC BASE64                    -- content-transfer encoding
                SIZE 654                      -- body size (in bytes)
                DISP ATTACHMENT               -- content-disposition
                DPARMS (                      -- content-disposition parameter
                    FILENAME BDY.RTF
                )
            )
        )
        SIZE 1083                             -- total size of message
        LINES 20                              -- total number of lines (for IMAP)
```

What is meant by "automatically maintained"?

In OpenMail B.05.10, if you remember, the object files, `imapHdr` and `imapMsg`, were not automatically updated when messages or settings changed. To ensure changes could take effect, you had to set the tweak, `UAL_IMAP4_DEL_OBJ=TRUE`, and issue an IMAP `CHECK` command to force the deletion of the object files. This is no longer required at B.06.00.

The item browser will now automatically recreate the `imapBodyStrc` file if any of the following change:

- Item browser steering files
- Item browser tweaks

or if the item browser itself is modified.

In the annotated example above, the `OMHEADER` line holds information that the item browser uses to validate the file. If this information is incorrect, it will automatically create a new object file for the message.

Since the `imapBodyStrc` (and therefore the original MIME message) can always be re-constructed from the message and content record files, there is no need to mark any `imapBodyStrc` object file as a permanent object file. They can then be deleted by the IMAP server when they have been made obsolete (see “Deleting Temporary And Permanent Object Files” on page 23).

How The Size Of A Message Is Calculated

You will notice the size of the message is included at the end of the `imapBodyStrc` object file. If the tweak, `IMAP_MIN_SIZE_ESTIMATE`, has the value 0 (zero) - the default value - the size of a message is always calculated by the item browser rendering the whole message.

For clients like Outlook Express and Netscape Messenger, which fetch the `RFC822.SIZE` and `RFC822.HEADER` attributes of a message when just listing the contents of a folder, this can represent a significant overhead.

The tweak can be set so that an estimated size can be returned for messages over a certain size. The estimated size is taken from size of message in the message store, which is held in the message container. So, for example, setting

```
IMAP_MIN_SIZE_ESTIMATE=5
```

would mean that an estimated size is to be returned for messages which occupy more than 5k bytes in the message store. Messages taking up less than 5k bytes in the message store are to be rendered to calculate their size. Rendering these small messages to discover their size is not time-consuming.

For large messages, the estimated size in the message store may be less, as parts of the message (e.g. binary attachments) would be rendered in base64. For small messages, the estimated size in the message store may often be greater than the real size; the minimum size of a message in the message store is 1k bytes. The estimate could be as much 30% out for large messages and several hundred percent for small messages. The *error* for large messages is usually acceptable to users, the error for small messages is not, which is why we have included this tweak.

Using this tweak may, therefore, give you a significant performance improvement at the cost of some loss of accuracy in the reported size of the message.

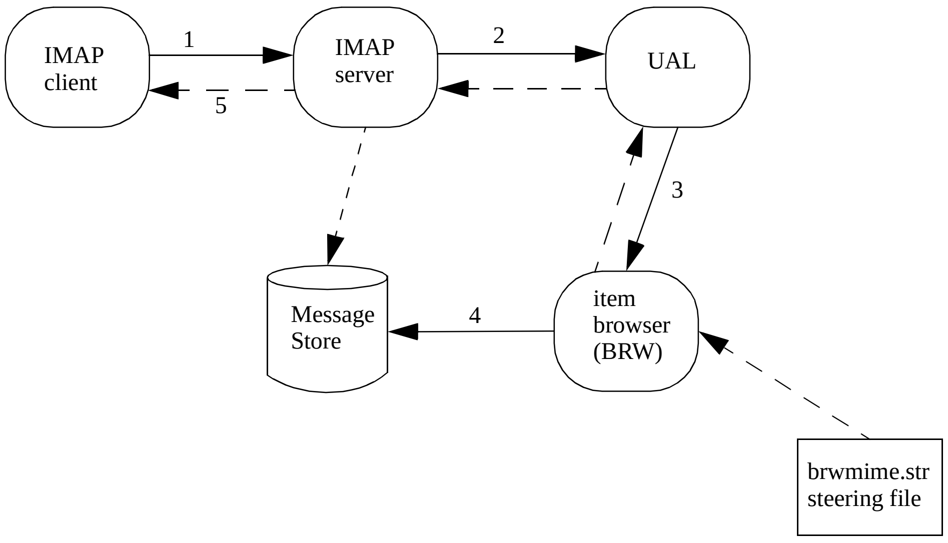
Note: Some clients, Pine, for example, rely on the `RCS822.SIZE` being accurate. So, for for users of these clients, this tweak must be set to zero.

How The OpenMail IMAP Server Talks To The Message Store

It is worth looking at the overall picture, at this stage, to see how the various servers and processes interwork at B.06.00. For example, what happens when an IMAP client issues a command to fetch a message from the message store:

```
FETCH 4 RFC822
```

The communication between the various parts of OpenMail is shown in the diagram below. Each stage in the communication is represented by a number. These stages are then explained in the numbered list following the diagram:



In this diagram I have used circular shapes to represent processes, to distinguish them from files (represented by rectangles) and the main Message Store database.

1. First of all, the client issues the `FETCH` command, asking the IMAP server to get all of the message with the UID of `4`.
2. The IMAP server then gives the UAL the name of a temporary file and asks it to render message into it.
3. The IMAP server first checks to see if an `imapMsg` message object file already exists for this message. If not, it passes the *render* request to the UAL. The UAL then passes it to the item browser, which actually does the rendering. *Rendering* here means reconstructing the original MIME message from the OpenMail internal format. The UAL also hands to the item browser the item reference of the message and the name of the destination temporary file.

   If an `imapMsg` message object file already exists for this message, the UAL copies it to the temporary file. Note that this removes the need to render the message.

4. Before rendering the message, the item browser checks the steering file, `brwmime.str`, for any required character set and bodypart conversion.

   The item browser renders the message into the temporary file indicated by the direct reference and also creates an `imapBodyStrc` object file to hold basic structural information about the message.

Note: For the most part, we no longer store the whole (rendered) message in both a permanent object file (`imapMsg`) and in OpenMail format in the Message Store. This significantly reduces the amount of disk space required for IMAP clients.

   When the item browser is finished, it reports back through the chain to the IMAP server to say the task is complete.

5. The IMAP server already knows the name of the temporary file and simply sends the contents out to the client.

In B.05.10, the UAL would have had to physically copy the file over the net to the IMAP server. Now this is not required, because the item browser puts the rendered message in a file already known to the IMAP server and just reports back when it is finished.

Old imapHdr And imapMsg Object Files

As the IMAP object files at B.06.00 are completely different from those created at B.05.10, we strongly recommend that you delete all old temporary IMAP object files after you have upgraded to OpenMail B.06.00, (see “Deleting Temporary And Permanent Object Files” on page 23).

When the IMAP server wants all or some of a message, or the message structure, it first checks for an `imapMsg` object file. If this exists, it will be used in preference to anything generated by the item browser. This check will show in a UAL trace as a `GET_OBJFILE` command:

```
COMMAND 192     GET_OBJFILE     11:18:29
{
        UARef            = 13589
        SeqNo            = 8
        ListRef          =
        ItemRef          =
        Flags            = 2
        ObjectIndex      = imapMsg
        TargetFileName   =
        TargetFileId     = 210
        DirectRef        = 000102662219e793
}

REPLY 192       GET_OBJFILE
{
        UARef            = 13589
        SeqNo            = 8
        ReplyFlags       = 0
        ErrorNo          = 1802  <---------  imapMsg obj file doesn't exist
        Err2             = 0
        Err3             = 0
        ObjFileName      =
        FileSize         = 0
        ModDate          = 0
        ObjFlags         = 0
}                       11:18:29
```

If an `imapMsg` object file is not present, then a new `imapBodyStrc` file is created.

Note: You will no longer see the `ADD_OBJFILE` command in the UAL trace, (except when the IMAP `append` command is issued), as this is now done implicitly, as required, by the item browser.

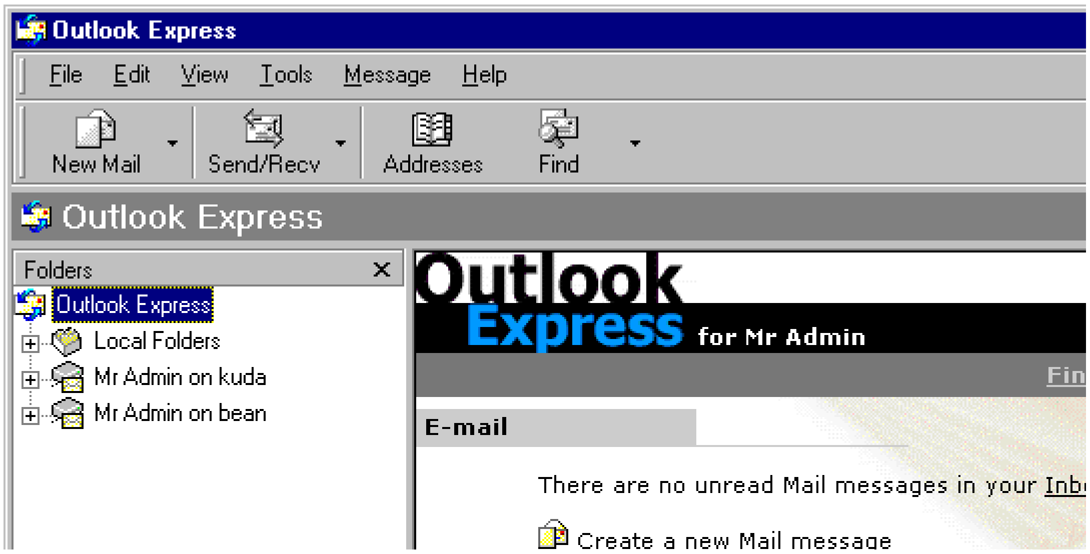
Permanent IMAP Object Files

When a message is dragged from one mailbox (on one server) to another mailbox (on a different server), the message is copied over using the IMAP `APPEND` command. The whole OpenMail message is stored as a literal in a permanent `imapMsg` object file. Currently, an `imapHdr` object file is also created, which is a known bug.

This means that copying files in this way will almost double the space the message occupies on the receiving machine.

Let's create a message with a permanent object file (`imapMsg`). Then use `omcontain` to list this.

- Use Outlook Express to set up two accounts, one on one server and one on another. I have Mr Admin on bean and Mr Admin on kuda, so my mailbox window looks like this:



- Move a message by dragging and dropping it from the inbox of one account to the inbox of the other. What will this create?

  I have dragged a message from kuda to bean.

- Open an `omcontain` session on the receiving server and open the intray of the IMAP user. I have opened and listed Mr Admin's intray on bean:

```
Option?l
Ctr Handle (3002):
    2) RE: from remote                         (MESSAGE)
    3) test message                            (NON-DELIVERY MESSAGE)
    7) who's this from                         (MESSAGE)
    8) testing again                          (MESSAGE)
    9) a little note                           (MESSAGE)
Done
```

  We can now use `lo` to list the object files associated with the message we moved across (9):

```
Option?lo
Ctr Handle (3002):
Entry Number:9
Record No.     0
Index name   = imapMsg   Item Number = 1602
Filename     = ~/data/0000001/0000202:2
```

```
Flags          = 2      Domain         = 2
SubDomain      = 1      FileId         = 2050
Object Flags   = 2      File Extension =
Record No.     1
Index name  = imapHdr   Item Number = 1602
Filename    = ~/data/0000001/0000202:3
Flags          = 3      Domain         = 2
SubDomain      = 1      FileId         = 2050
Object Flags   = 2      File Extension =
Done
```

Note that the `imapHdr` file has the temporary Object Flag set (0x01), but the `imapMsg` file doesn't, which means it is permanent. As I said before, the `imapHdr` object file shouldn't be created at all.

An `imapBodyStrc` object file is not needed for this message, because the complete, rendered message is permanently available.

Deleting Temporary And Permanent Object Files

As a result of improvements to the item browser at B.06.00, the IMAP object files are completely different from those created at B.05.10. We strongly recommend that you delete all old temporary IMAP object files after you have upgraded to OpenMail B.06.00. You can easily do this using `omscan` with the following parameters:

```
omscan -a -Tn -O 'imap[MH]*'
```

where

`-a` means perform this on all directories, users etc.

-T*n* specifies the time, in days, since the file was accessed. So, for instance, if you wanted to delete all the object files which have not been accessed in the last 3 days, you would include `-T3` in the command line.

`-O 'imap[MH]*'` identifies the object files to delete. `-O` means delete temporary object files and you can specify the filename filter as`'imapM*'` or `'imapH*'`.

So, the effect of the commands:

```
omscan -a -T4 -O 'imapM*'
```

and

```
omscan -a -T4 -O 'imapH*'
```

is to search all directories, users etc. and delete any temporary object files with names beginning "`imapM`" or "`imapH`", which have not been accessed in the last 4 days.

Object files can also be deleted using `omcontain` and the `X-OMTEST` extension command, `xom.delobj`. We look at how to do this using `omcontain` in the next section. The extension command is covered later, in the section “X-OMTEST” on page 53.

omcontain

You can also use the `do` option in `omcontain` to delete both permanent and temporary IMAP object files.

Let's use `omcontain` to delete the permanent object file (`imapMsg`) that we created when we dragged and dropped the message from Mr Admin's inbox on kuda to Mr Admin's inbox on bean.

- Open an `omcontain` session on the receiving server and open the intray of the IMAP user. I have opened and listed Mr Admin's intray on bean:

```
Option?l
Ctr Handle (3002):
    2) RE: from remote                           (MESSAGE)
    3) test message                              (NON-DELIVERY MESSAGE)
    7) who’s this from                          (MESSAGE)
    8) testing again                           (MESSAGE)
    9) a little note                            (MESSAGE)
Done
```

We can now use `lo` to list the object files associated with the message we moved across (9):

```
Option?lo
Ctr Handle (3002):
Entry Number:9
Record No.     0
Index name  = imapMsg   Item Number = 1602
Filename    = ~/data/0000001/0000202:2
Flags          = 2      Domain         = 2
SubDomain      = 1      FileId         = 2050
Object Flags   = 2      File Extension =
Record No.     1
Index name  = imapHdr   Item Number = 1602
Filename    = ~/data/0000001/0000202:3
Flags          = 3      Domain         = 2
SubDomain      = 1      FileId         = 2050
Object Flags   = 2      File Extension =
Done
```

Note that neither `imapMsg` nor `imapHdr` have the temporary Object Flag set (0x01), which means they are permanent object files.

- If we now use the `do` option on this message, all the object files, both temporary and permanent, are deleted:

