OTN OpenMail Technical Note

Queues and Parallel Processing (B.06.00)

Summary

This OTN explains the new queue system and parallel processing functionality introduced in OpenMail version 6.0.

The following topics are covered:

- How the new Queue Manager process manages the message queues in memory and communicates with queue reader processes.
- The Message Pool, where details of all the in transit messages are held on disk.
- The `POISON` and `IDEL` queues and the new message deletion system.
- How to configure and monitor multiple readers for a queue.
- New `omqdump` options to help you troubleshoot queue message problems.

The contents of the OTN will be included in the OpenMail Upgrade class, OM360. The class will be run for a limited period from October, 1999, after which the contents of this document will be merged with existing AE351 course material and this OTN will be withdrawn.

Readership

As this is an update OTN, I have assumed readers are familiar with OpenMail and, in particular, the way the message queues worked in previous releases. I have also assumed familiarity with `omqdump`.

Comments Please!

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

Revision History

18th May 1999	This is the first issue of this OTN.
11th November 1999	Clarification about: Turning off readers before manipulating messages with omqdump Not deleting old queues directory When auxiliary readers are given messages to process

Contents

Introduction **3**
An Overview Of the Changes 3
Message Life Cycle 5
Message pool **6**
How Message Details Are Held In Pool Files 6
How Are The Pool Files Created? 8
Upgrading Pre B.06.00 Queues 8
How Are Pool Files Updated? 9
How Can I View The Pool Files? 10
The Pool Control File 10
The Queue Manager **11**
What happens if the Queue Manager dies? 13
Starting Up Queue Manager If The Message Pool Is Corrupt 14
The POISON Queue 14
The Idel Queue And The Item Delete Daemon 15
Queue Reader Processes **17**
Multiple Readers For A Queue 17
How To Configure Multiple Reader Processes 18
How To Monitor Multiple Reader Processes 18
Choosing The Optimum Number Of Reader Processes 20
Logging Reader Processes 21
Troubleshooting Tools **23**
The Queue Manager’s Tombstone File 23
omfreeq 23
omqdump 23

Introduction

Before B.06.00 the main area, which needed improving if OpenMail was to scale effectively to very large, fast systems, was that of bottlenecks occurring on the `ROUTER` and `LOCAL` queues because the reader processes, Service Router and Local Delivery, were busy performing i/o. Because all the queued messages were held on disk, the whole message processing system was, to some extent, i/o bound.

To address this performance issue, the queues have been moved from disk to the memory of a new central daemon, called the Queue Manager. The responsibilities of this daemon are:

- To organize the queues of messages.
- To act as a scheduler for the queue reader processes.
- To monitor queue reader processes and keep track of all the in transit messages in the system.

Only the Queue Manager accesses the queues - not the service processes, as before - and it is almost totally CPU bound, making it very fast indeed. The Queue Manager communicates with the queue reader processes, such as Service Router, using Unix IPC message queues.

The introduction of the Queue Manager has brought several important, additional benefits:

- Communication between service processes has been simplified.
- A single process now keeps track of all the messages moving through the system.
- The Queue Manager is well placed to provide statistics on message throughput and the performance of reader processes.

So we have speeded up the queue access, but surely the bottleneck will just move to the service processes? This is where the B.06.00 *Parallel Processing* comes in. In previous releases, each queue could have only one reader. Now you can configure up to 21 reader processes for the main queues. Tests so far have shown significant improvements to message throughput.

Deleting messages also impacted the performance of the OpenMail processes, so a new message deletion system has been introduced, reducing the workload of the service processes. A new queue, called the `IDEL` queue, and an Item Delete daemon perform this function.

In this OTN we will look at how the new queue system works, how to configure and monitor multiple reader processes and how you can use new options in `omstat` and `omqdump` to monitor performance and track messages as they move through OpenMail.

An Overview Of the Changes

The **Introduction** gave the rationale for the changes introduced in B.06.00. We will now look, in a little more detail, at the new system as a whole, before studying the individual parts.

Remember the files: `~openmail/queues/qcontrol`, `<q-name>.qua` and `<q-name>.qub`? Well, they are all obsolete! At B.06.00 the new queue system comprises:

- Queue structures held in memory. The queues are no longer held on disk. Instead, the new Queue Manager daemon creates and maintains queue structures in its memory.
- A Message Pool directory, `~openmail/msgpool`. This directory contains up to 100 pool files, `0QP` - `99QP`, and a pool control file, `~openmail/msgpool/poolctrl`. Although the queues are not held on disk, a record for each in transit message in the system is held and updated in the pool files. The pool control file keeps a list of the configured queues and the number of pool files created.
- A Queue Manager daemon, `/opt/openmail/bin/queue.manager`. This `NON-STOP` daemon accesses and manages the queues, schedules queue reader processes and keeps track of all the in transit messages in the system. The Queue Manager also manages two

new, special queues, the `IDEL` queue and the `POISON` queue. The `IDEL` queue is part of the new message deletion system and the `POISON` queue is for messages which repeatedly kill reader processes.

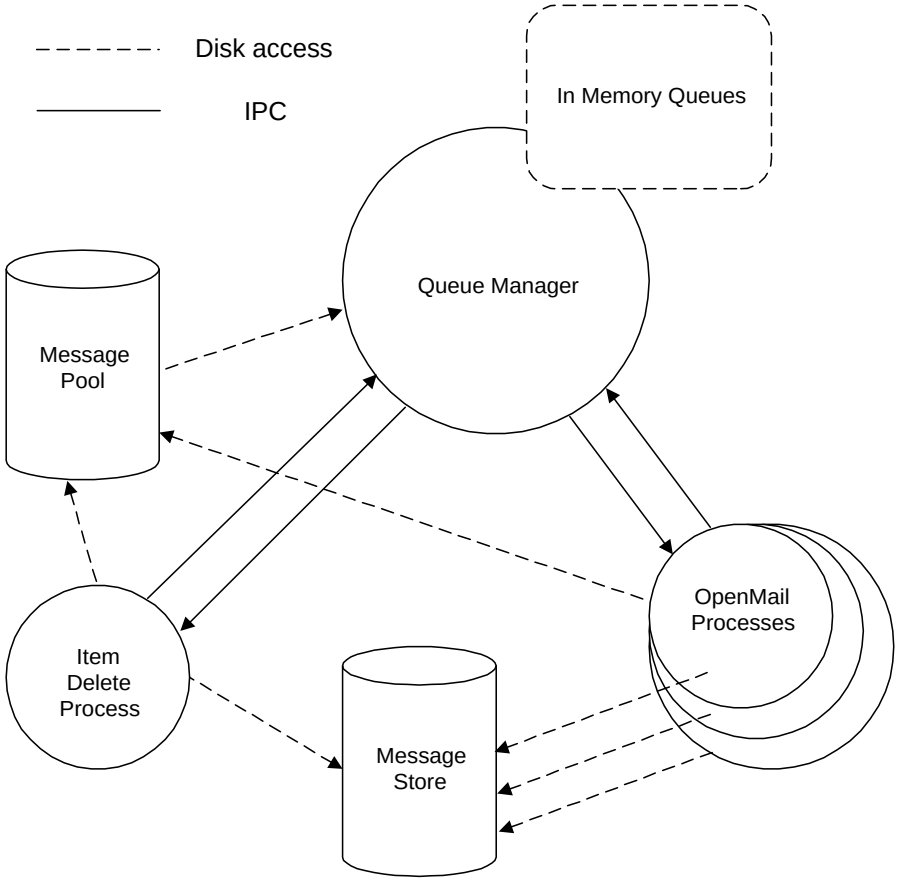
- An Item Delete daemon, `/opt/openmail/bin/idel.server`. A new mechanism for deleting messages has been introduced. At start up, this daemon is started by the Queue Manager. Its function is to delete messages on the `IDEL` queue. In previous releases, the reader processes were responsible for deleting messages.

To make full use of the speed of the Queue Manager, queue reading has also been enhanced so that you can run concurrently multiple instances of the following queue reader processes:

- Local Delivery
- Sevice Router
- Sendmail interface (xport.out)
- Internet Gateway (unix.out)

The diagram below shows the main parts of the new queue system and how they communicate.