```
Option?do
Ctr Handle (3002):
Entry Number:9
Done

Option?lo
Ctr Handle (3002):
Entry Number:9
Done
```

Using `omcontain` does not allow you to select specific object files to delete. To do this, you need to use the `X-OMTEST` extension, `xom.delobj`.

Multiple Access Support

How multiple IMAP users concurrently accessing the same mailbox should be handled is the subject of *RFC 2180*. However, *RFC 2683, IMAP4 Implementation Recommendations* is easier to read and takes a more practical approach.

The B.06.00 IMAP server uses server push notifications to present a "real time" view of the message store to clients, as opposed to a "snapshot" view, which the B.05.10 IMAP server presented.

Let’s assume we have two users accessing the same mailbox concurrently, one using a client which supports a "real time" view and the other using a client which supports a "snapshot" view. The "real time" user will see, almost immediately, the changes the other user makes to the mailbox, for example, deleting messages and creating folders. The "snapshot" user needs to reconnect before he sees the changes the other user has made. Clients such as Outlook 98 and Outlook Express support a "real time" view. OMGUI, on the other hand, supports a "snapshot" view.

Because of the way the IMAP protocol is implemented, multi-access of mailboxes in real time can sometimes get temporarily out of sync. The example below shows what can happen when one client marks a message for deletion, and another client actually deletes it.

Client 1	**Notification Traffic**	**Client 2**
`a1 SELECT inbox`		`b1 SELECT inbox`
(msg 5 modified)	modification <----------------------------	`b2 STORE 5 +FLAGS \DELETED`
`a2 EXPUNGE`	---Nothing happens!---	
`a3 NOOP`		
`*FETCH 5 FLAGS (/DELETED)`		
`a2 EXPUNGE`	deletion ---------------------------->	(msg 5 modified)
		`b3 FETCH 5 RFC822`
		`NO Some messages no longer exist`
		`b4 NOOP`
		`*5 EXPUNGE`

To explain what is happening above. Both Client 1 and Client 2 are concurrently accessing the same mailbox using IMAP:

1. Client 2 sets the `\Deleted` flag on message 5.
2. Client 1 sees the `\Deleted` flag set on message 5 and immediately attempts to expunge the message.
3. Nothing happens because the server has not yet had the opportunity to update this client with the server response to Client 2’s `STORE` command, which set the `\Deleted` flag.
4. A `NOOP` gives the server the opportunity to update the client with any outstanding responses, in this case, the response to the other client’s `STORE` command.
5. Client 1 then retries the `EXPUNGE` command, successfully this time.
6. Before the server can send a response to Client 2 to inform it that message 5 has been expunged, Client 2 attempts to fetch message 5 again.
7. As the message no longer exists, Client 2 gets the error, `NO Some messages no longer exist`.
8. Again, a `NOOP` command gives the server the opportunity to update the client with any outstanding responses, i.e. that this message has been expunged.

Most clients do not implement the multi-access aspect of the IMAP protocol correctly; after a "`NO`" response from server, they should send a `NOOP` to ensure they are updated with outstanding server responses from actions taken by other clients accessing the mailbox. Outlook Express gets away with it, because it issues an `IDLE` command, which allows the server to send the client unsolicited responses (see “IDLE” on page 46).

Lab

1. Start OMGUI and Outlook Express sessions logged into the same user’s inbox.
2. Send some messages to this user.
3. Do they appear immediately in the client windows? If not, why not?
4. In OMGUI, delete a message. What do you see in the Outlook Express window?

5. In Outlook Express, delete a message. What do you see in the OMGUI window? Why?
6. In OMGUI, create a new folder. What do you see in the Outlook Express window?
7. In Outlook Express, create a new folder. What do you see in the OMGUI window? Why?

The answer to most of the questions above is that OMGUI maintains a "snapshot" view and does not use server push notifications.

Searching

Searching has been significantly improved at B.06.00. Not all searches are handled by the background Search server. Any searches on `FLAGS` and `INTERNALDATE`, which the IMAP server caches in memory, are handled by the IMAP server itself. The response time for such searches is almost instantaneous.

The following search, for example:

```
a1 search deleted since 15-Oct-1999
```

would be performed by the IMAP server, as it holds the flags and `INTERNALDATE` for each message in the mailbox in its memory.

Where text searching is required, either in a header field, such as `From`, or the body of messages, the background Search server is used as described in the *IMAP4 OTN*. A search specification TF file is created containing search filter information and put into the "Search Area" folder (in the Filing Tray) via a `UAL_INCLUDE_FILE` command. Because of the extra processing required, these searches are not as fast as the IMAP server searches.

The Search server has been modified to allow searching against Internet addresses; at B.05.10 searches could only be made on OpenMail addresses, so we can now request a search like:

```
a1 search recent from "joyce@pwd.hp.com"
```

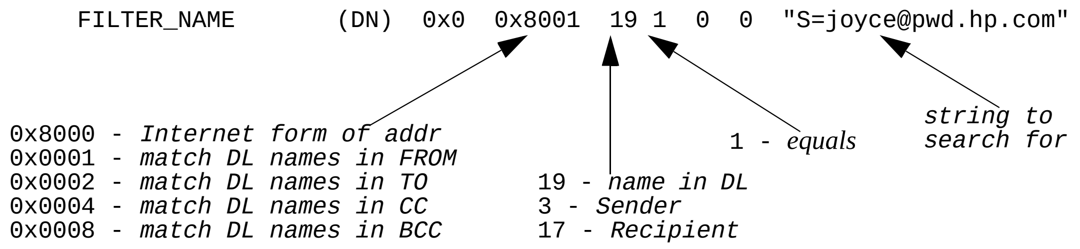
This search request involves searching flags and text, so both the IMAP server and the Search server would be required. First of all, the IMAP server checks its cached data structures to find which messages have the recent flag set. It then constructs a search specification file for the text search, `from "joyce@pwd.hp.com"`, on the "Recent" messages it has found and sends it to the Search server. Only the direct references of the messages which have the `\Recent` flag set are sent to the Search server.

We can look at the Search Specification file constructed for the above search request using `omcontain`:

```
Option?t
Ctr Handle (3003):
Entry Number:12
File Name: ~/data/000000t/00005sg
HEADER            (DN)  1  0  2  1005
  ** Warning - Insufficient fields present  (Record No. 1)
SEARCH_REQUEST    (DN)  0x0  -1  1000
  ** Warning - Insufficient fields present  (Record No. 2)
SEARCH_TARGET     (DN)  "00010241503e9b3d"
  ** Warning - Insufficient fields present  (Record No. 3)
SEARCH_TARGET     (DN)  "0001024234b1dee8"
  ** Warning - Insufficient fields present  (Record No. 4)
FILTER_START      (DN)  0x0  0x0  0
FILTER_NAME       (DN)  0x0  0x8001  19 1  0  0  "S=joyce@pwd.hp.com"
FILTER_END        (DN)
Done
```

From its cached information, the IMAP server determines the "Recent" messages and sends the Search server the direct references for these messages only in the `SEARCH_TARGET` records.

The contents of the `FILTER_NAME` record are:



```
FILTER_NAME       (DN)  0x0  0x8001  19 1  0  0  "S=joyce@pwd.hp.com"
```

0x8000 - Internet form of addr
0x0001 - match DL names in FROM
0x0002 - match DL names in TO
0x0004 - match DL names in CC
0x0008 - match DL names in BCC
19 - name in DL
3 - Sender
17 - Recipient
1 - equals
string to search for

Below are some more examples of Search Specification files. The first is for a combined search on the `\Deleted` flag and the value of the `FROM` field in the `DL.` Again, the IMAP server determines the messages with the `\Deleted` flag set and sends the direct references for these in the Search Specification file:

```
Option?t
Ctr Handle (3003):
Entry Number:8
File Name: ~/data/000000a/000069g
HEADER            (DN)  1  0  2  1005
  ** Warning - Insufficient fields present  (Record No. 1)
SEARCH_REQUEST    (DN)  0x0  -1  1000
  ** Warning - Insufficient fields present  (Record No. 2)
SEARCH_TARGET     (DN)  "000102540c47ff92"
  ** Warning - Insufficient fields present  (Record No. 3)

...

SEARCH_TARGET     (DN)  "00010244259ec6e0"
  ** Warning - Insufficient fields present  (Record No. 8)
FILTER_START      (DN)  0x0  0x0  0
FILTER_BOOLEAN    (DN)  0x0  0x0  24 ◄──── \Deleted flag
FILTER_NAME       (DN)  0x0  0x8001  19  1  0  0  "S=Mr Admin"
                                     ◄──────────── look in DL
FILTER_END        (DN)
Done
```

Possible values for the last value in the `FILTER_BOOLEAN` record are:

18	MAPI attachment
21	Read/Unread
22	IMAP `\Flagged`
23	IMAP `\Draft`
24	IMAP `\Deleted`
25	IMAP msg `UID`
26	IMAP `\Answered`

Note: No `FILTER_BOOLEAN` record is created for a search on "recent" messages. This is because only the IMAP server (and not the Search server) knows whether a message is "recent".

The next example is for a combined search on the `\Answered` flag and the string "clever" in the body of the message:

```
Option?t
Ctr Handle (3003):
Entry Number:23
File Name: ~/data/000000a/000069v
HEADER            (DN)  1  0  2  1005
  ** Warning - Insufficient fields present  (Record No. 1)
```

```
SEARCH_REQUEST    (DN)  0x0  -1  1000
  ** Warning - Insufficient fields present  (Record No. 2)
SEARCH_TARGET     (DN)  "0001024853c9dc56"
  ** Warning - Insufficient fields present  (Record No. 3)
FILTER_START      (DN)  0x0  0x0  0
FILTER_BOOLEAN    (DN)  0x0  0x0  26 <-------- \Answered flag
FILTER_STRING     (DN)  0x0  0x2  20  1  0  0  "clever"
                                  ^---------------- look in body
  ** Warning - Insufficient fields present  (Record No. 6)
FILTER_END        (DN)
Done
```

As at B.05.10, once the search specification file has been constructed and submitted, the IMAP server waits for the notification of search completion. The notification itself is handled by the normal notification server mechanism. On receipt of the notification, the IMAP server prints the list of matching messages listed in the `SearchResults` object file.

With `IMAP_LOGLEVEL=8`, here is an excerpt from the UAL trace file for Mr Admin; `OM102U.log`, showing the UAL notification and object file commands/responses initiated by an IMAP search which requires the Search server:

```
COMMAND 350     ADD_NOTIFICATION        16:20:22
{
        UARef            = 16279
        SeqNo            = 49
        ListRef          =
        ItemRef          =
        DirectRef        = 00010e1070647dd5 <------ search spec file direct ref
        AddNFlags        = 0
        AddNEvents       = 128 <--------------- search complete notification
        Key              =
}

...

COMMAND 352     GET_NOTIFICATION        16:20:23
{
        UARef            = 1
        SeqNo            = 0
        Key              =
        GetNFlags        =
        MaxNumRecs       = 100
}

REPLY 352       GET_NOTIFICATION
{
        UARef            = 1
        SeqNo            = 0
        ReplyFlags       = 0
        ErrorNo          = 0
        Err2             = 0
        Err3             = 0
        NotifRef         = 3
        Key              = 0
        Event            = 128 <--------------- search complete
        DateTime         = 940605623
        Status1          = 0
        Status2          = 1 <--------------- number of hits
        TopChildDRef     = 0000000000000000
        Flags            = 0
        ExtraDRefs       =
}                       16:20:23
```

```
COMMAND 192     GET_OBJFILE     16:20:23
{
        UARef            = 16279
        SeqNo            = 50
        ListRef          =
        ItemRef          =
        Flags            = 2
        ObjectIndex      = SearchResults
        TargetFileName   =
        TargetFileId     = 210
        DirectRef        = 00010e1070647dd5 <—— direct ref of SearchResults obj file
}

REPLY 192       GET_OBJFILE
{
        UARef            = 16279
        SeqNo            = 50
        ReplyFlags       = 0
        ErrorNo          = 0
        Err2             = 0
        Err3             = 0
        ObjFileName      = /var/opt/openmail/user/u000036/TMPA16279/000124o
        FileSize         = 53
        ModDate          = 940605623
        ObjFlags         = 0
}                       16:20:23
```

Although searching has been improved, there are still some limitations:

- Unable to search personal names in addresses, for example, "Joyce Ford", because they are usually missing.
- `CHARSET` option is not supported. Only ASCII can be searched.
- Can search plain text and html but not RTF.
- With Search server used for text searches and IMAP server used for flag and date searches, the limited understanding of each server can cause problems in complex searches, like the following:

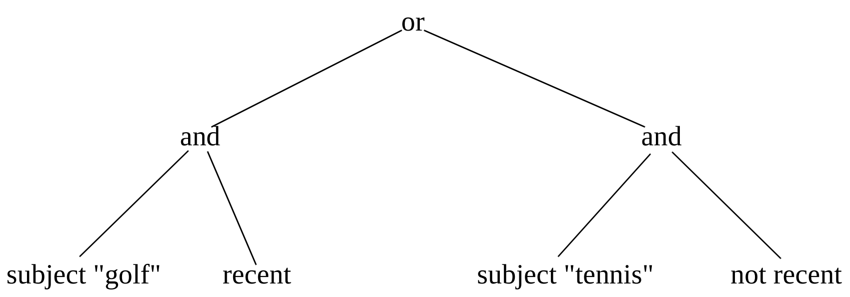
  Let us assume we have the following messages:

No.	*Subject*	*Flag*
1.	golf	
2.	tennis	
3.	golf	\Recent
4.	tennis	\Recent

  The client issues the following search request:

  > Find messages with subject "`golf`" and `\Recent` flag set or subject "`tennis`" and `\Recent` flag unset.

We could show these conditions using a parse tree:



The IMAP server understands:

`FLAGS` such as `\Answered` , `\Seen` etc.

`INTERNALDATE`

`ALL`

The Search server understands:

`FLAGS` such as `\Answered` , `\Seen` etc.

`INTERNALDATE`

most of the rest but ...

NOT `\Recent` flag

NOT most RFC822 headers. Some header information such as the value of `To`, `From`, `CC`, `BCC` and `Subject` are moved to OpenMail containers, which the Search server can search.

So the IMAP server will see the request as:

recent or not-recent

which will be TRUE for all messages! So it will pass direct reference for all the messages in the Search Specification file to the Search server.

The Search server will see the request passed to it as:

"subject golf" or "subject tennis"

which again will be TRUE for all messages.

So the result of this search request will be all the messages, when it should only be messages 2 and 3.

IMAP_SEARCH_TIMEOUT Tweak

Normally, a search will run until it completes. This tweak allows search requests to be abandoned after a specified number of seconds. Note that this merely abandons the search, it does not stop the Search server from running.

The default setting is