Figure 1: Overview of the New Queue System



Notice the queues are only accessed by the Queue Manager, not by the OpenMail processes. The processes, for example, Service Router and Local Delivery, send requests for messages and receive responses from the Queue Manager via IPC message queues. The Queue Manager also communicates with the Item Delete daemon using IPC messages.

Accessing messages and message details on disk, in the Message Store and the Message Pool, is performed mainly by the OpenMail processes and the Item Delete daemon. The Queue Manager only accesses the Message Pool at start up and if it ever puts a message on the `POISON` queue.

Probably the best way to summarise how all these parts work together to route messages within the OpenMail system, is to describe the process in terms of the lifecycle of a message.

Message Life Cycle

For the purposes of this discussion, we will assume that in the life cycle of a message there are three main stages: *birth*, *life* and *death*.

1. Birth takes place when a new message enters the system. Suppose we have created a message using our mail client and pressed the "Send" button to submit it to the OpenMail system. The UAL process then creates the cut down message container, Transaction File, DL and message content parts in the Message Store, (`~openmail/data`), as described in the Message Store OTN.

   The process (UAL process in this case) opens an existing pool file, *nn*`QP`, or creates a new pool file in the Message Pool. It then writes a pool file record, which includes among other details, the `ITEM_REF` of the cut down message container and the name of the first queue(s) the message is to be put on. In our example, this would be the `ROUTER` queue.

   The process sends an IPC request to the Queue Manager to put the message on the queue and includes the message details in the request.

2. During the *life* of a message, the Queue Manager passes details of the message in IPC requests/responses to the reader(s) of the queue(s) to which the message is attached. A reader is a process, e.g. Service Router, which processes messages from a queue. In B.06.00, there may be several Service Routers processing messages from the `ROUTER` queue concurrently.

   As the message moves between queues, the process currently handling the message updates the queue names in the message's pool file record and informs the Queue Manager of the change.

3. The *death* of a message is when the message leaves the OpenMail system completely or is delivered to a user's mailbox. The last process to handle the message removes the last queue name from the message's pool file record and informs the Queue Manager. If our message is destined for another user on this system, the process handling the message at this stage would be Local Delivery and the last queue name to be removed from the pool file record would be `LOCAL`.

   The Queue Manager moves the message to the `IDEL` queue.

   The Item Delete daemon reads the message details from the `IDEL` queue and deletes the pool file record and the cut down message components in the Message Store.

Now that you have a general understanding of how the system works, we shall take a closer look at the individual parts.

Message pool

Although all the queued messages are now held in memory, the details of all these messages need to be secured to disk somewhere, in case of a system crash. This is the function of the new directory `~openmail/msgpool`.

An `ls` of this directory on my system shows a number of files with the name *nn*`QP`, which we will refer to as pool files, and a `poolctrl` file. There can be up to 100 pool files (`0QP` to `99QP`). If you looked at this directory on a newly installed B.06.00 system, you would find only the `poolctrl` file, as the pool files are created as needed by the service processes.

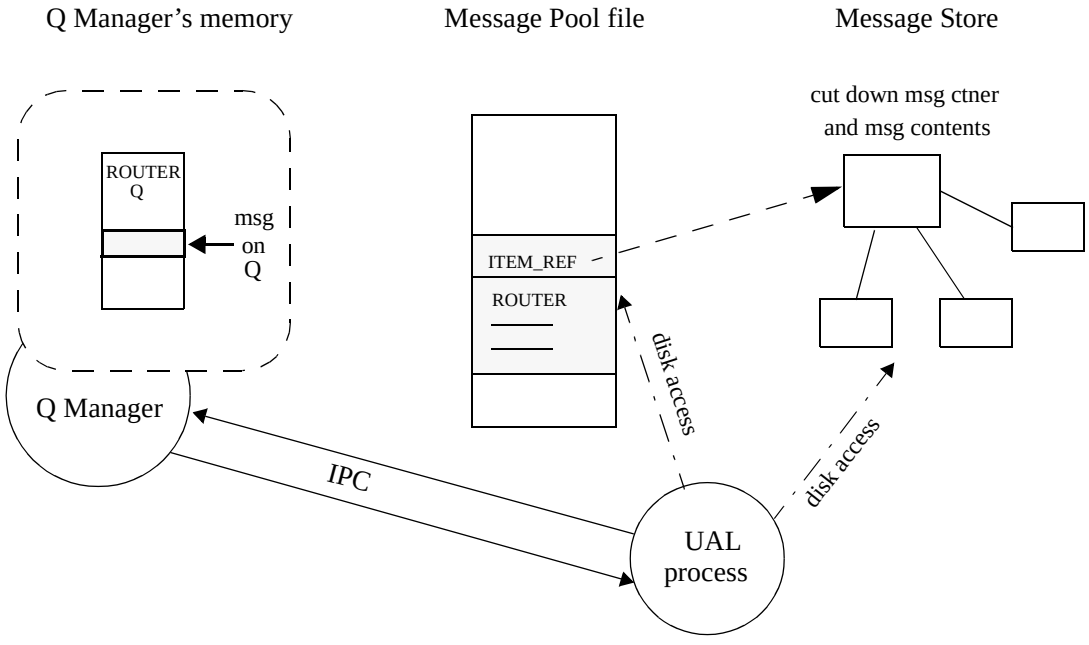
```
root@kuda[] #ls ~openmail/msgpool
0QP       17QP      24QP      31QP      39QP      46QP      53QP      60QP
10QP      18QP      25QP      32QP      3QP       47QP      54QP      6QP
11QP      19QP      26QP      33QP      40QP      48QP      55QP      7QP
12QP      1QP       27QP      34QP      41QP      49QP      56QP      8QP
13QP      20QP      28QP      35QP      42QP      4QP       57QP      9QP
14QP      21QP      29QP      36QP      43QP      50QP      58QP poolctrl
15QP      22QP      2QP       37QP      44QP      51QP      59QP
16QP      23QP      30QP      38QP      45QP      52QP      5QP
```

How Message Details Are Held In Pool Files

Each message in the system has only one record in the Message Pool. This record is held in one of the pool files and contains details, such as the `ITEM_REF` of the cut down message container and a list of the queues, which that message is on. In previous releases, there were several records in the queue files, one for each queue the message was on. Remember that copies of messages, for example, BCCs, are separate messages. Each BCC of a message will therefore have its own pool file record.

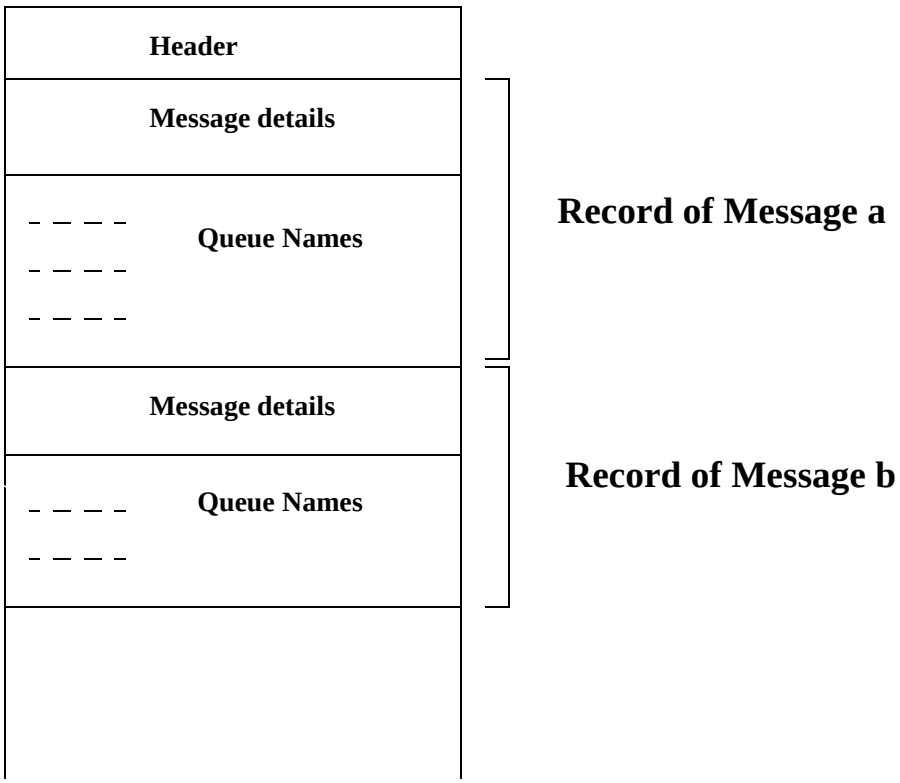
The diagram below shows the message details, created in a Message Pool file by the UAL process. This record includes a list of the queues the message is on; in this example, just the `ROUTER` queue. The record references the cut down message container, which has been created in the Message Store. The UAL process then passes the message details to the Queue Manager via IPC, requesting the message be added to the `ROUTER` queue in the Queue Manager's memory.

Figure 2: Overview Of How Message Details Are Held And Used



The diagram below outlines the format of the pool files.

Figure 3: Pool File Format



Each record has two parts, a *message part* and a *queue part*. The message part holds:

- `ITEM_REF` of the message's cut down message container.
- The time the record was created in the pool file.
- A sequential index ID number, which is unique within the Message Pool. This ID is called the `QPoolId` of the message.

The queue part holds an entry for each queue the message is on. If the message is on several queues, there will be several queue name parts for the message, but only one message part. Each queue part contains:

- The queue name.
- The message priority setting.
- The time the message was put on the queue.
- A defer time. This type of deferral is specific to Queue Manager and not the same as user deferred or administrator deferred messages. There are two instances when this might be set:

  a. When the Unix Gateway process cannot send the message to sendmail, for some reason, the message will be deferred and sent later.

  b. When a process dies while processing a message, the Queue Manager delays the message before giving it to the reader again.

How Are The Pool Files Created?

When a message is submitted to the system, the appropriate service process creates the cut down message container and message component files in the Message Store. After this, the process will select a pool file and write the message details in a free record in the file, including the name of the first queue that the message is to be attached to. This pool file is then sync'd to disk and the message is now secure should the system crash.

The process selects which pool file to open by calculating

```
<PID of the process> modulus 100
```

This gives a number in the range 0 - 99, which the process then uses as part of the pool file name, e.g. `12QP`. If this file does not yet exist, the process creates the next highest unused pool file and uses that. For example, If the files `0QP` to `15QP` are present and the modulus gives a higher number, `16QP` will be created and used.

The first time a pool file is opened the process reads the file sequentially looking for a slot in which to store a new message record. As messages are added, the process progresses though the file. Once the end of file is reached, the process will either append more messages to the file or restart looking from the beginning of the file, by which time some messages may have been processed, and spaces created.

Why so many pool files? The number was chosen to reduce locking conflicts and spread disk traffic for striped/raid disks. On a heavily used system, disk i/o on the Message Pool will be high, so make sure that you have this part of your system optimized for i/o. The document, **Configuration Planning Guide** (ID 200-0200), on the OpenMail website, gives useful information on how to configure your system for OpenMail.

Upgrading Pre B.06.00 Queues

When you install B.06.00, a command called `omqconv` is called automatically to upgrade any messages in the `~openmail/queues` directory to the Message Pool. This routine reads each queue file in the `queues` directory. Whenever it finds a message, it extracts the information needed for the new Message Pool and creates a pool file, if required, and record for that message. After the upgrade has finished, the old `queue` files will remain. Some flag files, `<queue name>.R6`, are written to the directory to indicate that the queues have been upgraded. On subsequent B.06.00 installations, `omqconv` sees these files and reports that the queues have already been upgraded.

Note: As the information written into the old `queues` directory may be required should you need to reinstall OpenMail B.06.00, we suggest you don't delete this directory. It shouldn't be big anyway, because, unless you had real problems with your system, there would not have been many messages on queues at the time the system was upgraded.

The upgrading of each queue file and each message is clearly logged in the installation log file, `/tmp/ominstall.log`. If `omqconv` comes across a problem during the upgrade, for example, a message which cannot be upgraded for some reason, it writes lots of helpful information to the log file, leaves the message in the `queues` directory and carries on. So you need to check `/tmp/ominstall.log` carefully after installation, to find out if any messages have not been upgraded. In the unlikely event of a message failing the upgrade, it is most probably a message on the `ERROR` queue.

You can use `omqdump` to investigate any failed messages. In the `omqdump` section, we will look at a new option, `O`, which allows you to open a cut down message container file in `omqdump`. You can then move the message to the appropriate queue to perform the upgrade.

Warning! Make sure you manually upgrade any problem messages before running `omscan`. If you have messages left in the old `queues` directory after upgrading, then you run `omscan` before dealing with these messages, they will be treated as orphans by `omscan`.

Here's an example of some of the logging you will see in `/tmp/ominstall.log`:

```
executing /opt/openmail/bin/omqconv
Starting to upgrade messages on queues to new format.
Starting to upgrade queue "SMINTFC"
Finished Queue.
Starting to upgrade queue "SMERR"
Upgrading message "~/data/00000k2/04qb6hh"
Message upgraded correctly
Finished Queue.
Starting to upgrade queue "ERROR"
Upgrading message "~/data/00000k1/04qb6g9"
Message upgraded correctly
Upgrading message "~/data/00000k1/04qb6gb"
Message upgraded correctly
....
```

Just to outline what `omqconv` is doing, it:

- Reads the details of each message from the `~openmail/queues` files.
- Creates a new message pool file, `~openmail/msgpool/`*nn*`QP`, as required.
- Adds a record for each upgraded message to the appropriate pool file.

When the Queue Manager starts up, it reads the information from the pool files and constructs the queues in its memory, as normal.

How Are Pool Files Updated?

Service processes are responsible for sync'ing message details to disk when something in the pool file record is changed, for example, a message is put on a different queue. Only after the details are written to disk, does the process contact the Queue Manager to ask for the message to be put on the queue.

As a message is processed and the queue information changes, the queue part of the message record is overwritten. If this part of the record is empty, the message is not attached to any queue and is therefore ready to be deleted, that is, it is 'on' the `IDEL` queue. Note, the `IDEL` queue is never written to the pool file records.

Once a pool file is opened by a process this file will be kept open by the process until the process exits. For this reason, when a UAL process opens a pool file, all messages processed in a session will be put in the same pool file. You can see this if you turn off the Service Router, then send about 50 messages to yourself using a simple `omsend` script. If you then list (`ls -l`) the files in `~openmail/msgpool`, you will see one file is bigger than the rest. This is because the UAL process used the same pool file throughout the session.

Normally the pool files will not grow very large, because messages are constantly being processed and slots in the files freed for reuse.

Any service which creates new messages as part of processing a message from a queue, will have two pool files open; the pool file of the queued message it is processing and the pool file it opens to create new message records in. For example, the Service Router would have two pool files open when processing a message with BCCs. The service would need to update the original message's pool record and select another pool file (using its PID modulus 100), in which to create a new pool file record for each new BCC message it needs to create. Remember, during the existence of this Service Router process, it will always select the same pool file whenever it needs to create new pool file records.

This reduces the need for a large number of open files; the only files open by a process at any point in time should be the 'add' pool file and the 'last modified' pool file.

Whenever you restart OpenMail, pool files with unused space at the end are truncated to the last used record.

How Can I View The Pool Files?

The pool files are not text files, so you cannot view them using `cat`. However, the new `t` option in `omqdump` gives you a text dump of the contents of all the Message Pool files. We shall look at the output of this in the `omqdump` section.

The Pool Control File

The pool control file, `~openmail/msgpool/poolctrl`, keeps a list of the queues configured on the system and details of the number of pool files present. Again, you cannot view this file directly, but the new `L` option in `omqdump` reads this file and lists the queues.

The Queue Manager

Now let's find out more about the Queue Manager.