```
IMAP_SEARCH_TIMEOUT=0
```

This can be set for system, client or user. However, it is not advisable to set it under normal running conditions, as it is likely to lead to dead search specification files cluttering up the search folder.

Support For Internet Acks

Support for the Internet Acks, MDNs and DSNs, is discussed in the B.06.00 upgrade OTN, *Changes To The Internet Gateway.* However, there are two IMAP tweaks for Internet Acks; one to enable the IMAP server to generate the acks on behalf of the client, the other allows you to customise the name of the IMAP flag which gets set when an MDN or DSN has been sent for a message.

IMAP_AUTOMATIC_MDN

IMAP clients are supposed to be be able to generate Message Disposition Notification messages (MDNs, see *RFC 2298*). Since this is a new standard, many existing clients do not know what to do with the `Disposition-Notification-To:` header, let alone generate MDNs. If, this tweak is set to `TRUE`, the IMAP server will generate MDNs automatically when a message becomes read or is deleted before being read.

It can be set for the system, a client or a user and the default setting is:

```
IMAP_AUTOMATIC_MDN=FALSE
```

IMAP_MDNSENT_FLAG

This IMAP flag is set by the IMAP client or the IMAP server (see the description for `IMAP_AUTOMATIC_MDN`) to indicate that a Message Disposition Notification message (MDN) has been sent for this message. The flag shows up in the list of flags that can be set in response to an IMAP `SELECT` or `EXAMINE` command. If the flag name is empty, then nothing is advertised by the IMAP server.

At present, the name of the MDN sent flag has not been standardized and different clients may expect of different names. When, eventually, the name is standardized then the default value will be changed to reflect the standard.

Note that because *RFC 2298* specifies that an MDN must only be sent once for a given message, it is not possible to clear the `$MdnSent` flag once it has been set for a given message.

The tweak can be set for the system, a client or a user and the default setting is:

```
IMAP_MDNSENT_FLAG=$MdnSent
```

If, for example, you wanted to set the MDN sent IMAP flag to one of the earlier, now discredited, proposals for this flag, you could set the tweak as follows:

```
IMAP_MDNSENT_FLAG=\MdnSent
```

Security Issues

There have been numerous alerts about the security of freely available IMAP servers. The two most common forms of attack are buffer overflow and causing denial of service.

Buffer Overflow

This is usually caused by a very long argument, for example, a password, sent with some destructive programming tagged on the end. In this way, access can be gained to the stack.

To counter this, the OpenMail IMAP server performs size checks on everything that is sent to it, often before the data is retrieved.

Denial Of Service Attacks

This may happen when someone quickly starts up a large number of IMAP sessions which do nothing. The server quickly becomes overloaded and is unable to service any connection requests.

The following two tweaks are available to prevent this:

`IMAP_CONNECTION_LIMIT` - This system wide option enables you to set the maximum number of concurrent connections allowed at any one time. The default setting is

```
IMAP_CONNECTION_LIMIT=0
```

which means the IMAP server will accept as many connections as machine resources permit. If you set the value to greater than zero, the IMAP server will only allow the specified number of connections at any one time.

On a server with, potentially, a very high number of IMAP connections, it is advisable to set the `IMAP_CONNECTION_LIMIT` option. As one IMAP session can involve several connections and associated processes, the default setting (zero) allows the IMAP server to use all the available process slots, preventing any other users from accessing the server.

We mentioned earlier the problem of a single Netscape Messenger client session being able to have as many as 7 concurrent connections (i.e. 21 processes). With the current limit of 30,000 processes on an HP-UX server, this means a maximum configuration of 1428 Netscape Messenger clients at any one time. Obviously, for other clients that do not make so many connections during a session, the limit can be higher.

To increase the number of concurrent clients, you can configure Netscape as described in the section, "Limiting The Number Of New Processes Netscape Can Start In A Session" on page 58.

`IMAP_CONNRATE_LIMIT` This system wide option enables you to limit the number of connections in any one second. The default setting is

```
IMAP_CONNRATE_LIMIT=0
```

which means that the IMAP server will accept new connections as fast as it possibly can. If the `IMAP_CONNRATE_LIMIT` is greater than zero, the IMAP server will allow, at most, the specified number of connections in any one second.

For this option to take effect, you need to restart the IMAP server.

Warning! It is advisable to set this option on a server with a very high number of IMAP users, who all tend to login at the same time. The default setting of zero may mean that the system is devoted almost entirely to creating new IMAP processes. By setting the limit to, 3, for example, you will ensure that the IMAP server will only allow about 180 new connections per minute, which should be well within the capabilities of most servers.

The `IMAP_CONNRATE_LIMIT` tweak can be used together with the `UAKD_CONNRATE_LIMIT` tweak, which limits the rate at which connections can be made to the UAL. If you set both, we advise that you either set them to the same value or set `UAKD_CONNRATE_LIMIT` to slightly more than `IMAP_CONNRATE_LIMIT`.

Catching The Culprit

If you have problems with someone causing denial of service attacks, a useful tool is the openly available utility, `lsof` (List Open Files). You can obtain `lsof` from

```
ftp://vic.cc.purdue.edu/pub/tools/unix/lsof
```

To find the culprit, you could issue the command:

```
lsof -i:143
```

which not only lists the files being accessed visa port 143 (IMAP), but also lists the sender and receiver IP addresses.

What To Set When The UAL Server Is On A Different Machine

If your UAL server and IMAP server are running on different machines, you need to set the tweak, `IMAP_REMOTE_UAL_ENABLED`, to `TRUE`. This can be set for the system (`~/sys/general.cfg`), a client (`~/sys/client.cfg/`*`<client host machine name>`*) or a user (`~/sys/user.cfg/`*`<user OpenMail ID number>`*).

If set to `TRUE`, users can specify the name of a remote machine on which the UAL Server is running. The IMAP Server will then use this remote UAL Server.

Users specify the use of a remote UAL Server by connecting as *`<username>@<hostname>`*, where *`<hostname>`* is the name of the remote machine to which they want to connect.

For example, Mr Admin’s OpenMail account and the UAL server he is to use are on the machine called bean. The IMAP server he is to use is on kuda. To enable the use of a remote UAL server, I first added the option

```
IMAP_REMOTE_UAL_ENABLED=TRUE
```

to Mr Admin’s user specific configuration file, `~/sys/user.cfg/102`. (It is advisable to then `omoff` and `omon` the IMAP server).

From his client machine, he can then start a telnet IMAP session on kuda, which uses the UAL server on bean to access Mr Admin’s OpenMail account on bean:

```
#telnet kuda 143
Trying...
Connected to kuda.pwd.hp.com.
Escape character is '^]'.
* OK OpenMail IMAP server B.06.00.R0 ready on kuda
a1 login admin@bean admin
a1 OK LOGIN completed, now connected to bean.pwd.hp.com
a2 select inbox
* 3 EXISTS
* 0 RECENT
...
```

The IMAP/UAL logging settings and IMAP capabilities used in this session are those set up on the IMAP server machine (kuda). The IMAP log files will be on the IMAP server machine (kuda) and UAL trace files will be on the UAL server machine (bean).

Warning! If you set the test IMAP capabilities, `X-OPENMAIL` and `X-OMTEST` (see “Optional IMAP4rev1 Functionality Supported” on page 44), on the IMAP server host machine, these will be enabled automatically on the remote UAL server machine. The `X-OMTEST` capability allows a user to perform potentially dangerous operations, such as manipulating IMAP configuration settings and deleting files in the message store. To prevent such access on the remote UAL server machine, you have to disable these capabilities on the IMAP server machine.

The default setting of this option is

```
IMAP_REMOTE_UAL_ENABLED=FALSE
```

which will prevent users from conecting to a remote UAL Server.

Support For Imap4rev1 Functionality

The main functional changes between IMAP4 (*RFC 1730*) and IMAP4rev1 (*RFC 2060*) are the additions of a `STATUS` command and a new partial `FETCH` syntax

STATUS Command

The purpose of the `STATUS` command is similar to that of the `EXAMINE` command, in that it allows a client to find out the state of another mailbox; whether new messages have arrived, how many messages are unseen, the next UID to be assigned, etc.. Unlike `EXAMINE`, `STATUS` does not require a new connection to be made and does not cause the currently selected mailbox to be deselected.

The command syntax is:

```
STATUS <mailbox name> (<list of one or more data items>)
```

The mailbox name is obviously the name of the target folder and data items can be `MESSAGES`, `RECENT`, `UIDNEXT`, `UIDVALIDITY`, `UNSEEN`. Unlike `LIST`, `STATUS` does not accept wildcards.

In the example below, `inbox` remains the currently selected mailbox, even the `STATUS` request on the `mine1` mailbox. The status request is for the number of messages in the mailbox and the UID to be assigned to the next message that arrives:

```
a1 select inbox
* 18 EXISTS
* 0 RECENT ...
a1 OK [READ-WRITE] SELECT completed
a2 status mine1 (messages uidnext)
* STATUS mine1 (MESSAGES 3 UIDNEXT 4)
a2 OK STATUS completed
```

Examples of client use of this command are:

- Outlook Express, when you press `Send and Receive`.
- Netscape Messenger, when you press `File` -> `Update Message Count`.

As this command avoids the need for another connection to the IMAP server, it does lighten the load on the IMAP server. However, the UAL still has to do a `LIST` command on the mailbox to service the request, which can be costly.

Looking at a UAL trace on the server, you would see a `LIST` command with `ListFlags` set to 4, to force a reread of the container records. The example below shows part of the UAL trace for an IMAP `STATUS` command.

```
COMMAND 121     LIST    15:08:41
{
        UARef            = 7772
        SeqNo            = 47
        ListRef          =
        ListFlags        = 4 ◄─────────────── reread container records
        StartRecord      = 0
        MaxNumRecords    = 1
        Fields           = 4194554
        DirectRef        = 000102505e55ab6b
        MAPIFields       =
        PrepListFlags    =
        PrepListFilter   =
        NfFlags          =
        NfEvents         =
        NfKey            =
        FilterFile       =
}
REPLY 121       LIST
```

```
{
        UARef            = 7772
        SeqNo            = 47
        ReplyFlags       = 0
        ErrorNo          = 0
        Err2             = 0
        Err3             = 0
        ListStatus       = -1
        ItemRef          = 15
        Depth            = 1
        Subject          =
        ItemType         = -100
        CreateDate       =
        AttachDate       = 938614119
        Contents         = 2 ◄─────────────── no. of new records
        AbsoluteRef      = 28
        Flags1           = 131232
        Flags2           = 32768
        MsgFlags         = 4096
        ...
```

Fetch Part Of a Message

In IMAP4rev1, the `FETCH` command has been extended to let you specify more precisely the part(s) of the message body you want returned. In this section, I only intend giving an overview of the extended features of the `FETCH` command. For more detail, I suggest you read the `FETCH` command part of *RFC 2060*.

You can still have, for example:

`FETCH 1:* ALL`

to fetch all of the message or

`FETCH 1:* BODY`

to fetch the basic structure.

The extended form of the command is:

`FETCH` *`<msgs>`* `BODY[`*`<section>`*`]<`*`<partial>`*`>`

where

`<msgs>` is the message UID or range of messages, for example, 1:*

`[`*`<section>`*`]` is the part of the message body that you want. This can be one of `HEADER`, `HEADER.FIELDS`, `HEADER.FIELDS.NOT`, `MIME` or `TEXT` and/or a numeric message part number. Here's what *RFC 2060* says about part numbers:

"Multipart messages are assigned consecutive part numbers, as they occur in the message. If a particular part is of type message or multipart, its parts MUST be indicated by a period followed by the part number within that nested multipart part.

A part of type `MESSAGE/RFC822` also has nested part numbers, referring to parts of the `MESSAGE` part's body...

Here is an example of a complex message with some of its part specifiers:

`HEADER`	`([RFC-822] header of the message)`
`TEXT`	`MULTIPART/MIXED`
`1`	`TEXT/PLAIN`
`2`	`APPLICATION/OCTET-STREAM`
`3`	`MESSAGE/RFC822`
`3.HEADER`	`([RFC-822] header of the message)`

```
3.TEXT     ([RFC-822] text body of the message)
3.1        TEXT/PLAIN
3.2        APPLICATION/OCTET-STREAM
4          MULTIPART/MIXED
4.1        IMAGE/GIF
4.1.MIME   ([MIME-IMB] header for the IMAGE/GIF)
4.2        MESSAGE/RFC822
4.2.HEADER ([RFC-822] header of the message)
4.2.TEXT   ([RFC-822] text body of the message)
4.2.1      TEXT/PLAIN
4.2.2      MULTIPART/ALTERNATIVE
4.2.2.1    TEXT/PLAIN
4.2.2.2    TEXT/RICHTEXT"
```

Don’t worry about this now. We will look at messages and try fetching some parts later on.

<*`<partial>`*> is the starting octet, followed by a dot, followed by the number of octets required, for example, <0.128>

Below are some possible examples of `FETCH` commands.

The first returns 350 octets of message 1, starting at octet 0. Without the partial specifier, the whole message would be returned:

```
a1 FETCH 1 BODY[] <0.350>
```

The next example would return the text part of message 1, without headers:

```
a1 FETCH 1 BODY[TEXT]
```

The next would return 200 octets, beginning at 0, of the text part of all the messages in the mailbox:

```
a1 FETCH 1:* BODY[TEXT] <0.200>
```

The next would return the `FROM` and `DATE` header fields of messages 1, 2 and 3:

```
a1 FETCH 1:3 BODY[HEADER.FIELDS (FROM DATE)]
```

The final example would return the header within part 1 of message 2:

```
a1 FETCH 2 BODY[2.1.HEADER]
```

An Example

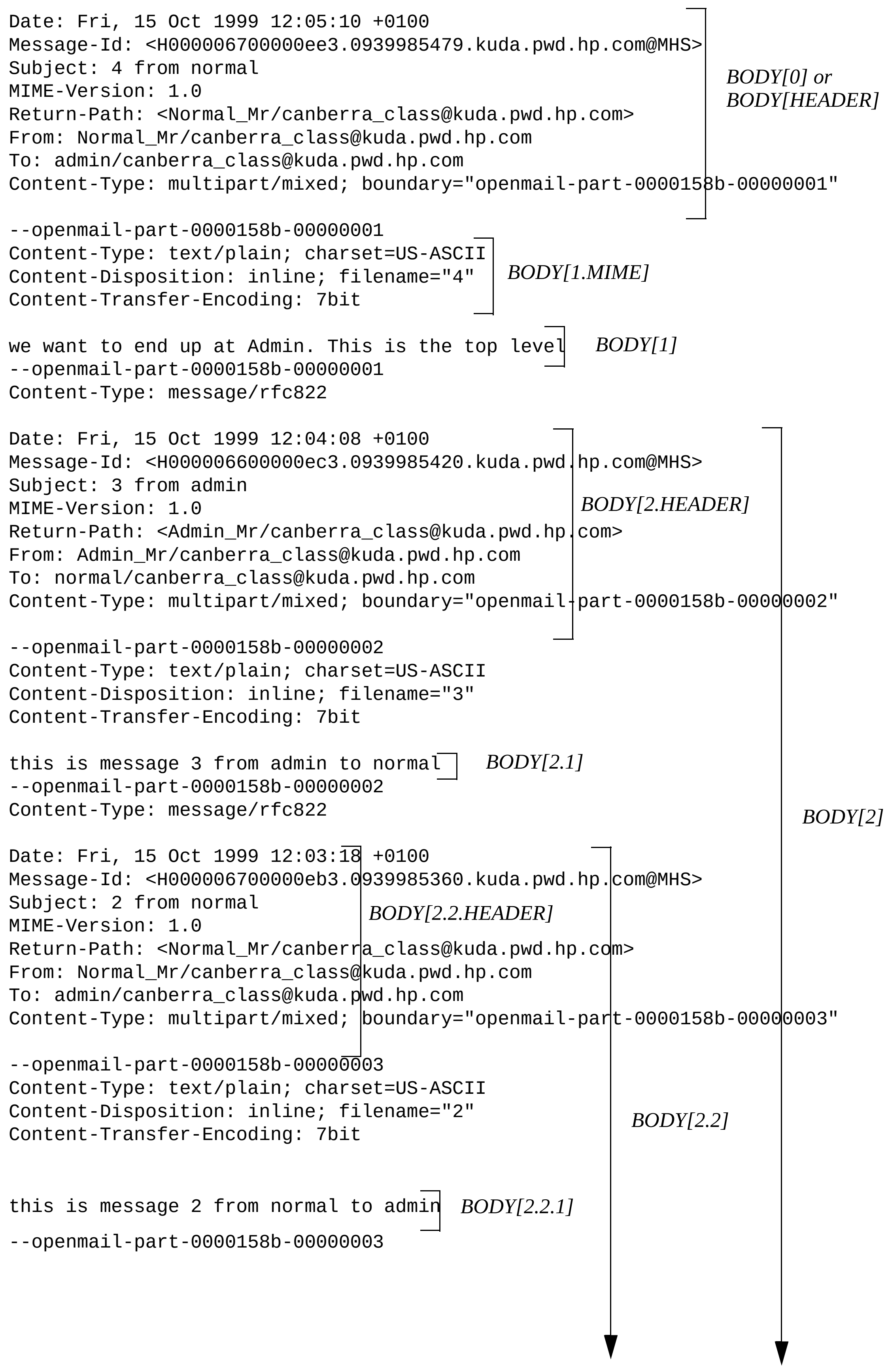
Here’s how to sort out IMAP part numbers in the `FETCH` command. Create a short text message and forward it to yourself 3 times. Use plain text not RTF at this point, just to keep it simple. In each of the forwarded messages use a different subject and add a little text. This is to help you to see the parts of the message.

Start a telnet’d IMAP session to access the final message.

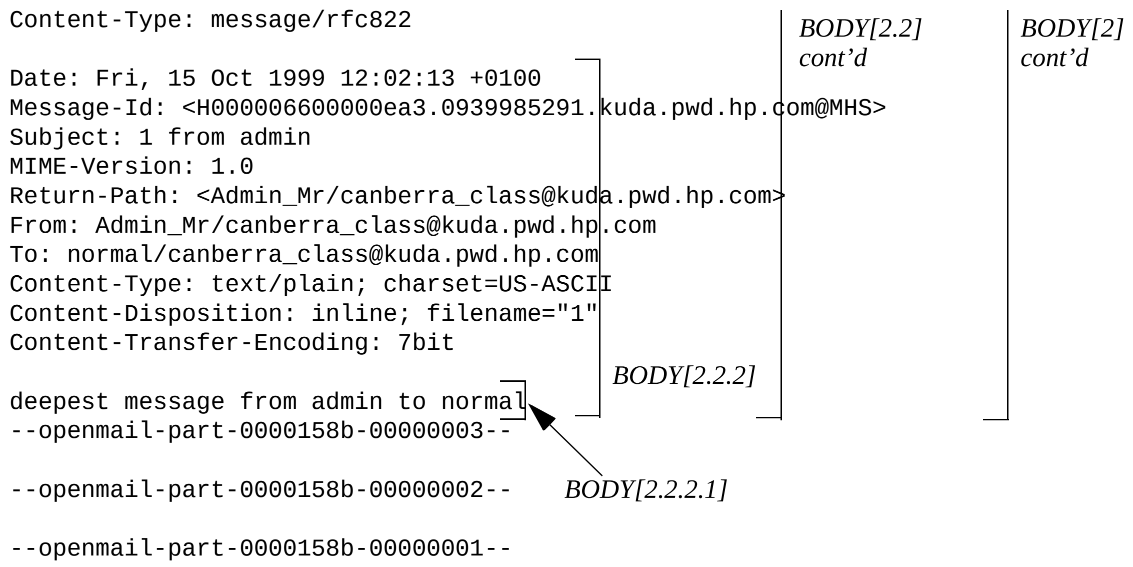
Fetch the whole message:

```
a3 fetch rfc822
```

On the next page is the result of this `FETCH` on the message I created.



```
Date: Fri, 15 Oct 1999 12:05:10 +0100
Message-Id: <H000006700000ee3.0939985479.kuda.pwd.hp.com@MHS>
Subject: 4 from normal
MIME-Version: 1.0
Return-Path: <Normal_Mr/canberra_class@kuda.pwd.hp.com>
From: Normal_Mr/canberra_class@kuda.pwd.hp.com
To: admin/canberra_class@kuda.pwd.hp.com
Content-Type: multipart/mixed; boundary="openmail-part-0000158b-00000001"

--openmail-part-0000158b-00000001
Content-Type: text/plain; charset=US-ASCII
Content-Disposition: inline; filename="4"
Content-Transfer-Encoding: 7bit

we want to end up at Admin. This is the top level
--openmail-part-0000158b-00000001
Content-Type: message/rfc822

Date: Fri, 15 Oct 1999 12:04:08 +0100
Message-Id: <H000006600000ec3.0939985420.kuda.pwd.hp.com@MHS>
Subject: 3 from admin
MIME-Version: 1.0
Return-Path: <Admin_Mr/canberra_class@kuda.pwd.hp.com>
From: Admin_Mr/canberra_class@kuda.pwd.hp.com
To: normal/canberra_class@kuda.pwd.hp.com
Content-Type: multipart/mixed; boundary="openmail-part-0000158b-00000002"

--openmail-part-0000158b-00000002
Content-Type: text/plain; charset=US-ASCII
Content-Disposition: inline; filename="3"
Content-Transfer-Encoding: 7bit

this is message 3 from admin to normal
--openmail-part-0000158b-00000002
Content-Type: message/rfc822

Date: Fri, 15 Oct 1999 12:03:18 +0100
Message-Id: <H000006700000eb3.0939985360.kuda.pwd.hp.com@MHS>
Subject: 2 from normal
MIME-Version: 1.0
Return-Path: <Normal_Mr/canberra_class@kuda.pwd.hp.com>
From: Normal_Mr/canberra_class@kuda.pwd.hp.com
To: admin/canberra_class@kuda.pwd.hp.com
Content-Type: multipart/mixed; boundary="openmail-part-0000158b-00000003"

--openmail-part-0000158b-00000003
Content-Type: text/plain; charset=US-ASCII
Content-Disposition: inline; filename="2"
Content-Transfer-Encoding: 7bit


this is message 2 from normal to admin
--openmail-part-0000158b-00000003
```



```
Content-Type: message/rfc822

Date: Fri, 15 Oct 1999 12:02:13 +0100
Message-Id: <H000006600000ea3.0939985291.kuda.pwd.hp.com@MHS>
Subject: 1 from admin
MIME-Version: 1.0
Return-Path: <Admin_Mr/canberra_class@kuda.pwd.hp.com>
From: Admin_Mr/canberra_class@kuda.pwd.hp.com
To: normal/canberra_class@kuda.pwd.hp.com
Content-Type: text/plain; charset=US-ASCII
Content-Disposition: inline; filename="1"
Content-Transfer-Encoding: 7bit

deepest message from admin to normal
--openmail-part-0000158b-00000003--

--openmail-part-0000158b-00000002--

--openmail-part-0000158b-00000001--
```

Start by fetching first level parts, e.g. `FETCH` *`n`* `body[0]` then `FETCH` *`n`* `body[1]` etc. I've outlined what these gave me in the example above.

Next, fetch the next level, e.g. `FETCH` *`n`* `body[2.1]` then `FETCH` *`n`* `body[2.2]` etc.

Carry on down the levels until you fetch the last text part of the nested message, which in the example above, is `body[2.2.2.1]`.

Now try introducing some names, e.g.:

```
FETCH n body[2.header]
FETCH n body[2.2.2.text]
FETCH n body[2.2.header.fields (subject)]
```

To get the MIME headers, precede MIME with the text part number, e.g. `body[1.mime]` above.

What's Happening On The Server?

Both the item browser and the IMAP server can return part of a message, so how do you know which one is returning the part you requested?

The diagram below shows repeated fetches on part [3.2] of message 1. When the client first issues a fetch on the message, the IMAP server doesn't have any cached information about this message, so it asks the item browser to render the message and place it in a particular file (it sends the item browser the direct reference of this file).

The item browser renders the whole message into the designated file and constructs the `imapBodyStrc` object file for this message.

The IMAP server then caches in its memory basic information about this message and maps into virtual memory the complete message file. It parses the complete message to find part [3.2], which it then sends to the client. In this case, it is the IMAP server which finds part [3.2] and returns it to the client.

The client then requests message 2 to be fetched, so the IMAP server cache now has details of message 2, not message 1.

Part [3.2] of message 1 is then requested again. The IMAP server no longer has that information in its cache, so it asks the item browser for information. The item browser then returns the imapBodyStrc object file for that message and the IMAP server parses it to find the "path" of part [3.2].

The IMAP server now hands the item browser the path to part [3.2] and a direct reference and asks the item browser to find the message part and put in the file indicated by the direct reference.

All the IMAP server needs to do, when the item browser indicates it is finished, is to pass the file to the client.

Client	**IMAP**	**UAL**	**Item Browser (BRW)**
FETCH 1 BODY[3.2]	---------------BODYSTRUCTURE------------>		
			Renders whole msg and generates imapBodyStrc
	Parse to get part [3.2]	<---Complete Msg----------	
<-----[3.2]-------------			
FETCH 1 BODY[3.2]			
<-----[3.2]------------- IMAP gets this from cache			
FETCH 2 RFC822 ...			
FETCH 1 BODY[3.2]	---------------BODYSTRUCTURE--------------->		
	Parse object file to find path of [3.2]	<--------------------------	Returns existing imapBodyStrc object file
	--------------------fetch "path [3.2]"-------------->		
			part 3.2 put in specified file
	<----------------------OK done that-------------------------		
<--part [3.2] file sent to client-------			

Later, we will look at proprietary OpenMail test extensions (`X-OMTEST`), which allow us to clear the IMAP server cache and/or delete the `imapBodyStrc` object file to force certain behaviour by the IMAP server or item browser.

The IMAP_USE_ITEM_BROWSER_SELECTION Tweak

If the item browser is having problems rendering parts of a message, you can set this tweak to