The actual process name is `/opt/openmail/bin/queue.manager`. As with other daemons, you can see its status by typing `omstat -a`:

```
root@kuda[] #omstat -a
PC Monitor                  Started             NON-STOP       0
Notification Server         Started             04.29.99       0
Shared memory daemon        Started             NON-STOP
Notification Monitor        Started             NON-STOP
Container Access Monitor    Started             NON-STOP
Item Structure Server       Started             04.29.99
Database Monitor            Started             04.29.99
Licence Monitor Daemon      Started             NON-STOP
LDAP Daemon                 Started             04.29.99
Queue Manager               Started             NON-STOP
Item Delete Daemon          Started             NON-STOP
IMAP Server Daemon          Started             04.29.99
```

It is a `NON-STOP` daemon, which means that it will start when OpenMail is started using `omrc`.

Note: If the Queue Manager dies for some reason, you cannot use `omon` to restart it. You must always use `omshut` then `omrc`, otherwise the queues would be in an inconsistent state.

At start up the daemon reads in the details of all the queued messages from the pool files in the Message Pool and builds the queues in its memory. The following information is held in the Queue Manager's memory:

- An active message list, which lists all messages that are currently being processed.
- A message list, which holds details of each message present on the system.
- Queue tables, which hold details of all queues configured on the system .
- The queues. These are lists of all the messages on each queue sorted in the order that they will be processed.
- A registered reader table, which holds a record of each queue reader that has registered with the Queue Manager.

As we shall see later, you can use the new `t` option in `omqdump` to get a text dump of the Queue Manager's memory. Obviously, this option is only available when the Queue Manager is running. I've included an example of the output from this option in the `omqdump` section of this OTN.

After the initial disk access at start up, the Queue Manager *almost* never accesses disk again; all further transactions are performed via IPC messages. I say almost never, because if it ever adds a message to the special `POISON` queue, the Queue Manager itself will need to access the message's pool file record to update the queue information. The `POISON` queue is discussed later in this section.

When the Queue Manager reads the message details into memory, it assigns a unique ID to each message in the Message Pool. This sequence number is called the `Message Pool ID` or `QPoolId` and it continues to increase as new messages are submitted to the Queue Manager to be put on queues. The sequence is reset when OpenMail is restarted.

We can see the `QPoolId` of each message if we use the `t` option of `omqdump`. The following excerpt shows the part of a text dump of the Message Pool which relates to the message with the `QPoolId` of 584 (decimal).

```
Pool File=57
Record Num=7
Offset=0x00001604
FileName=~/data/0000005/000084u
Flags=
NextRecord=NONE
Create Time=926005835  05.06.99 16:50:35
QPoolId=584
message flags=0x00000000
Queue=LOCAL
Priority=0
Queue Flags=0x00000000
Defer Time=NONE
Submit Time=926005835  05.06.99 16:50:35
```

Here is the corresponding excerpt from a text dump of the `LOCAL` queue in the Queue Manager's memory:

```
Item Ref          = ~/data/0000005/000084u
Priority = 0
Item Type = 0
Submit Date/Time  = 05.06.99/16:50:35
Mail Date/Time    = 01.01.70/00:00:00
Entry Number=227
Sequence Num=971
Message Pool Id=584
Message Pool File=57
Message Pool Rec=7
Failed processing attempts=0
```

Don't worry about what it all means at this stage. The important thing to note is that this ID is called `QPoolId` in the first and `Message Pool Id` in the second!

The Queue Manager arranges the in memory messages in the correct queues and in priority order. As in the previous release, urgent messages (priority = 2) on a queue will be processed before normal priority messages (priority = 0) and normal priority before low priority messages (priority = 1). Where several messages have the same priority setting, the order in which the messages were submitted to the queue determines their place on the queue.

Note: The messages are only ever sorted into order in the Queue Manager's memory. The message records in the Message Pool are never sorted.

The IPC mechanism which the Queue Manager uses to communicate with other processes is very similar to that of the Container Access Monitor and the Item Structure Server. At start up, the Queue Manager sets up two IPC message queues; one for handling requests (the input queue) and one for responses (the output queue). If you type:

```
omsetsvc -r qmgr
```

you will see the ID of these IPC queues:

```
root@kuda[] #omsetsvc -r qmgr
Details for subsystem Queue Manager:
Service Number                          = 100
...
Context dependent information: 904 755 0 0 0 0 0 0
Auxiliary processes                     = 0
PID's of auxiliary processes: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

The first number, `904`, is the input queue and `755` identifies the output queue.

If you then use the `ipcs` command:

```
ipcs -q
```

you will see that these queues do in fact exist:

```
root@kuda[] #ipcs -q
IPC status from /dev/kmem as of Mon May 17 16:27:50 1999
T      ID     KEY        MODE        OWNER     GROUP
Message Queues:
q       0 0x3c1c06f5 -Rrw--w--w-      root      root
q       1 0x3e1c06f5 --rw-r--r--      root      root
q     752 0x00000000 -Rrw-rw----  openmail  hpoffice
q     753 0x00000000 --rw-rw----  openmail  hpoffice
q     904 0x00000000 -Rrw-rw----  openmail  hpoffice
q     755 0x00000000 -Rrw-rw----  openmail  hpoffice
q     556 0x00000000 -Rrw-rw----  openmail  hpoffice
```

It should be obvious by now that the Queue Manager daemon is critical to the OpenMail system. Just as `omctmon` needs to be running before the message store can be accessed, the Queue Manager needs to be running before the queues can be accessed, even using `omqdump`.

A question you may be asking is "What is the maximum number of messages that can be in memory ?" This depends on the size of the Queue Manager's data area. The queues can grow to fill the Queue Manager's data area, which is determined by the `MAXDSIZE` kernel parameter. The default size is 64 Mbytes but, if you have a large system, you have probably already increased this for `omscan`.

To give you some idea of how many messages could be held in queues on a system with the default data area size: Each message entry occupies approximately 160 bytes and the default data area of a process is 64 Mbytes, so you could have roughly 400,000 messages on queues before the Queue Manager dies of exhaustion! Remember, this is *in transit* messages we are talking about, not messages in the Message Store, so it is extremely unlikely that you will ever hit this limit.

For each message on a queue, the Queue Manager will hold the following information:

- The `ITEM_ REF` of the cut down message container.
- The message priority setting.
- The message's `Message Pool ID` (also known as the `QPoolId`).

What happens if the Queue Manager dies?

In the unlikely event of the Queue Manager dying, the services trying to communicate with it will gradually die too. Before exiting (provided you haven't done a `kill -9` on it), the Queue Manager puts an IPC shutdown message on its IPC queue, then logs the state of its memory and other debugging information to a file, `~openmail/logs/qmgr.tomb`. This is a text file, which you can `cat` or `more` and is very similar in format to the output of `omqdump` when you use the `t` option to dump the Queue Manager's memory (we will be looking at this output in the `omqdump` section). The additional debugging information given in this file is really only of use to the lab engineers. So if you raise a support call, which requires in-depth troubleshooting, be aware that this is one of the files you may be asked for.

If the Queue Manager dies, you must shutdown OpenMail using `omshut` and restart it using `omrc`. Do not try to `omon` Queue Manager; it needs to rebuild the queues in its memory and this only happens when you use `omrc`.

What happens to the messages on the queues if the Queue Manager dies? Before a process sends a request to the Queue Manager to put a message on a queue, it writes the updated information to that message's record in the Message Pool. The information includes which queues the message is on. When the Queue Manager restarts, it will read this information and

put the message on the appropriate queues, ready to start from the point at which it died. This is, of course, assuming you have fixed the problem which caused the Queue Manager to die in the first place!

Starting Up Queue Manager If The Message Pool Is Corrupt

The tweak `QM_DONT_READ_MESSAGE_AT_START=TRUE` prevents the Queue Manager from reading the Message Pool at start up.This may prove useful if the Message Pool is corrupt, but you need to keep the messaging service running. Then you can schedule time later on to investigate the problem.

Warning! There is currently a fatal problem that could arise when you use this tweak. I have documented the scenario at the end of this section. Please read this carefully before using this tweak on your production system.

When this tweak is used, the existing messages remain in the Message Pool at start-up but are not read into memory by the Queue Manager and are therefore not processed. You will not be able to see them when viewing the queues using `omqdump`, but you can use the `t` option in `omqdump` to produce a text dump of the Message Pool. This will provide information about all the the contents of the Message Pool, including queued messages from the session before the tweak was set. From this output you can find out the cut down message container filename for each message. Output from `omshowlog` should help you what the problem is. Then if you need to access a file, you can use `omqdump` with the new `O` option, which lets you open a message file.

If you have set this tweak, it is a good idea to restart OpenMail using `omrc -n`, then dump the Message Pool contents to a file immediately, before starting the OpenMail services. This will ensure you have a record of which entries relate to the previous OpenMail session. Otherwise, when the services start up and new entries are added to the Message Pool, it will be more time consuming to determine which messages relate to which session.

Warning! At the time of writing, there is a problem you should be aware of before you decide to use this tweak. As we have already discussed, each new message is given a `QPoolID` number, which must be unique in the Message Pool. When OpenMail is started the `QPoolID` sequence is reset...for all new messages. When you use the `QM_DONT_READ_MESSAGE_AT_START` tweak, you could end up with `QPoolID` numbers which are not unique. If you unset the tweak and restart OpenMail, the Queue Manager finds two identical `QPoolID` numbers and dies! The error message logged in the event log makes it quite clear what the problem is and the offending message records in the `msgpool` files can be located by looking at the `~openmail/logs/qmgr.tomb` file. Currently, the only way round this problem is either to make sure there are no duplicate `QPoolID`'s before you shut down OpenMail and unset the tweak, or to use `omxed` to change the `QPoolID` of one of the messages in the Message Pool.

The POISON Queue

This special queue is for messages which have a nasty habit of killing off queue reader processes. Only the Queue Manager ever puts a message on this queue.

If the Queue Manager gives a reader a message and the reader dies. The Queue Manager notices that the reader has not sent a message back to say that it has processed the message. The Queue Manager then defers the message for 30 seconds, then gives it to a reader again. If the reader dies again, the Queue Manager defers the message for 30 seconds, then gives it to a reader a third time. If the reader dies again, the Queue Manager assumes the message is 'poisonous' to that reader process and puts it on the `POISON` queue.

The Queue Manager then updates the pool file record for this message to show it is on the `POISON` queue. Apart from at start up, this is the only time the Queue Manager accesses disk.

As with the ERROR queue, you will have to monitor the POISON queue and investigate any messages on it manually. You can use omqdump to access messages on this queue.

The POISON queue facility can limit the disruption to a messaging system, if a service dies and is restarted without the cause of the problem being discovered. If you are running auxiliary readers, let's say four Service Router processes, a 'poisonous' message may kill three before being assigned to the POISON queue, but you will still have one Service Router running.

Currently, there is no way of changing the number of readers that can die before a message is put on the POISON queue.

The Idel Queue And The Item Delete Daemon

In previous releases, when OpenMail had finished processing a message, the last process to handle the message was responsible for deleting it. Deleting a message involves removing a number of files, which involves a significant amount of synchronous disk i/o. Obviously, if reader processes are busy deleting messages, their capacity to process new messages is severely impacted.

The new IDEL queue and its associated reader, the Item Delete daemon (idel.server), provide a new message deletion system. The Item Delete daemon is a NON-STOP daemon. It uses IPC to request from the Queue Manager the item references of messages on the IDEL queue, in the same way as other services get messages from a queue. Currently, there can only be one Item Delete daemon.

With this new system, removing items happens in the background of other system activity. The responsibility for message deletion is removed from the service processes, so performance is not impacted. When a system is particularly busy, the Item Delete daemon will not be able to delete messages as quickly as during quieter periods, so you may see a high number of messages on the IDEL queue for a time. This is fine and will not affect message delivery times.

The IDEL queue is a 'pseudo' queue in that, when the queue part of a message's pool file record is empty (that is, the message is not on any queue), the Queue Manager notes that it is not attached to any queue and puts it on the IDEL queue. The IDEL queue is never written into a message's pool file record. The message is then picked from this queue by the Item Delete daemon, which deletes the message record from the pool file and the cut down message container and contents files from the Message Store.

Before deleting the pool record, the process checks that there are no entries in the queue part. Before deleting the message files from the Message Store, the process reduces the access count in the cut down message container by one. If the count is zero, the process deletes the file.

The way the access count in the cut down message container works has been changed slightly. In previous releases, this count reflected the number of queues the message was on. Now the count can be 0 (zero), 1 or 2. 0 (zero) means the message has no references in the Message Pool and can therefore be deleted. 1 means that it has one reference in the Message Pool. On rare occasions, it may be 2. This may happen with a message that has been deferred by a user and the message is then put on the deferred message list by the Service Router. (Note this is different from the Deferred Mail Manager facility). The access count is reduced by one when the message is eventually sent, that is, added to the Message Pool, and will subsequently be reduced to 0 (zero) when processing is complete.

You can test the access count by using the p option in omqdump to put a message on three different queues. Now 'g'et the message and 'r'ead it. The access count in the first (message container) part of the output should be 1, even though the message is on three queues. When you put the message on the queues, the message's pool file record was updated to show the queues. There is still only one reference to the message in the Message Pool.

Here's a summary of the actions performed by the Item Delete daemon:

- At start up, it is started by the Queue Manager and registers with the Queue Manager as a reader of the IDEL queue.

- It requests a message from the `IDEL` queue.
- When a message is supplied by the Queue Manager, the Item Delete daemon removes the message from the pool file, provided the queue names list is empty.
- It decreases the access count in the cut down message container.
- If the access count is zero, the message and all its content parts are deleted.
- The Queue Manager is informed that the message is no longer on the `IDEL` queue.
- [The Queue Manager then removes all references to the message from its memory].

What happens if the Item Delete daemon dies while deleting messages? First of all, you can restart the Item Delete daemon by typing:

```
idel.server
```

The last thing that happens is that the Queue Manager removes all references to the message from its memory. So, up until this point, the Queue Manager can still give the reference of this message to the Item Delete daemon. This daemon is pretty laid back about things that don’t exist...if the pool file record does’t exist, it will just continue to delete the other things it has on its list, that is, the message files. It does not mind if some of these do not exist either.

Queue Reader Processes

When a service is started, it sends an IPC message to the Queue Manager to register as a reader of its associated queue. The service then waits for the Queue Manager to send it a message to process. If there is a message on that queue, the Queue Manager sends details of the message to the service.

The service processes the message and updates the pool file record with the next queue(s) the message needs to be put on. Processing may also include creating new messages, if there are BCCs, for example. Once the changes are sync'd to disk, the service informs the Queue Manager of the changes. The Queue Manager updates its queues and sends the service the next message in its queue.

Some services, such as DirSync, can have a timeout set. This means that, if there are no messages for them in the timeout period, they will go off and do some of the other processing they are responsible for.

Note: The following tweaks concerned with queue readers are now obsolete or not advised: `Q_WAKE_READER`, `Q_TIME_OUT.`

`Q_WAKE_READER` is now obsolete. In the previous release, processes used signals to wake up the next queue reader process. It was possible to set `Q_WAKE_READER` so that the main services were not signalled when a message was placed on a queue. In combination with `Q_TIME_OUT`, this allowed a process to be idle and only process messages at fixed intervals. This feature was mainly for low end systems, which where short of memory. As this is not really a problem anymore, the functionality is not implemented in the Queue Manager.

With the new system and, in particular, the way the the Queue Manager manages the scheduling of the reader processes, it is not necessary, or advisable, to set `Q_TIME_OUT`.

Multiple Readers For A Queue

In the previous release, a queue could have many processes writing to it but only one process reading from it. As a result, the `ROUTER` and `LOCAL` queues tended to be a bottleneck in large, heavily used systems. In B.06.00, to take full advantage of the additional efficiency and speed introduced by the Queue Manager, the main queues can have multiple reader processes.

Queues that can have multiple readers configured are:

- Local Delivery
- Service Router
- `xport.out` (sendmail interface)
- `unix.out` (Internet Gateway)

Even if multiple copies of the above services are configured, only one copy of the following sub-services will be configured:

- `error.manager` This is a sub-service of Local Delivery.
- `defer.manager` This is a sub-service of the Service Router.
- `licence.server` This is a sub-service of the Service Router.

When you start OpenMail using `omrc`, only one reader will start for the above queues. This is called the *main* reader for the queue. If you then configure additional readers for a particular queue, these are called *auxiliary* readers. It is important to understand this distinction, because some of the monitoring output refers only to auxiliary readers and does not include the main reader.

How To Configure Multiple Reader Processes

Before configuring auxiliary reader processes for a queue, you must turn the service off using `omoff`. You then use the `omsetsvc` command to configure auxiliary readers:

`omsetsvc -x <service> <no. of auxiliary readers>`

The total number of readers for this queue will then be :

main reader + number of auxiliary readers

For example,