```
IMAP_USE_ITEM_BROWSER_SELECTION=TRUE
```

If set to true, the IMAP server will always ask the item browser to render the complete message, instead of parsing the message and asking the item browser to render a part. In this case, it will always be the IMAP server which parses the rendered message and extracts the requested part.

Some clients, such as older Netscape clients, require the whole message to be rendered even if only part is needed.

Asking the item browser to always render the complete message can be a considerable performance hit for some clients, so don't set this tweak in a normally running system unless you have to.

You can set it at system, client or user level.

Unicode Support

Support in OpenMail for the Universal Character Set (Unicode) is new at B.06.00. Both IMAP and LDAP include Unicode support.

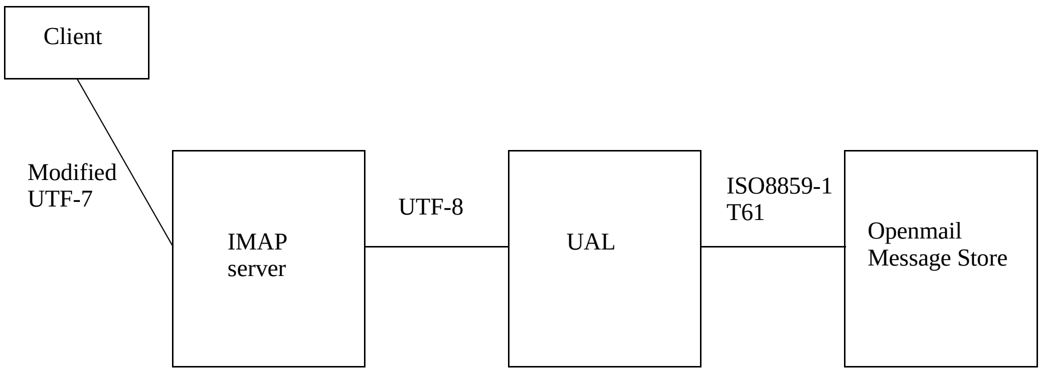
Unicode characters are encoded using various Transformation Formats (UTF). There are two used in IMAP; a modified version of UTF-7 for non-ASCII mailbox names and UTF-8 for other strings. The IMAP server handles conversion between Modified UTF-7 and UTF-8.

Note: When we talk about the use of UTF-7 or UTF-8 in relation to IMAP, we are referring to the character set used in the communication of the IMAP protocol only, not the character set used in any message content. The encoding of all message content is covered by MIME specifications.

The IMAP4rev1 specification is unclear about the treatment of MIME header field encoding and this may cause problems for some clients. In the OpenMail implementation, MIME headers are encoded according to MIME specifications. So any non-ASCII personal names in addresses in MIME headers will be MIME encoded. The encoding of personal names in addresses may also cause problems for the IMAP `SEARCH` request (see "Searching" on page 26).

Another area where the specification is unclear is the character set to be supported at login. The OpenMail IMAP server uses UTF-8 for login names, while Netscape Messenger and Outlook Express use ISO-8859-1. Therefore, if you are using one of these clients with OpenMail, a problem could arise if the IMAP login name includes a character outside the ASCII range.

The diagram below shows the character sets used in client and server communication during an IMAP session.



UTF-7 and Modified UTF-7

Internet mail (RFC 822) and many mail gateways and applications currently support only the 7 bit US-ASCII character set. UTF-7 encodes Unicode characters in 7 bit ASCII octets (shift sequences are used to for characters outside the US-ASCII range). As UTF-7 preserves

US-ASCII characters, it offers an encoding solution suitable for both multibyte and non-multibyte applications; clients that do not understand Unicode can still see US-ASCII characters.

As some of the characters used in UTF-7 conflict with the special characters used in some IMAP clients, a modified version of UTF-7 is used for non-ASCII mailbox names. Below is an extract from RFC 2152 describing the conflicting characters and the basic difference between UTF-7 and Modified UTF-7:

"UTF-7 uses the "+" character for shifting; this conflicts with the common use of "+" in mailbox names, in particular USENET newsgroup names.

UTF-7's encoding is BASE64 which uses the "/" character; this conflicts with the use of "/" as a popular hierarchy delimiter.

UTF-7 prohibits the unencoded usage of "\"; this conflicts with the use of "\" as a popular hierarchy delimiter.

UTF-7 prohibits the unencoded usage of "~"; this conflicts with the use of "~" in some servers as a home directory indicator.

UTF-7 permits multiple alternate forms to represent the same string; in particular, printable US-ASCII chararacters can be represented in encoded form.

In modified UTF-7, printable US-ASCII characters except for "&" represent themselves; that is, characters with octet values 0x20-0x25 and 0x27-0x7e. The character "&" (0x26) is represented by the two- octet sequence "&-".

All other characters (octet values 0x00-0x1f, 0x7f-0xff, and all Unicode 16-bit octets) are represented in modified BASE64, with a further modification from [UTF-7] that "," is used instead of "/". Modified BASE64 MUST NOT be used to represent any printing US-ASCII character which can represent itself.

...

"&" is used to shift to modified BASE64 and "-" to shift back to US- ASCII. All names start in US-ASCII, and MUST end in US-ASCII (that is, a name that ends with a Unicode 16-bit octet MUST end with a "- ")."

UTF-8

As already mentioned, with the exceptions of international mailbox names and MIME encoded headers, all other strings in the IMAP protocol are in the UTF-8 character set.

UTF-8 encodes Unicode characters as a varying number of octets. As with UTF-7, US-ASCII characters are also valid UTF-8 characters; character values from 0000 0000 to 0000 007F (US-ASCII range) correspond to octets 00 to 7F (7 bit US-ASCII values). So clients which do not support Unicode can still see ASCII text.

What Can We See On The Server?

To see the conversation between the IMAP server and UAL server, when the IMAP server wants to set up a UAL connection, turn on UAL symbolic tracing for an IMAP connection as follows:

- Set `IMAP_UAL_TRACE_LEVEL=2` in a file in the directory, `~openmail/sys/client.cfg`. The name of the file should be the fully qualified name of the client machine. For example, my client machine is `bean`, so I set up a file, `bean.pwd.hp.com`.
- From the client machine, start an IMAP session on the server, for example:

  ```
  telnet kuda 143
  ```
- Login to a user:

  ```
  a1 login admin admin
  ```

Look in `~openmail/tmp` and you should see a trace file with a name of the form:

UX*nnnnn*C.trc

where *nnnnn* is either the Unix user ID number or the OpenMail User ID number, if the user was created using a pool ID (for more information on OpenMail pool IDs, see the OTN, *Changes to OpenMail user creation and management at B.06.00*).

Below is an extract from my trace file. You can see UTF-8 specified by the IMAP server as the character set it will use during UAL session initialisation:

```
UAL Server Command Trace at Wed Oct 20 15:53:37 1999
--------------------------------------------------------
INIT Command: UA Id='IMAP4 Server B.06.00.R0', Debug Level=2, Debug
File=''
INIT Command: Flags=0x20020, CharSet='UTF8', Service Level=6
INIT Command: User Name = 'Admin^]Mr^]^]^]canberra^]class'
INIT Reply:   Version='B.06.00.R0', Features=0x6ff, GMT Offset=3600, Host
Name='kuda'
INIT Reply:   Host File Name = '/var/opt/openmail/tmp/UAL15294'
...
```

Some References

The following references may prove useful, if you need to look further into Unicode or the way it works with MIME:

"*The Unicode Standard, Version 2.0*", The Unicode Consortium, Addison-Wesley, 1996. ISBN 0-201-48345-9.

ISO/IEC 10646-1:1993(E) Information Technology--Universal Multiple-octet Coded Character Set (UCS). See also amendments 1 through 7, plus editorial corrections.

MIME - (*RFC 2045* through *2049*)

UTF-7 - *RFC 2152*

UTF-8 - *RFC 2279*

Unicode with MIME - *RFC 1641*

Summary Of Differences Between IMAP4 And IMAP4rev1

The following list just summarizes the differences between *RFC 1730* (IMAP4) and *RFC 2060* (IMAP4rev1):

- A server that supports IMAP4rev1 will include the string " `IMAP4rev1`" in response to the `CAPABILITY` command. A server that also provides backwards support for IMAP4, should include the string "`IMAP4`" in addition to "`IMAP4rev1`".
- The `STATUS` command has been added.
- To support fetching parts of messages, various part specifiers have been added

  ```
  HEADER
  HEADER.FIELDS
  HEADER.FIELDS.NOT
  MIME
  TEXT
  ```

  In addition, the "<" *`origin`* "." *`size`* ">" suffix for `BODY` text attributes has been added to allow you to specify the starting point and number of octets to be fetched.
- You can now fetch nested MULTIPART parts of a message.
- Server supported authenticators, such as `AUTH=LOGIN`, are now identified by capabilities.
- The `PARTIAL` command is obsolete
- The following `FETCH` attributes are obsolete

`RFC822.HEADER.LINES`

`RFC822.HEADER.LINES.NOT`

`RFC822.PEEK`

`RFC822.TEXT.PEEK`

- Body part number 0 is obsolete

A full list of obsolete syntax is given in *RFC 2062*.

The OpenMail IMAP server maintains compatibility with IMAP4 (*RFC 1730*), although the origunal Nokia Communicator 9000, which uses `PARTIAL`, and Eudora Pro appear to be the only strictly IMAP4 clients.

Optional IMAP4rev1 Functionality Supported

In this section, we will look at the optional extensions to the IMAP4rev1 standard, which are supported by the OpenMail IMAP server at B.06.00. These can be enabled using the tweak, `IMAP_CAPABILTIES`.

To find out what extensions a server supports, a client issues a `CAPABILITY` command. Here is the response to a `CAPABILITY` command on a default B.06.00 setup:

```
root@bean[] #telnet bean 143
Trying...
Connected to bean.pwd.hp.com.
Escape character is '^]'.
* OK OpenMail IMAP server B.06.00.00 ready on bean
a2 capability
* CAPABILITY IMAP4 IMAP4rev1 AUTH=LOGIN NAMESPACE IDLE
a2 OK CAPABILITY completed
```

Possible values that can be returned by the B.06.00 IMAP server are:

- `IMAP4`
- `IMAP4rev1`
- `AUTH=LOGIN`
- `NAMESPACE` (*RFC 2342*)
- `IDLE` (*RFC 2177*)
- `LITERAL+` (*RFC 2088*)
- `CHILDREN`
- `X-NETSCAPE`
- `X-OPENMAIL`
- `X-OMTEST`

The default value of the tweak is:

```
IMAP_CAPABILTIES=IMAP4 IMAP4rev1 AUTH=LOGIN NAMESPACE IDLE
```

Although, IMAP capabilities can be configured on a system-wide, client or user basis, changing the set of capabilities between initial connection and becoming authenticated is contrary to *RFC 2060* and may confuse some clients. Most clients issue a `CAPABILITY` request when they first connect and use whatever capabilities are advertised. For this reason, the IMAP server only allows the set of capabilities advertised and supported to be extended, not reduced.

So, if we want to give one user, Mr Admin (OpenMail user ID, 102), the additional capability, `X-OPENMAIL`, while other IMAP connections across the system use the default capabilities, we can add the line

```
IMAP_CAPABILTIES=X-OPENMAIL
```

to Mr Admin's user configuration file, `~openmail/sys/user.cfg/102`. Notice that the system wide capabilities are assumed. We only have to specify the additional extension.

To see the effect of this, we can start a telnet session and issue the `CAPABILITY` command twice; once when the session starts, to show what extensions are available system wide, and again when Mr Admin is authenticated, to show the additional extension available to this user.

```
root@kuda[sys] #telnet kuda 143
Trying...
Connected to kuda.pwd.hp.com.
Escape character is '^]'.
* OK OpenMail IMAP server B.06.00.R0 ready on kuda
a2 capability
* CAPABILITY IMAP4 IMAP4rev1 AUTH=LOGIN NAMESPACE IDLE
a2 OK CAPABILITY completed
a3 login admin admin
```

```
a3 OK LOGIN completed, now connected to kuda.pwd.hp.com
a4 capability
* CAPABILITY IMAP4 IMAP4rev1 X-OPENMAIL AUTH=LOGIN NAMESPACE IDLE
a4 OK CAPABILITY completed
```

At the time of writing, Netscape Messenger 4.6 and OutLook Express 5 support the following capabilities:

```
IMAP4 IMAP4rev1 IDLE X-NETSCAPE NAMESPACE AUTH=LOGIN
```

As we shall see, the extent to which they support `NAMESPACE` is debatable.

The following sections give a description of what each of the extensions offer.

IMAP4

This capability indicates that the IMAP server supports the basic protocol as defined in *RFC 1730*. Note that the IMAP4 commands defined in *RFC 1730* but absent in *RFC 2060* are still supported even if this capability is not advertised.

IMAP4rev1

This indicates that the server supports the basic protocol as defined in *RFC 2060*. The B.06.00 IMAP server always advertises this capability, regardless of tweak settings.

AUTH=LOGIN

The IMAP protocol allows for two different styles of authentication:

- Using a `LOGIN` request that simply passes the user name and password in clear to the IMAP server
- An `AUTHENTICATION` request that allows a more secure credentials exchange according to some mechanism advertised by the IMAP server

The only "secure" authentication mechanism offered by the IMAP server at the moment is a non-standard `AUTH=LOGIN` mechanism, which encodes the username and password in base64. This mechanism is used by Netscape and Outlook Express and offers a degree of protection, but is still not very secure.

Both authentication mechanisms pass the username and password unchanged (apart from base64 decoding) to the UAL server for authentication.

One potential problem here is that the UAL server us told that the client character set is UTF-8, so the username (at least) is presumed to be in UTF-8. Unfortunately, the IMAP specification is silent about the character set used for login names and different clients assume different character sets. Some clients, for example, assume that the character set is ISO 8859-1 (Netscape and Outlook Express) and trying to log in with any user name that includes non-ASCII characters will necessarily fail.

CHILDREN

The `CHILDREN` extension is the subject of an Internet draft, *draft-gahrns-imap-child-mailbox-01.txt*. The purpose of the extension is to provide an efficient way of letting clients know if a mailbox has child mailboxes.

If the client issues an initial `LIST "" %` to list the first level mailboxes, the IMAP server returns the attribute `\HasChildren` or `\HasNoChildren` to indicate the existence of nested mailboxes accessible to the current, authenticated user. This saves the client having to issue a `LIST "" *` or `LIST "" %` on each mailbox to find out if there are nested folders. This can be a significant overhead if there are a lot of mailboxes.

Here is the server response I get from a `LIST "" %` on my machine, with the `CHILDREN` capability enabled:

```
* OK OpenMail IMAP server B.06.00.R0 ready on kuda
a1 login admin admin
a1 OK LOGIN completed, now connected to kuda.pwd.hp.com
a2 list "" %
* LIST (\NoInferiors \HasNoChildren) "/" INBOX
* LIST (\HasNoChildren) "/" Meetings
* LIST (\HasNoChildren) "/" Products
* LIST (\HasChildren) "/" Personel
* LIST (\HasNoChildren) "/" Admin
a2 OK LIST completed
```

Don't confuse `\HasNoChildren` with the `\NoInferiors` attribute. `\HasNoChildren` means there are currently no nested mailboxes below the mailbox. `\NoInferiors` indicates that nested mailboxes can never be created below the mailbox.

IDLE

If this capability is advertised, then the server will support the `IDLE` extension as defined in *RFC 2177*. This extension allows a client to tell the server that it is not busy but wants to be kept updated on changes to to selected mailbox.

In IMAP4, the client had to poll the server in order to be updated on new mail, deletions etc. In terms of what was going on on the server - both the IMAP server process and the associated UAL process were "woken up" to find out what had changed. To see the impact of this, consider a system with 10,000 IMAP users logged on, but not doing very much. Each client polling every 5 minutes, on average. The result is 20,000 processes (one IMAP server process and one UAL process for each client connection) starting up every 5 minutes. In most cases, all the processes will do is report to the client that nothing has changed.

With the `IDLE` capability enabled, the client no longer polls the server. Instead, when the client is not busy, it issues an `IDLE` command, which signals to the server that it can send unsolicited responses to keep the client informed of changes to the mailbox.

To see this in action, try the following:

- Turn on logging by setting the option `IMAP_LOGLEVEL=8`.
- Restart the IMAP server:

  ```
  omoff -d0 -a imap
  omon -a imap
  ```
- Start an IMAP session by telnet'ing to port 143.
- `LOGIN` to a user then select his inbox.
- Issue an IDLE command:

  a1 `IDLE`
- Start up `OMGUI` and login to the same user.
- Send a message to youself (ie the user you are logged on as).
- Look at the message, then delete it.
- Send the string `"DONE"` to end the `IDLE` command.
- Close OMGUI and `LOGOUT` of the telnet session.
- Go to `~openmail/tmp` and look at the appropriate log file. I used the default setting for `IMAP_LOGFILE` and my client machine was `kuda.pwd.hp.com`, so the log file was `~openmail/tmp/imap.kuda.pwd.hp.com`.

Below is an excerpt from my log file:

```
11847 14:24:17 C: a1 login admin admin
11847 14:24:26 S: a1 OK LOGIN completed, now connected to kuda.pwd.hp.com
11847 14:24:36 C: a2 select inbox
```

```
11847 14:24:36 S: * 21 EXISTS
* 0 RECENT
* OK [UIDVALIDITY 1] UIDVALIDITY value
* FLAGS (\Answered \Flagged \Deleted \Seen \Draft $MdnSent)
* OK [PERMANENTFLAGS (\Answered \Flagged \Deleted \Seen \Draft $MdnSent)]
flags will stay set
11847 14:24:36 S: a2 OK [READ-WRITE] SELECT completed
11847 14:26:29 C: a3 idle ◄────────────── client issues IDLE command
11847 14:26:30 S: + idling
11847 14:28:02 S: * 22 EXISTS ◄────────────── new mail arrived
11847 14:28:02 S: * 1 RECENT
11847 14:28:20 S: * 22 FETCH (FLAGS (\Seen \Recent))◄────── new mail read
11847 14:28:38 S: * 22 EXPUNGE ◄──────── new msg deleted
11847 14:28:38 S: * 21 EXISTS
11847 14:28:38 S: * 0 RECENT
11847 14:29:01 C: DONE◄────── client exits IDLE command
11847 14:29:01 S: a3 OK IDLE completed
```

When the client sends the `IDLE` command, the server initially responds with `+ idling`.

During the minute or so following the `IDLE` command, while the client is inactive, the server sends unsolicited responses to tell the client that a new message has arrived:

```
11847 14:28:02 S: * 22 EXISTS
11847 14:28:02 S: * 1 RECENT
```

Then, when the new message has been read by the other user accessing this mailbox using OMGUI, the server sends:

```
11847 14:28:20 S: * 22 FETCH (FLAGS (\Seen \Recent))
```

When the other user then deletes the new message, the server sends:

```
11847 14:28:38 S: * 22 EXPUNGE
11847 14:28:38 S: * 21 EXISTS
11847 14:28:38 S: * 0 RECENT
```

Before issuing any further commands, the client interrupts by sending any string, in this case "DONE", to tell the server that the `IDLE` command is no longer in operation.

The IMAP_IDLE_TIMEOUT Tweak

This tweak allows you to set the number of minutes an IMAP connection may remain idle before the connection is closed by the IMAP server. The default setting is 30 minutes. Any non-negative integer is permitted. A value of zero (0) will disable the timeout completely.

Some clients, notably Outlook Express, wait exactly thirty minutes between commands and may, therefore, get logged out prematurely. For these clients, it is sometimes useful to set the idle timeout to 31 minutes:

```
IMAP_IDLE_TIMEOUT=31
```

Generally, the idle timeout will be set on a system-wide basis, although it may be set differently for specific users and/or specific client machines to meet local needs.

For compatability with B.05.10, the old tweak name, `UAL_IMAP4_TIMEOUT`, will be used if `IMAP_IDLE_TIMEOUT` is not defined.

What Is Happening On The Server?

The `IDLE` extension enables "server push" notifications on the OpenMail server. This feature was introduced at release B.05.20 of the MAPI Service Providers for Outlook on OpenMail. Rather than repeat the detailed discussion here, I suggest you take a look at the section on Notifications in the OTN, *OpenMail Outlook (MAPI) Support [web ID: 300-0154]*.

Now, instead of the idling client regularly polling the server and waking up the UAL and IMAP processes to see what has changed, the client does not contact the server, except perhaps a `NOOP` command once every 30 minutes to ensure it is not disconnected. All that is required to service these occasional requests is for the IMAP server process to respond that nothing has changed. The UAL process is not involved and can *sleep* until something happens to the mailbox, for example, new mail arrives. When something happens, the Notification Server will send the UAL process unsolicited notifications to be sent to the client via the IMAP server.

The `IDLE` extension is used by Outlook Express and gives significant performance benefits for that client.

To see the effect of server push notifications from the end user's point of view:

- Open Outlook Express, Outlook 98 and OMGUI and log into the same user.
- Select his inbox.
- In Outlook Express, create or delete a couple of messages and you will see the changes immediately in Outlook 98 (because, as we have said, the MAPI service providers use server push notifications).

  OMGUI does not use server push notifications, so the changes will not be visible in OMGUI until the client next polls the server.

LITERAL+

This capability indicates support for "non-synchronizing" literals, as defined in *RFC 2088*. This is an attempt to reduce traffic, by reducing the need for the server to respond before the rest of input can be sent.

For example, a client could send login data in sections. Without the `LITERAL+` extension, the conversation would look something like this:

```
17:39:05.38:C: a1 login {5}
17:39:06.39:S: + Continue please
17:39:16.73:C: admin {8}
17:39:16.74:S: + Continue please
17:39:19.13:C: Maty6_45
17:39:28.45:S: a1 OK LOGIN completed, now connected to kuda.pwd.hp.com
```

`admin` is the user name, `Maty6_45` is the password. `{5}` and `{8}` specify the length, in octets, of the next bit of data (i.e. literal) that the server is to expect.

Adding "+" immediately after the octet count indicates that the `LITERAL+` feature is being used. The above communication can then be speeded up as follows:

```
17:45:33.60:C: a1 login {5+}
17:45:50.95:C: admin {5+}
17:45:55.70:C: admin
17:45:59.90:S: a1 OK LOGIN completed, now connected to kuda.pwd.hp.com
```

While this is a supported and standards track capability, it should be used with caution as it leaves the server open to denial-of-service attacks (see "Denial Of Service Attacks" on page 31).

NAMESPACE

The purpose of the `NAMESPACE` capability is to simplify client configuration by enabling clients to find out the prefix and delimiters used by the server for the namespaces available to the current, authenticated user. The standard defines three types of namespace:

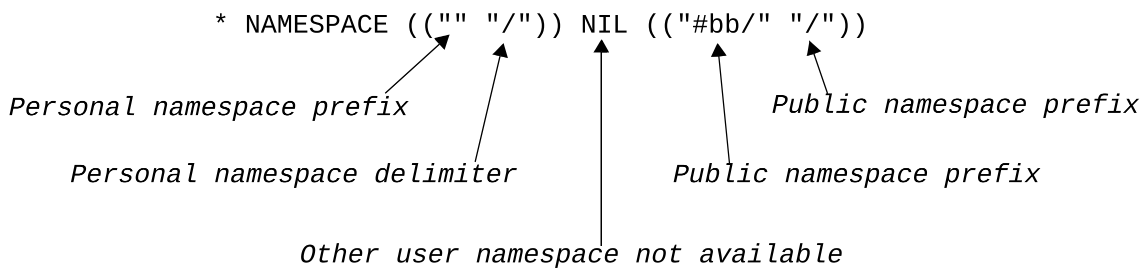
- Personal
- Other users
- Public

A user's inbox and personal folders are in the personal namespace. The other users' namespace defines the other users' mailboxes, to which this user has access. The public namespace defines the publicly shared mailboxes, such as OpenMail bulletin boards. The OpenMail IMAP server only supports personal and public namespaces.

The `NAMESPACE` command is defined in *RFC 2342*. As the command relates to a user, the session needs to be in authenticated state before the `NAMESPACE` command is issued. Here's an example of the command and server response:

```
a1 login admin admin
a1 OK LOGIN completed, now connected to kuda.pwd.hp.com
a1 namespace
* NAMESPACE (("" "/")) NIL (("#bb/" "/"))
a1 OK NAMESPACE completed
```

The server response gives the following information:


* NAMESPACE (("" "/")) NIL (("#bb/" "/"))
Personal namespace prefix
Public namespace prefix
Personal namespace delimiter
Public namespace prefix
Other user namespace not available

These are the default settings. There are several tweaks available to enable you to change the settings:

- `IMAP_FOLDER_PREFIX` can be used to change the prefix string which precedes all personal folder names.
- `IMAP_BB_PREFIX` can be used to change the prefix string which precede all bulletin board names.
- `IMAP_FOLDER_SEPARATOR` can be used to change the separator used in folder names in the personal namespace.
- `IMAP_BB_SEPARATOR` can be used to change the separator used in bulletin board names.

IMAP_FOLDER_PREFIX and IMAP_BB_FOLDER_PREFIX

These options, which can be set for the system, client or user, have the following default values:

```
IMAP_FOLDER_PREFIX=
IMAP_BB_FOLDER_PREFIX=#bb
```

The folder prefix string precedes all mailbox names for private folders. The BB folder prefix precedes all mailbox name for bulletin board (public) folders. These prefixes must be distinct and must not contain the folder separator characters defined in either `IMAP_FOLDER_SEPARATOR` or `IMAP_BB_FOLDER_SEPARATOR`.

In addition, the strings should be ASCII rather than modified UTF-7 or UTF-8. This means restricting the prefix to printable ASCII characters and avoiding the character "&".

Configuring Netscape Messenger To See Bulletin Boards

Despite the fact that Netscape Messenger supports the `NAMESPACE` capability, it does not recognise anything with # in it, so does not recognise the default bulletin board namespace, which has the prefix #bb.

To enable Netscape Messenger to see existing OpenMail bulletin boards:

On the server:

- Change the value of `IMAP_BB_FOLDER_PREFIX` to something that does not include a "#" character, for example:

  ```
  IMAP_BB_FOLDER_PREFIX=Bulletin Boards
  ```
- omoff/omon the IMAP daemon.

In Netscape Messenger:

- Select Edit -> Preferences -> Mail Servers
- Select the required mail server and press Edit.
- Go to the Advanced tab.
- In the field for the name of the Public (shared) namespace, type `"Bulletin Boards"`.
- Restart Messenger and the Bulletin Board folder should appear in the folder hierarchy.

IMAP_FOLDER_SEPARATOR and IMAP_BB_FOLDER_SEPARATOR

These options, which can be set for the system, client or user, have the following default values:

```
IMAP_FOLDER_SEPARATOR=/
IMAP_BB_FOLDER_SEPARATOR=/
```

The folder separator is the character used to separate folder names in an IMAP mailbox specification. For example, with a separator of / and a folder, `kanga`, with a nested folder, `roo`, the corresponding IMAP mailbox name is `kanga/roo`.

There are two distinct folder separators, one for private folders and one for bulletin board (public) folders. To make sure your OpenMail folders are visible to IMAP clients, the separator must be something that does not appear in the folder names. For example, if an existing OpenMail user has a folder called `Sales/Forecasts`, this folder will not be seen through the IMAP server, because it cannot be distinguished from a folder called `Forecasts` within a folder called `Sales`.

The folder separator is a single ASCII character, anything in the range 1 to 127. Although you can choose any ASCII character, choosing letters like `e` is not generally advised. Indeed, choosing anything from the modified base64 alphabet (0-9, a-z, A-Z, "+" and ",") or "&" or "-" is not recommended. Also, using the characters "%" and "*" may cause problems since these characters are special to the IMAP protocol.

To avoid clashes with characters that can reasonably appear in folder names, you could specify a control character as the separator, for example, Ctrl-A, although some IMAP clients may be upset by this.

RFC 2683, *IMAP4 Implementation Recommendations*, recommends that the separator character should only be "/", "\" or ".". The character must be represented in the configuration file literally, sequences like `\x65` are not supported.

For compatability with B.05.10, the tweak name `UAL_IMAP4_SEPARATOR` may be used if `IMAP_FOLDER_SEPARATOR` is not defined.

X-NETSCAPE

This capability provides support for the non-standard `NETSCAPE` command used by the Manage Mail Account feature in Netscape Messenger (Communicator -> Server Tools -> Mail Account). Used together with the tweak, `IMAP_X_NETSCAPE_URL`, the `X-NETSCAPE` capability allows a customized mail management web profile to be used with Netscape Messenger. The intent is that the page referred to by the URL can be used to set passwords, define auto-action rules etc.

So, in addition to adding `X-NETSCAPE` to `IMAP_CAPABILITIES`, you also need to specify a URL for the tweak `IMAP_X_NETSCAPE_URL`. For example, the tweak could point to an OpenMail web client profile on my machine, kuda:

```
IMAP_X_NETSCAPE_URL=http://kuda.pwd.hp.com/cgi-bin/ice.cgi/prof=ICE
```

If the `X-NETSCAPE` capability is not defined, then this tweak has no effect. If the `IMAP_X_NETSCAPE_URL` is not defined, then the `X-NETSCAPE` capability is implicitly turned off.

The capability can be set for the system or specific clients or users.

When the user selects the Mail Account option in Netscape Messenger, the client issues the `NETSCAPE` command and the IMAP server returns the URL defined by `IMAP_X_NETSCAPE_URL`. The IMAP server doesn't actually do anything with the URL and doesn't care what it points to. Here's an IMAP trace of what happens:

```
12181 16:50:12 S: 1 OK AUTHENTICATE completed, now connected to
kuda.pwd.hp.com
12181 16:50:12 C: 2 netscape
12181 16:50:12 S: * OK [NETSCAPE]
* VERSION 1.0 UNIX
* ACCOUNT-URL http://kuda.pwd.hp.com/cgi-bin/ice.cgi/prof=ICE
12181 16:50:12 S: 2 OK NETSCAPE completed
```

Since `IMAP_X_NETSCAPE_URL` can be specified on a per-user basis, it is possible to have a separate URL for each user on a machine. For example, Mr Admin's URL might be specified in his configuration file, `~openmail/user.cfg/102`, as

```
IMAP_X_NETSCAPE_URL=http://www.kuda.pwd.hp.com/~admin/mail.html.
```

X-OPENMAIL

Extensions providing useful IMAP testing tools are provided under two OpenMail proprietary capabilities: `X-OPENMAIL` and `X-OMTEST`. Either or both capabilities can be given to any OpenMail user. Without administrator privileges, the user only has access to his own messages in the message store.

The extensions under the `X-OPENMAIL` extension are "safe" and can be present in a production server. The extensions under the `X-OMTEST` capability should not be present in a production server as they circumvent some security checks. These extensions are not, as yet, used by any client.

In this section we will look at the `X-OPENMAIL` extensions. `X-OMTEST` extensions are covered in the next section.

Folder Syntax Extension.

With `X-OPENMAIL` enabled, folder names can be represented as an IMAP list rather than as a string with folder separators. The first element of the list is the namespace prefix. The folder name, `kanga/roo`, represented as an IMAP list would be `("" "kanga" "roo")`, assuming that the namespace prefix for the folder is the empty string and that the folder separator is /. The strings in these lists are UTF-8 rather than modified UTF-7.

Using this extension the folder `Sales/Forcasts` will be visible, even when the folder Separator is /. For example, we could use the list format to specify the folder in the IMAP `SELECT` or `LIST` command:

```
SELECT ("" "Sales/Forcasts")
```

where "" is the namespace prefix for private folders.

Note that the command `CREATE ("" "x/y")` will create an almost invisible folder if the folder separator is /.

The `LIST` and `LSUB` commands can be used with list form folder names. The first argument is a `LIST` folder name and the second argument is a maximum depth. So, for example,

```
LIST ("" kanga roo) 1
```

will list all the immediate children of the specfied folder.