```
omsetsvc -x sr 2
```

will configure two auxiliary Service Router processes, giving a total of three Service Router processes all together.

Theoretically, you can configure up to a maximum of 20 auxiliary readers for a queue. However, before rushing off and starting lots of extra readers, thinking this is the answer to all your throughput problems, I strongly suggest you read the section below (see “Choosing The Optimum Number Of Reader Processes” on page 20)!

When you `omon` or `omoff` the reader service, the associated auxiliary readers are also started or stopped. There is currently no way of closing down some of the readers. To run fewer readers, you need to `omoff` the service, configure the required number of auxiliary readers you want using `omsetsvc -x` and `omon` the service again.

When the service is started, the reader processes sit waiting for the Queue Manager to hand them messages to process. If there is no backlog on the queue, the Queue Manager will hand the message to the main process. As soon as there is a backlog, of whatever size, the Queue Manager will start handing messages to the auxiliary processes.

How To Monitor Multiple Reader Processes

There are various ways you can monitor the reader processes but the main tools are `omstat` and `omsetsvc`. `omstat -s -p` shows you the number of auxiliary processes running and their status.

Start the Service Router, after configuring two auxiliary Service Router processes, as described above. Then type `omstat -s -p`. Here’s an excerpt of the output you should see:

```
root@kuda[tmp] #omstat -s -p
Service Router              Started             15:21:47       2
Local Delivery              Disabled            05.06.99       0
Internet Mail Gateway       Disabled            04.29.99       0
X400 Interface              Disabled            04.29.99       0

...
```

`Started` means auxiliary Service Router processes have been configured and are all running. The `2` in the right hand column tells us the number of auxiliary processes running. Remember, this does not include the main Service Router process, so in total there are three Service Router processes running.

`Disabled` in the second column means that no auxiliary readers have been configured. For example, if we were to re-configure the Service Router so that there are no auxiliary processes, then `omon` the service. The output from `omstat -s -p` is:

```
root@kuda[tmp] #omstat -s -p
Service Router              Disabled            16:03:17       0
Local Delivery              Disabled            05.06.99       0
Internet Mail Gateway       Disabled            04.29.99       0
X400 Interface              Disabled            04.29.99       0
```

This may seem a little disconcerting at first, because the main Service Router process is running. You just have to remember that the `-p` option gives you the status of auxiliary processes only. `omstat -s` will show you that the main process is started.

`Stopped` in the second column means that auxiliary readers have been configured but they are all stopped (`omoff`). If some have died, it would say `partially aborted` and the number on the right would show how many are still running. If all the auxiliary readers have died, this column would say `aborted`.

You can use `omsetsvc -r <service>` to view the reader configuration details. In the example below, I turned the Sevice Router off first, then typed `omsetsvc -r sr`:

```
root@kuda[] #omsetsvc -r sr
Details for subsystem Service Router:
Service Number                      = 2
Number of components                = 2
Logging Level                       = 9
Audit logging Level                 = 0
Has an input queue?                 - YES
Queue name                          = ROUTER
Show details from omstat?           - YES
Subsystem can be enabled?           - YES
Required state                      - Disabled
Last state change (on/off)          = 13:25:56
Last delayed off time               = 13:25:56
Startup prog name                   = ~/bin/service.router
Shutdown program name               = ~/bin/shut.queue -q ROUTER -d %d -s 2
Status program name                 =
PID's of subsystem processes: -1 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Nice Level                              = 1
Additional Resolve Flag                 = 0
Subsystem is controlled by 'all'        - YES
Minimum temporary processes             = 0
Maximum temporary processes             = 0
Context dependent information: 0 0 0 0 0 0 0 0
Auxiliary processes                     = 2
PID's of auxiliary processes: -1 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

The last two lines show the number of auxiliary readers configured and their PIDs (`-1` means the process is currently off).

If we start the Service Router and take another look at the `omsetsvc -r sr` output:

```
root@kuda[] #omsetsvc -r sr
Details for subsystem Service Router:
Service Number                          = 2
Number of components                    = 2
Logging Level                           = 9
Audit logging Level                     = 0
Has an input queue?                     - YES
Queue name                              = ROUTER
Show details from omstat?               - YES
Subsystem can be enabled?               - YES
Required state                          - Enabled
Last state change (on/off)              = 13:48:46
Last delayed off time                   = 13:25:56
Startup prog name                       = ~/bin/service.router
Shutdown program name                  = ~/bin/shut.queue -q ROUTER -d %d
-s 2
Status program name                     =
PID's of subsystem processes: 14511 14512 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0
Nice Level                              = 1
Additional Resolve Flag                 = 0
```

```
Subsystem is controlled by 'all'        - YES
Minimum temporary processes             = 0
Maximum temporary processes             = 0
Context dependent information: 0 0 0 0 0 0 0 0
Auxiliary processes                     = 2
PID's of auxiliary processes: 14513 14514 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0
```

In the line "`PID's of subsystem processes:`", the first number, `14511`, is the PID of the main Service Router process. (The second PID on this line, `14512`, is the PID of the sub-service, `defer.manager`). On the line "`PID's of auxiliary processes:`", we can see the PIDs of the two auxiliary Service Router processes, `14513` and `14514`.

To double check these readers are indeed running, we can use the `ps` command and `grep` for `router`:

```
root@kuda[] #ps -ef | grep router
openmail 14511     1  0 14:42:51 ?       0:00 service.router
openmail 14513 14511  0 14:42:51 ?       0:00 service.router
openmail 14514 14511  0 14:42:51 ?       0:00 service.router
```

Choosing The Optimum Number Of Reader Processes

Important! Your system must have enough spare capacity for the extra processes you want to configure, as each new process requires a certain amount of cpu and will incur disk i/o. Configuring auxiliary reader processes on a system, which is already running near capacity, will reduce performance.

A good general rule is to configure the minimum number needed to stop backlog building up on the queue. Increase the readers one at a time, then monitor performance for an extended period to ensure the system is not overloaded. It is particularly important to monitor the disk activity on the `~openmail/msgpool` directory. Check disk i/o, cpu and disk i/o!

For some valuable information on how to optimize your configuration for OpenMail, see the document: **Configuration Planning Guide** (ID 200-0200) on the OpenMail website.

The `s` and `T` options in `omqdump` allow you to measure the load and throughput of the queues. We will cover these options in the `omqdump` section.

Configuring a large number of auxiliary processes all at once could also lead to reduced performance. This is because there is a small overhead for idle processes, so excessive numbers will increase the cpu load significantly.

If you increase the number of reader processes for the Internet gateway (`unix.out`) or sendmail interface (`xport.out`), sendmail may start lots of child processes to handle the increased message throughput and overload the system. You can configure `unix.out` or `xport.out` to wait until each message is processed by sendmail before sending the next one, thus limiting the number of sendmail processes running. However, this may reduce throughput, so you may need to increase the number of auxiliary readers to achieve the required performance.

This configuration option is documented in the files `~openmail/sys/unix.mapper` and `~openmail/sys/xport.mappers/XPORT`.

Kernel Parameters

The release notes for B.06.00 give recommendations for kernel parameter settings. As configuring multiple reader processes may hit kernel parameter limits, I suggest you read the recommendations carefully before configuring lots of auxiliary readers. The parameter most likely to be affected is `MAXUPROC`.

Logging Reader Processes

When there was only one reader process per queue, it was unnecessary to log the PID of the reader process in `omshowlog`. However, with multiple readers it is important to know which reader is actually writing a particular log entry. To enable us to distinguish which entries belong to which reader, there is a new `-P` option for `omshowlog`. This reports the PID of the process which logged each entry.

To get the following example, I turned the Service Router on and Local Delivery off and configured two auxiliary readers for Local Delivery:

```
omsetsvc -x local 2
```

I then sent 100 messages to Mr Admin.

When all the messages were on the `LOCAL` queue, I turned Local Delivery on. Remember, this turns on all the auxiliary processes too.

The reason for sending as many as 100 is to cause a backlog on the queue (the classroom machines are not busy, so a few messages will go through very quickly). If there is no backlog on the queue, the Queue Manager will just hand the message to the main process, so you would not see the auxiliary processes being handed messages.

To find out the PIDs of the processes, I typed:

```
omsetsvc -r local
```

Then I ran `omshowlog` with the `-P` option to show the PIDs of the reporting processes:

```
omshowlog -l 9 -P -p 10
```

The following is just an excerpt of the output to show you the different processes logging:

```
REPORT                        Local Delivery(Local Delivery) 05.15.99
14:26:31
[OM 7604] Finished delivery of message
Pid of logging process: 15400


REPORT                        Local Delivery(Local Delivery) 05.15.99
14:26:31
[OM 7608] Finished delivery to recipient
   Mr Admin/canberra,class
Pid of logging process: 15401


REPORT                        Local Delivery(Local Delivery) 05.15.99
14:26:31
[OM 7604] Finished delivery of message
Pid of logging process: 15401


REPORT                        Local Delivery(Local Delivery) 05.15.99
14:26:31
[OM 7608] Finished delivery to recipient
   Mr Admin/canberra,class
Pid of logging process: 15399


REPORT                        Local Delivery(Local Delivery) 05.15.99
14:26:31
[OM 7604] Finished delivery of message
Pid of logging process: 15399
```

```
REPORT                          Local Delivery(Local Delivery) 05.15.99
14:26:31
[OM 7603] Started delivery of message
Pid of logging process: 15400
```

If you use another new option, `-i <PID>`, together with `-P`, `omshowlog` will show you only the entries logged by the reader process with the specified PID. The output below was returned when I typed:

```
omshowlog -l 9 -P -i 15400 -p 10
```

```
REPORT                          Local Delivery(Local Delivery) 05.15.99
14:26:28
[OM 7601] Local Delivery Started Up
Pid of logging process: 15400


REPORT                          Local Delivery(Local Delivery) 05.15.99
14:26:28
[OM 7603] Started delivery of message
Pid of logging process: 15400


REPORT                          Local Delivery(Local Delivery) 05.15.99
14:26:28
[OM 7618] Creator of message
   Mr Admin/canberra,class
Pid of logging process: 15400


REPORT                          Local Delivery(Local Delivery) 05.15.99
14:26:28
[OM 7605] Current message Id
   H00000650000234d.0926687601.kuda.pwd.hp.com
Pid of logging process: 15400
```

To help debug more difficult problems, the current value of the Unix variable, `errno`, is reported at level ‘error’ or above. `Errno` reports error messages from system calls. The value reported may refer to the error reported in the OpenMail logs, but it may not! This is really just an extra bit of information for the lab engineers.

```
ERROR                           Local Delivery(Error Manager ) 02.16.99
17:09:29
[OM 8807] No Error Manager Configured.  Please Configure One
Pid of logging process: 8454

ERROR                           Administration(omadmidp      ) 02.18.99
18:48:07
[SYS 9] Bad file number
Pid of logging process: 5791
Current errno value: 9
```

Troubleshooting Tools

Throughout this OTN, I have referred to ways in which you can see what is happening in various parts of the queue system:

- `omstat -s -p` to see the status of the auxiliary processes.
- `omshowlog -l <level> -P` to see the PIDs of the processes logging the entries and `omshowlog -l <level> -i <PID> -P` to list only the entries logged by the process with the specified PID.
- `omsetsvc -r <service>` to see the configuration details of a service, including the PIDs of the auxiliary processes. (`omsetsvc -r qmgr` allows you to identify the Queue Manager's IPC queues).

Two further troubleshooting tools are the Queue Manager's tombstone file and `omfreeq`, which are covered in the sections below.

However, as before, the main tool for troubleshooting queues is `omqdump`. In addition to the queue manipulation features, it now provides load and throughput figures to help you monitor performance. `omqdump` also has a text dump option which enables you to see the contents of the `~openmail/msgpool` files and the Queue Manager' memory. The new `omqdump` options and changes to the way existing options work are covered in the `omqdump` section.

The Queue Manager's Tombstone File

Should the Queue Manager ever die, it creates a *tombstone* file in the logs directory, `~/logs/qmgr.tomb`. This, together with messages in the logs, should help you diagnose the problem. As the contents of this file is very similar to the text dump of the Queue Manager's memory, I suggest you look at that in the `omqdump` section covering the `t` option to get a general understanding of what is in this file. The additional information given in this file is really only of use to the lab engineers.

omfreeq

`omfreeq` queue has a `-p` option to remove a reader PID from a queue.

A situation can arise where a reader dies and the PID is reused by another process, without the Queue Manager being aware that the reader has died. The Queue Manager still thinks the PID belongs to the queue reader. You can use `omfreeq` to tell the Queue Manager to release the reader from the queue. For example:

```
omfreeq -q local -p 12345
```

removes reader 12345 from the LOCAL queue.

omqdump

To access the queues, `omqdump` communicates with the Queue Manager using IPC messages, so the Queue Manager needs to be running if you want to use the queue manipulation functions of `omqdump`. However, Queue Manager does not need to be running to view the contents of the Message Pool files.

You can use `omqdump` to access the special `POISON` and `IDEL` queues in the same way as you would access any of the other queues.

Note: It is still advisable to turn off the reader service before manipulating messages on its queue. Theoretically, you could access messages on a queue while readers are processing messages from it, but it would be extremely difficult to find or manipulate the right message. Walking the queue accurately, for example, would be almost impossible.

I've underlined the new options in the menu below:

```
Please select an option:
  s: Show summary of queues          o: Output opened msg to files
  l: List (browse) msgs on queue     i: Input a msg from files
  m: Move opened msg between queues  g: Get (serial open) next msg
  M: Move msgs between queues        G: Get (serial open) nth+1 msg
  d: Delete opened msg from queue    r: Read (browse) opened msg
  D: Delete msgs from queue          c: Close opened msg
  v: View status of queue            f: Find (priority open) next msg
  a: Access opened msgs info         p: Put opened msg on a queue
  O: Open message file               t: Text dump
  L: list configured queues        C: Copy msgs
  T: List queue throughput averages  Z: Zap (forcefully delete) msg
  q: Quit
```

s

Although there was a summary option before, the output has changed considerably. In addition to the number of messages on a queue, you are now given:

- How many of those messages are being processed at this moment (`Active`).
- How many messages have been put on this queue since OpenMail was last started (`Ever`).
- The average number of messages on this queue over the last 1, 5 and 15 minutes (`Load`).

To show some figures, turn off the Service Router and Local Delivery and send about 100 messages to Mr Admin, or a local user. Configure two auxiliary readers for the Service Router:

```
omsetsvc -x sr 2
```

Start `omqdump` in one terminal window. In another, turn on the Service Router, then immediately type `s` in the `omqdump` window. You should see something like the following:

```
Option?s
QUEUE          Messages     Active       Ever       Load [1min,5min,15min]
BB                    0          0          0       0.00       0.00       0.00
DESK                  0          0          0       0.00       0.00       0.00
DIRSYNC               0          0          0       0.00       0.00       0.00
DMM                   0          0          0       0.00       0.00       0.00
DUMP                  0          0          0       0.00       0.00       0.00
ERRMGR                0          0          0       0.00       0.00       0.00
ERROR                 3          0          0       3.00       3.00       3.00
LICENSE               0          0          0       0.00       0.00       0.00
LOCAL                55          0        728      21.61      19.70      12.74
NOTES                 0          0          0       0.00       0.00       0.00
OMX400                0          0          0       0.00       0.00       0.00
PRINT                 0          0          0       0.00       0.00       0.00
REQ                   0          0          0       0.00       0.00       0.00
RESOLVE               0          0          0       0.00       0.00       0.00
ROUTER               66          3        769      95.08      47.29      23.15
SMERR                 0          0          0       0.00       0.00       0.00
SMINTFC               0          0          0       0.00       0.00       0.00
SMS                   0          0          0       0.00       0.00       0.00
TEST                  0          0          0       0.00       0.00       0.00
UNIX                  0          0          0       0.00       0.00       0.00
X400                  0          0          0       0.00       0.00       0.00

Special queues.
IDEL                  0          0        727       0.00       0.00       0.00
POISON                0          0          0       0.00       0.00       0.00
Done
```

Notice that there are three active messages on the `ROUTER` queue. That is because there are three Service Router processes running; the main reader and two auxiliary readers.