```
LIST ("" kanga roo) 2
```

will list list grandchildren as well.

```
LIST ("" kanga roo) 0
```

will list all the descendants.

Lab

- Set X-OPENMAIL capability for the user, Mr Admin.
- Test that X-OPENMAIL extension works when set in user.cfg. Remember you have to login to user then send CAPABILITY command to see it. Have you restarted the IMAP daemon?

Seeing folder names in modified UTF-7 and UTF-8:

1. Use IMAP telnet to create folder X&AM8-Y
2. Use MAPI(Outlook) to view this folder. What do you see?
3. Use Folder syntax extension to view folder name. What do you see? Why is it different from MAPI(Outlook) view?

Note: "&" is used to shift to modified BASE64 and "-" to shift back to US- ASCII.

Seeing UTF 7 folder names:

1. Use MAPI to create folder X&AM8-Y
2. Look at it in IMAP. What do you see? Why is it different?
3. Use Folder syntax extension and look again. Again, what do you see and why?

Direct Reference Access

With `X-OPENMAIL` enabled, the `FETCH` command takes an additional argument, `DIRECTREF`, to fetch the OpenMail direct reference of a message:

```
a1 fetch 13 (directref)
* 13 FETCH (DIRECTREF 000102567276e431)
a1 OK FETCH completed
```

Similarly, the `STATUS` command can be used to return the direct reference of the folder. (At present, you need to enable `X-OMTEST` for this to work):

```
a2 status mine1 (messages directref)
* STATUS mine1 (MESSAGES 1 DIRECTREF 0001061704ccdb0f)
a2 OK STATUS completed
```

Viewing Configuration (Tweak) Settings

The `XOM.CONFIG` command is particularly useful in a test situation, as it will report the current settings of IMAP tweaks:

```
a6 xom.config
* CONFIG "IMAP_CAPABILITIES=IMAP4 IMAP4rev1 X-OPENMAIL AUTH=LOGIN
NAMESPACE IDLE"
* CONFIG "IMAP_AUTOMATIC_MDN=FALSE"
* CONFIG "IMAP_BB_FOLDER_PREFIX=Bulletin Boards"
* CONFIG "IMAP_BB_FOLDER_SEPARATOR=/"
* CONFIG "IMAP_CACHE_RFC822_HEADER=FALSE"
* CONFIG "IMAP_DELETE_SUBFOLDERS=FALSE"
* CONFIG "IMAP_CACHE_RFC822_MESSAGE=FALSE"
* CONFIG "IMAP_FOLDER_PREFIX="
* CONFIG "IMAP_FOLDER_SEPARATOR=/"
* CONFIG "IMAP_IDLE_TIMEOUT=30"
* CONFIG "IMAP_LOGFILE=~/tmp/imap.%h"
* CONFIG "IMAP_LOGLEVEL=0x8"
* CONFIG "IMAP_MDNSENT_FLAG=$MdnSent
* CONFIG "IMAP_MIN_SIZE_ESTIMATE=0
* CONFIG "IMAP_SEARCH_TIMEOUT=0
* CONFIG "IMAP_REMOTE_UAL_ENABLED=TRUE"
* CONFIG "IMAP_UAL_TRACE_LEVEL=8"
* CONFIG "IMAP_USE_ITEM_BROWSER_SELECTION=TRUE"
* CONFIG "IMAP_X_NETSCAPE_URL=http://kuda.pwd.hp.com/cgi-bin/ice.cgi/
prof=ICE
a6 OK XOM.CONFIG completed
```

The `X-OMTEST` extensions allow you to change configuration settings using the `XOM.CONFIG` command.

Note: IMAP_CACHE_RFC822_HEADER=FALSE and IMAP_CACHE_RFC822_MESSAGE=FALSE are now ignored. At B.05.10 they were used to turn off the creation of cached object files.

X-OMTEST

As with `X-OPENMAIL`, this capability denotes proprietary extensions for testing against the OpenMail server. This capability should not normally be enabled, as it allows a client to modify the message store and the configuration of the running IMAP server in ways which are dangerous.

Dynamically Modify Configuration Settings

In addition to reporting the current setting of various tweaks, the `XOM.CONFIG` command can also modify them. For example, to change the `IMAP_IDLE_TIMEOUT` tweak to 31, the command is:

```
XOM.CONFIG IDLE.TIMEOUT 31
```

The option name used in the command is the tweak name without the initial `IMAP_` prefix and the remaining underscores replaced with periods. The option name part of the command line is case insentive.

Each IMAP server process keeps a copy of the current configuration option settings in its memory. The `XOM.CONFIG` command makes changes to this copy. This means that most of the changes have immediate effect, without having to stop and restart the IMAP server process. It also means that the changes are only applicable to the current session.

Note: If logging is already enabled, you cannot use `XOM.CONFIG` to change the log filename. The logging level can be changed mid session and logging can be turned on (in which case, changes to the log filename will be effective).

This extension is particularly useful for testing folder prefix and separator settings and for changing capability settings in a test environment.

Some care has to be exercised when dynamically changing tweaks. Obviously, changing the set of known capabilities, disabling the `X-OMTEST` capability, for example, can render the `XOM.CONFIG` command inoperable. Some changes are slightly less obviously dangerous; for example, setting folder separator to, say, "e" will tend to make a lot of folders invisible.

Object File Access

The `FETCH` command takes an additional verb, `OM.OBJFILE`, that takes a list of object file names as its parameter, so, for example:

```
FETCH 1 OM.OBJFILE (imapBodyStrc)
```

will fetch details about the `imapBodyStrc` object file associated with message 1. Note that `OM.OBJFILE` is case insensitive, but the object file name is case sensitive:

```
a5 fetch 1 om.objfile (imapBodyStrc)
* 1 FETCH (OM.OBJFILE ((imapBodyStrc "16-Sep-1999 15:26:39 +0100" 0x3
{880}
OMHEADER (VERSION 1.2 FLAGS "Q 0 0 0 NIL Q 0 Q 0 NIL NIL F 0 0 1")
RFC822.HEADER {429}
Date: Thu, 16 Sep 1999 15:24:11 +0100
Message-Id: <H000006600000a85.0937491849.kuda.pwd.hp.com@MHS>
Subject: 2nd rtf test
MIME-Version: 1.0
Return-Path: <Admin_Mr/canberra_class@kuda.pwd.hp.com>
From: Admin_Mr/canberra_class@kuda.pwd.hp.com
To: Admin_Mr/canberra_class@kuda.pwd.hp.com
Content-Type: application/rtf; name="BDY.RTF"
Content-Disposition: attachment; filename="BDY.RTF"
Content-Transfer-Encoding: base64

ENV (DATE "Thu, 16 Sep 1999 15:24:11 +0100" SUBJECT "2nd rtf test" FROM
(Admin_Mr/canberra_class@kuda.pwd.hp.com) TO (Admin_Mr/
canberra_class@kuda.pwd.hp.com) MSGID "<H000006600000a85.0937491849.kuda.
pwd.hp.com@MHS>") BODY ((PATH 1 TYPE APPLICATION STYPE RTF TPARMS (NAME
BDY.RTF) ENC BASE64 SIZE 654 DISP ATTACHMENT DPARMS (FILENAME BDY.RTF)))
SIZE 1083 LINES 20)))
a5 OK FETCH completed
```

For a description of the contents of the imapBodyStrc object file, look back at the section “The imapBodyStrc Object File” on page 16.

Search Specification

You use the `XOM.SEARCHSPEC` command in the same way as the `SEARCH` command, for example:

```
a5 xom.searchspec subject golf
```

except that it returns the search specification transaction file (base64 encoded) instead of running the search.

To decode the output, you can use the `mmencode` utility for encoding/decoding MIME messages. You can obtain this, as part of Metamail, by ftp:

```
ftp://ftp.research.telcordia.com:/pub/nsb/mm2.7.tar.Z
```

To decode the output from the `XOM.SEARCHSPEC` command, type the following line:

```
mmencode -u > /tmp/xxx
```

The command will then sit, waiting for input. Cut the encoded text output by `XOM.SEARCHSPEC` and paste it as input to `mmencode`. Press Enter. The decoded output is sent to file `/tmp/xxx`. Then run `tfbrowse` on the file:

```
tfbrowse -i /tmp/xxx
```

The output is the search specification file.

Count Strings

The `XOM.COUNT.STRINGS` command was written to test the lexical phase in the server. It returns the MD5 checksum of each of its arguments. The purpose of this extension is to ensure that the server correctly retrieves atoms, quoted strings and literal strings from the client. The server computes the MD5 checksum of each of its argument strings. You can then compare these checksums against those you generate independently using the GNU `md5sum` utility.

It is fairly unlikely that you would want (or need) to use this extension.

Delete Object Files

The command `XOM.DELOBJ` will delete object files. For example,

```
XOM.DELOBJ 1:* (imapMsg imapHdr)
```

will delete all the old IMAP server cache files in the current folder. The command

```
XOM.DELOBJ 1:* ALL (imapMsg imapHdr)
```

will also delete any object files that are marked as permanent.

This command, together with `XOM.CLEARCACHE`, can be used in test situations to return to the point before a message is initially processed.

There are alternative ways of deleting object files. You could use the "`do`" option in `omcontain` to delete temporary or permanent object files. As described in “Deleting Temporary And Permanent Object Files” on page 23, `omscan` can be run wth the `-O` option to delete temporary, but not permanent, object files.

Flush Internal Cache

The `XOM.CLEARCACHE` command flushes the internal caches of the IMAP server. The cache comprises, at most, the message structure and a memory mapped file for the current message and computed message sizes for some or all the messages in the current folder.

The reason you might want to use this command is to clear any cached information in the server, so that tests can be sure that they are getting fresh information from the UAL server and/or the item browser.

Following illustrates Netscape Messenger type interaction, where the message body structure is requested first:

Client	**IMAP**	**UAL**	**BRW**
FETCH [3.2]	---------------BODYSTRUCTURE------------>		
			Renders msg, generates imapBodyStrc
	Parse to get part 3.2	<---Complete Msg----------	
<-----3.2--------------			
FETCH [3.2]			
<-----3.2-------------- IMAP gets this from cache			
XOM.CLEARCACHE			
FETCH [3.2]	---------------BODYSTRUCTURE--------------->		
	Parse object file to find path of 3.2	<--------------------------	Returns existing imapBodyStrc object file
	--------------------fetch "path [3.2]"-------------->		
			part 3.2 put in specified file
	<----------------------OK done that-------------------------		
<--part 3.2 file sent to client-------			

Note: In the first instance it is the IMAP server that returns the message part. In the second instance, the item browser returns it).

To get back to the "new message" state, you would have to delete the `imapBodyStrc` object file as well as clearing the cache.

`XOM.CLEARCACHE` will also clear out its knowledge of the size of messages. This could be useful if you are testing the effects of the `MIN_SIZE_ESTIMATE` tweak. Try the following exercise:

Initial setup: IMAP_MIN_SIZE_ESTIMATE=0, send a message greater than 10k bytes in size.

```
a1 FETCH 1:* RFC822.SIZE
```

Where are we getting the size from? (*Answer* - `imapBodyStrc` created by item browser).

```
a2 FETCH 1:* RFC822.SIZE
```

Where are we getting the size from this time? (*Answer* - IMAP server's in memory data structure of this msg).

Change the configuration setting as follows:

```
a3 XOM.CONFIG MIN.SIZE.ESTIMATE 10
```

What does this do? (*Answer* - changes threshold below which messages are rendered to get size. The estimated size of messages greater than 10kbytes is taken from the message container in the message store).

Enter the request:

```
a4 XOM.CLEARCACHE
```

What have we cleared? (*Answer* - IMAP server's in memory cache of this message).

```
a5 FETCH 1:* RFC822.SIZE
```

Where are we getting the size from this time? (*Answer*- `imapBodyStrc` created by item browser).

What would we have to do to get back to "new message" state? (*Answer* - clear the IMAP server cache *and* delete the `imapBodyStrc` object file).

Common Problems

Missing Folders And Bulletin Board Access

Folders that contain the folder separator (usually /) generally go missing. They are easily brought back by changing the folder separator (using the tweaks `IMAP_FOLDER_SEPARATOR` for user folders or `IMAP_BB_FOLDER_SEPARATOR` for Bulletin Board folders). This can be done on a per-client machine or per-user basis.

Clients that use `LIST ""%` are likely to be able to see the Bulletin Boards. Clients that use `LIST "" *` are quite likely not to be able to see the Bulletin Boards.

For `LIST "" %` to work in this instance the folder prefix and and Bulletin Board prefix should be defaulted.

Configuring Netscape To See Folders And Bulletin Boards

Netscape Messenger 4.5/6 gets confused with \ (back slash) as a folder separator. Messenger also has problems with `#bb` (the default) as the Bulletin Board prefix. In fact, any folder that contains a # is ignored by Netscape.

Use the tweaks, `IMAP_FOLDER_SEPARATOR` and/or `IMAP_BB_FOLDER_SEPARATOR` , to change the user and Bulletin Board folder separators to something else, making sure that you do not use a character which appears in folder names.

Use the tweak, `IMAP_BB_FOLDER_PREFIX`, to change the Bulletin Board prefix, for example:

```
IMAP_BB_FOLDER_PREFIX=bulletin-boards
```

Configuring Outlook Express To See Bulletin Boards

Outlook Express 5’s implementation of the `NAMESPACE` extension is unreliable, leading to problems seeing Bulletin Boards. I can get Outlook Express to view the Bulletin Boards, but it needs reconfiguring each time I want to update the list of Bulletin Boards. Here is what I did:

- Server setting in `general.cfg`:

  ```
  IMAP_BB_FOLDER_PREFIX=Bulletin Boards
  ```
- In Outlook Express, add an account (`Tools`->`Accounts`) for access to the Bulletin Boards.
- In Outlook Express (`Tools`->`Accounts`->`Properties`->`IMAP`) set root folder path to be `/Bulletin-Boards`
- Untick "`Check for new messages`" and "`Store special folders`" boxes.
- Now right click on the account name, select `IMAP folders`, and `Reset list`. This will give an error.
- Now, again in `Tools`->`Accounts`->`Properties`->IMAP, set root folder path to be `Bulletin Boards` (remove the leading '/').
- Once again, right click on the account name, select `IMAP folders`, and `Reset list`. This time the Bulletin Board names should be downloaded.

This does not seem to be a very workable solution, but it shows that the client is behaving in an inconsistent way, and that it can see the Bulletin Boards under some circumstances.

Error Message: "NO some messages no longer exist"

This can tend to be a "catch all" error. It normally indicates concurrent access to a user’s folders, that is, one client has deleted a message that another client would like to be able to see (see “Multiple Access Support” on page 24). Most clients handle this reasonably well.

Unfortunately, the same error message tends to crop up when the item browser fails in some way.

Deleting Folders Containing Subfolders

Although the IMAP protocol forbids deleting folders that contain other folders, some clients will attempt to delete a folder without first deleting any folders that it contains. If the tweak, `IMAP_DELETE_SUBFOLDERS`, is set to `TRUE`, then the IMAP server will allow folders to be deleted, regardless of whether they contain any subfolders. For these non-conforming clients, this can improve usability.

Note: This tweak does not affect conforming clients at all.

The tweak can be set for the system, a client or a user and the default setting is:

```
IMAP_DELETE_SUBFOLDERS=FALSE
```

Lack Of Understanding Of Standards

Many clients do not have a correct understanding of MIME and/or IMAP. The OpenMail IMAP server can cope with most excesses, but the incorrect implementation of the following parts of the standards by certain clients is difficult to deal with:

Standards Relating To ...	**Clients At Fault**
Forwarded messages	Eudora Pro
multipart/alternative	Eudora Pro and Netscape, to some extent
Content-type	Outlook ‘98 and Outlook Express
Content-disposition	Outlook‘98, Outlook Express and Netscape, to some extent

Unreliable And Fragile Clients

Some clients seem to have just been inadequately tested. Mulberry, for example, is very unreliable in the face of concurrent access: if some other client is reading messages or deleting them, Mulberry gets a notification that upsets it badly.

Limiting The Number Of New Processes Netscape Can Start In A Session

Normally, for each new IMAP session there will be a minimum of three processes - a UAL process (`ual.remote`), an IMAP server process (`in.imap41d`) and item browser process (`item.browse`). During the session, the client usually makes additional, temporary connections to perform operations, such as authentication. Each of these temporary connections will require their own instances of the server processes, `ual.remote`, `in.imap41d` and `item.browse`.

During an IMAP session, Netscape 4.5 (and later) will keep up to seven connections open at any one time - a total of 21 processes! It seems that one of these is for the inbox, one for checking new mail and five for looking at different folders. The number of connections builds up gradually, you get one when you first sign on, one when the first check for new mail kicks in, and then the remaining five fill up as you start visiting folders. A server with a large number of IMAP users may quickly run out of process resources! If you want a more in depth discussion of this problem, see the OpenMail web document, web ID 100-1396 "Problem with Netscape and Multiple Connections and the solution".

Fortunately, there is an option in Netscape to limit the number of connections the client can make.

It is possible to set a parameter called "`mail.imap.max_cached_connections`" to 1 (the default is 5) in a preferences file, `prefs.js`, which you will find (on a PC) in Program Files -> Netscape -> Users -> default. It is advisable to take a copy of this file before editing it. This parameter can also be set on a site-wide basis at installation time. It is documented as being the maximum number of non-inbox connections that will be held open at any one time. See:

`http://developer.netscape.com/docs/manuals/deploymt/jsprefs.htm`

With this set, you will probably notice that the maximum number of connections stays around one: occasionally a new connection will be fired up and so you'll have two connections for a few seconds, but one of the connections gets shut down fairly soon. With new mail notification turned off, it seems that you just have the one connection all the time.

In addition, Netscape actually seems to be rather faster. The reason for this is may be that instead of creating new sessions, which takes a while, the server just re-uses the one connection that it already has, which doesn't take long.

If you were going back and forth between a small number of largish folders, then you would get the performance benefit of having more than one connection available, because the UAL server wouldn't have to keep listing folders.

Watcher TCL Script

```
#!/bin/sh
# -*- tcl -*- \
##
#This TCL script allows you to monitor client/server connections.
#You need to have the TCL interpreter library included in your PATH.
#IMPORTANT NOTE: This script is provided as is and is NOT supported by HP
##
   exec wish "$0" "$@"

if {$tcl_platform(platform) == "unix"} {
    catch {
   tk_setPalette \
      foreground [option get . foreground Foreground] \
      background [option get . background Background]
    }
    option add *borderWidth 1
}


proc watchConversation {serverHost serverPort clientFD addr port} {
    for {set i 0} {[winfo exists .watch$i]} {incr i} {}

    set w [toplevel .watch$i]
    menu $w.menu
    menu $w.menu.file -tearoff false
    menu $w.menu.help -tearoff false
    $w.menu add cascade -label File -underline 0 -menu $w.menu.file
    $w.menu add cascade -label Help -underline 0 -menu $w.menu.help
    $w conf -menu $w.menu

    $w.menu.file add command -label "Save transcript..." -underline 5 \
       -command [list saveTranscript $w]
    $w.menu.file add command -label "Save client..." -underline 5 \
       -command [list saveTranscript $w C]
    $w.menu.file add command -label "Save server..." -underline 5 \
       -command [list saveTranscript $w S]
    $w.menu.help add command -label About -underline 0 \
       -command [list aboutConnection $w]

    text $w.text \
       -wrap none \
       -font textFont \
       -insertbackground red \
       -yscrollcommand "$w.yscroll set" \
       -xscrollcommand "$w.xscroll set"
    scrollbar $w.yscroll \
       -orient vertical \
       -command "$w.text yview"
    scrollbar $w.xscroll \
       -orient horizontal \
       -command "$w.text xview"

    global tcl_platform
    if {$tcl_platform(platform) == "unix"} {
   $w.text conf \
      -foreground Black \
      -background WhiteSmoke \
      -cursor "xterm red"
   $w.yscroll configure -width 10
   $w.xscroll configure -width 10
```

```
        }
        $w.text tag configure "S" -font serverFont
        $w.text tag configure "C" -font clientFont

        grid $w.text    -row 0 -column 0 -sticky nswe
        grid $w.yscroll -row 0 -column 1 -sticky nsw
        grid $w.xscroll -row 1 -column 0 -sticky swe
        grid columnconfigure $w 0 -weight 1
        grid rowconfigure $w 0 -weight 1
        focus $w.text

        set serverFD [socket $serverHost $serverPort]
        fconfigure $clientFD -buffering line -translation binary
        fconfigure $serverFD -buffering line -translation binary
        fileevent $clientFD readable "clientToServer $w"
        fileevent $serverFD readable "serverToClient $w"

        global $w.clientFD
        global $w.serverFD
        set $w.clientFD $clientFD
        set $w.serverFD $serverFD
        bind $w <Destroy> {
       if {[string compare "%W" "[winfo toplevel %W]"] == 0} {
           catch {close ${%W.serverFD}}
           catch {close ${%W.clientFD}}
           catch {fileevent ${%W.serverFD} readable {}}
           catch {fileevent ${%W.clientFD} readable {}}
           catch {unset %W.serverFD}
           catch {unset %W.clientFD}
       }
        }
    }

    proc clientToServer {w} {
        upvar #0 $w.clientFD clientFD
        upvar #0 $w.serverFD serverFD

        if {[gets $clientFD line] < 0} {
       if [catch {close -write $serverFD}] {
           close $serverFD
       }
       if [catch {close -read $clientFD}] {
           close $clientFD
       }
       appendText $w.text "C" "*** EOF ***"
        } elseif [catch {puts $serverFD $line} msg] {
       appendText $w.text "C" [string trimright $line "\r\n"]
       if [catch {close -write $clientFD}] {
           close $clientFD
       }
       if [catch {close -read $serverFD}] {
           close $serverFD
       }
       appendText $w.text "S" "ERROR: $msg"
        } else {
       appendText $w.text "C" [string trimright $line "\r\n"]
        }
    }

    proc serverToClient {w} {
        upvar #0 $w.clientFD clientFD
        upvar #0 $w.serverFD serverFD
        if {[gets $serverFD line] < 0} {
       catch {close $serverFD}
       unset serverFD
       after idle "catch {close \${$w.clientFD}}; catch {unset $w.clientFD}"
       appendText $w.text "S" "*** EOF ***"
        } elseif [catch {puts $clientFD $line} msg] {
       appendText $w.text "S" [string trimright $line "\r\n"]
       catch {close $clientFD}
       catch {unset clientFD}
       appendText $w.text "C" "ERROR: $msg"
        } else {
       appendText $w.text "S" [string trimright $line "\r\n"]
        }
    }

    proc timestamp {} {
        global tcl_platform
        # We should probably calibrate this on a per-platform basis, but I
        # "know" that UNIX clicks are microseconds and Windows95 clicks are
        # milliseconds.
        switch $tcl_platform(platform) {
       unix {
           set centis [format "%02d" [expr ([clock clicks] % 1000000)/10000]]
       }
       windows {
           set centis [format "%02d" [expr ([clock clicks] % 1000)/10]]
       }
        }
        return [clock format [clock seconds] -format "%H:%M:%S.$centis"]
    }

    proc appendText {w tag msg} {
        $w insert end \
         "[timestamp]" timestamp \
         ":$tag: " "" \
         $msg $tag \
         "\n" ""
        global $w-update
        if {![info exists $w-update] || [lsearch [after info] [set $w-update]]
    < 0} {
       set $w-update [after 500 "catch {$w see insert}; update idletasks"]
        }
    }


    proc makeWatcher {{from {}} {to {}}} {
        for {set i 0} {[winfo exists .f$i]} {incr i} {}
        set f [frame .f$i]

        upvar #0 $f watcher
        checkbutton $f.active -command [list runStopWatcher $f] \
           -text "" \
           -onvalue 1 -offvalue 0 \
           -variable ${f}(start) \
           -takefocus 0
        set watcher(start) 0
        entry $f.from -textvariable ${f}(fromHostPort)
        set ::${f}(fromHostPort) $from
        entry $f.to -textvariable ${f}(toHostPort)
        set ::${f}(toHostPort) $to

        bind $f.from <Return> [list $f.active invoke]
        bind $f.to <Return> [list $f.active invoke]
        grid $f.active -row 0 -column 0 -sticky we
        grid $f.from -row 0 -column 1 -sticky we
        grid $f.to -row 0 -column 2 -sticky we
        grid columnconfigure $f 1 -weight 1
        grid columnconfigure $f 2 -weight 1
        pack $f -side bottom -expand 1 -fill both

        focus $f.from
        focus .
    }

    proc runStopWatcher {f} {
        upvar #0 $f watcher

        if {$watcher(start)} {
       set fromHost ""
       set fromPort ""
       set toHost ""
       set toPort ""
       foreach {fromHost fromPort} [split $watcher(fromHostPort) ":"] break
       foreach {toHost toPort} [split $watcher(toHostPort) ":"] break

       if [string match "" $fromPort] {
           set fromPort $toPort
       } elseif [string match "" $toPort] {
           set toPort $fromPort
       }
       if {[string match "" $fromPort] || [string match "" $toPort]} {
           tk_messageBox -icon error -type ok \
              -message "At least one port must be specified"
           set watcher(start) 0
           return
       }

       if [string match "" $toHost] {
           if [info exists env(OMCURRENT)] {
         set toHost $env(OMCURRENT)
           } else {
         tk_messageBox -icon error -type ok \
            -message "No target host"
         set watcher(start) 0
         return
           }
       }

       if [catch {
           if [string match "" $fromHost] {
         set watcher(sock) [socket -server [list watchConversation $toHost
    $toPort] $fromPort]
           } else {
         set watcher(sock) [socket -myaddr $fromHost -server [list
    watchConversation $toHost $toPort] $fromPort]
           }
       } msg] {
           tk_messageBox -icon error -type ok -message $msg
           set watcher(start) 0
       }
        } else {
       close $watcher(sock)
        }
    }
```

```
proc saveTranscript {w {tag ""}} {
    if ![regexp {\.([^.]+)} $w {} file] {
   set file transcript
    }
    set file [tk_getSaveFile \
       -defaultextension .scr \
       -initialfile $file \
       -parent . \
       -title "Save Transcript" \
       -filetypes {
     {"Transcript Files" {.scr}}
     {"All Documents" {.*}}
       }]

    if ![string match "" $file] {
   if [catch {
       set fd [open $file w]
       if [string match "" $tag] {
     puts $fd [$w.text get 1.0 end-1c]
       } else {
     foreach {first last} [$w.text tag ranges $tag] {
         append text [$w.text get $first $last] "\n"
     }
     puts -nonewline $fd $text
       }
       close $fd
   } msg] {
       tk_messageBox -icon error -type ok \
         -message $msg
   }
    }
}

proc aboutConnection {w} {
    upvar #0 $w.clientFD clientFD
    upvar #0 $w.serverFD serverFD

    if {[info exists clientFD] && ![catch {fconfigure $clientFD} conf]} {
   append msg "Client Connection:\n"
   foreach {tag value} $conf {
       switch -- $tag {
     -peername {
         append msg "    peername: [join [lrange $value 1 end] :]\n"
     }
     -sockname {
         append msg "    sockname: [join [lrange $value 1 end] :]\n"
     }
       }
   }
   append msg "\n"
    } else {
   append msg "No client connection\n\n"
    }

    if {[info exists serverFD] && ![catch {fconfigure $serverFD} conf]} {
   append msg "Server connection:\n"
   foreach {tag value} $conf {
       switch -- $tag {
     -peername {
         append msg "    peername: [join [lrange $value 1 end] :]\n"
     }
     -sockname {
         append msg "    sockname: [join [lrange $value 1 end] :]\n"
```

```
              }
                }
            }
              } else {
            append msg "No server connection"
              }

              tk_messageBox -icon info -type ok -message [string trim $msg]
        }


        if {$tcl_platform(platform) == "unix"} {
            # Set a 75dpi screen regardless ...
            tk scaling [expr 75.0/72.0]
            set family "Courier"
            set size 12
            event add <<Cancel>> <Control-c>
        } else {
            bind all <F12> {console show}
            bind all <Shift-F12> {console hide}
            event add <<Cancel>> <Escape>
            set family "MS Sans Serif"
            set size 8
        }
        bind all <<Cancel>> {after idle [list destroy [winfo toplevel %W]]}

        foreach font {textFont clientFont serverFont} {
            font create $font -family $family -size $size
        }
        font configure clientFont -weight bold

        if {$argc < 2} {
            makeWatcher
        } else {
            foreach {from to} $argv {
          makeWatcher $from $to
            }
        }
```