Over time, the load figures across the load columns will become closer, then reduce, as the surge of messages finishes. If the load figures for a queue are constantly high and your system has spare capacity, you could consider configuring an auxiliary reader for that queue.

So what number might be considered as high? It depends on the message delivery times expected. However, if you divide the average load figure by the number of readers and get a figure which is more than 20 times the number of readers, then the load is probably high.

To collect load figures over a period of time, you could set up a script to run `omqdump` as a batch job at regular intervals and use the `s` option.

T

The T option gives similar output to the `s` option, but provides throughput figures instead of load figures, that is, the number of messages processed on this queue in the last 1, 5 and 15 minutes. For the example below, I turned on the Local Delivery in one terminal window and quickly typed `T` in the `omqdump` window:

```
Option?T
QUEUE          Messages     Active       Ever       Throughput
                                                  [1min,5min,15min]
BB                    0          0          0       0.00       0.00       0.00
DESK                  0          0          0       0.00       0.00       0.00
DIRSYNC               0          0          0       0.00       0.00       0.00
DMM                   0          0          0       0.00       0.00       0.00
DUMP                  0          0          0       0.00       0.00       0.00
ERRMGR                0          0          0       0.00       0.00       0.00
ERROR                 3          0          0       0.00       0.00       0.00
LICENSE               0          0          0       0.00       0.00       0.00
LOCAL                86          1        763      28.22      31.18      31.72
NOTES                 0          0          0       0.00       0.00       0.00
OMX400                0          0          0       0.00       0.00       0.00
PRINT                 0          0          0       0.00       0.00       0.00
REQ                   0          0          0       0.00       0.00       0.00
RESOLVE               0          0          0       0.00       0.00       0.00
ROUTER                0          0        835       0.00       2.30      30.65
SMERR                 0          0          0       0.00       0.00       0.00
SMINTFC               0          0          0       0.00       0.00       0.00
SMS                   0          0          0       0.00       0.00       0.00
TEST                  0          0          0       0.00       0.00       0.00
UNIX                  0          0          0       0.00       0.00       0.00
X400                  0          0          0       0.00       0.00       0.00

Special queues.
IDEL                  0          0        762      27.47      30.24      30.74
POISON                0          0          0       0.00       0.00       0.00
Done
```

This time there is only one active message, because no auxiliary readers have been configured for Local Delivery.

O

The `O` option allows you to open the `ITEM_REF` of a cut down message container directly in `omqdump`. You can then put the message on the appropriate queue.

You could use this to take a message file off a backup, or upgrade a message that `omqconv` has failed to upgrade for some reason. If you look in `/tmp/ominstall.log`, the details of the failed message will be logged, including the `ITEM_REF` of its cut down message container. In `omqdump`, type `O` and you will be asked for the filename:

```
Option?O
Msg File Name (/tmp/omqdmsg):
```

Type the ITEM_REF of the message, and it will be opened by omqdump, ready for you to 'p'ut on the required queue. The message is now upgraded.

L

The L option lists the configured queues:

```
Option?L
BB
DESK
DIRSYNC
DMM
DUMP
ERRMGR
ERROR
LICENSE
LOCAL
NOTES
OMX400
PRINT
REQ
RESOLVE
ROUTER
SMERR
SMINTFC
SMS
TEST
UNIX
X400
Done
```

As omqdump obtains this list from the file ~openmail/msgpool/poolctrl, and not from the Queue Manager, Queue Manager does not need to be running to use this option. Note this list does not include the POISON or IDEL queues. These new queues are, however, included in the output from s option of omqdump.

C

The C option allows you to copy multiple messages from one queue to another. It works in the same way as the other options which allow you do 'multiple' operations.

This option creates new copies of the message(s). If you want to put a message on several queues, that is, the queue part of that message's pool file record will have several entries, use the p option not C.

Z

Z(ap) removes a message from a queue without opening the message.

With the delete options, omqdump tries to open the cut down message file. If this file is corrupt or missing, you cannot use the d option. The message would then continue to have a reference in the Message Pool.

With the Z option, omqdump removes the message record from the pool file without attempting to open any of the associated files, then tells the Queue Manager that this message is no longer present.

Changes To The Browse Options (l and r)

You will see some differences in the output from the l'ist' and r'ead' options. Here's an example of reading a message on the LOCAL queue:

```
Option?r
Msg Handle (2000):

Msg File = ~/data/0000004/000083k
```

```
Access Count = 1
TF File = ~/data/0000010/00007vq
TF Info Flags = 7
Filter Route =
Attach File #1 = ~/data/0000004/000083l:1
Attach File #2 = ~/data/0000004/000083l:2

HEADER            (DN)  1  0  0  0  0x2000400  0x3  1  0x0
  ** Warning - Insufficient fields present  (Record No. 1)
CREATOR           (DN)  0  101  0  0  0  0  "S=Admin/G=Mr/OU1=canberra/
OU2=class"  ""  ""
SUBJECT           (DN)  "TestMsg (1) Thu-16:50:24"  ""  "IA5"  ""  ""
CREATE_DATE       (DN)  99/5/6 15:50.26+60
MSG_INT_ID        (DN)  0  "H000006500002073.0926005826.kuda.pwd.hp.com"
"H000006500002073.0926005826.kuda.pwd.hp.com"
ORIGINAL_EITS     (DN)  0  0  0  0  1  "2.6.1.4.0"
ITEM_CREATE_DATE  (DN)  99/5/6 15:50.25+60
CONTENT_FILE      (DN)  1166  1166  0x2  136  0  ""  ""  ""  ""  ""  "IA5"
"" ""  ""  ""  ""  ""  ""  ""  ""
ITEM_CREATE_DATE  (DN)  99/5/6 15:50.25+60
CONTENT_FILE      (DN)  1167  1167  0x0  633  0  "passwd"  "passwd"  ""
"IA5" ""  "IA5"  ""  ""  ""  ""  ""  ""  ""  ""  ""
RECIPIENT         (DN)  0x20  0  1  "LOCAL"  ""  "S=Admin/G=Mr/
OU1=canberra/OU2=class"
OPERATION_TRACE   (DN)  99/5/6 15:50.26+60  5  ""
ROUTE_TRACE       (DN)  99/5/6 15:50.30+60  0  "kuda/canberra,class"
Done
```

I have underlined the new information.

The filename of the cut down message container is now included. This means you can reconstruct a message whose transaction file has become corrupt. Just create and send a message with the same number of attachments as the corrupted message, then copy the new transaction file into the location of the corrupted one.

The message ID has been extended to ensure it is globally unique. The first part, `H000006500002073`, is the original message ID. The next part, `0926005826`, is seconds since the beginning of the epoch (1st Jan 1970) and the last part, `kuda.pwd.hp.com`, is the fully qualified domain name. You can turn off the extending of the message ID by setting the tweak, `IM_MAKE_MSG_ID_GLOBAL_UNIQUE` to `FALSE` but there is no guarantee that the shorter message ID will be globally unique.

Changes To g Option

You no longer need to exit `omqdump` to get back to the beginning of a queue! You don't go backwards, you just keep going forward till you reach the first message again. If you repeatedly use '`g`'et and '`c`'lose to walk a queue, you will know you have reached the end when you see a `TIME_OUT` message. Use '`g`'et again and you should have the first message on the queue again.

Note: Don't use `v` in between the '`g`'ets. The reason is explained below!

The v Option Does Some Unusual Things Now

If you are walking a queue using `g` and `c`, then you use `v` to see where you have got to, it will initialize the count and put you back to the beginning of the queue. With circular queues, this can make finding out where you are a little difficult, especially if you want to get to a point then '`M`'ove several messages to another queue. This is a bug and will be addressed. Meanwhile, experiment with walking the queue to familiarize yourself with how it works before '`D`'eleting multiple messages from the middle of a queue!

t

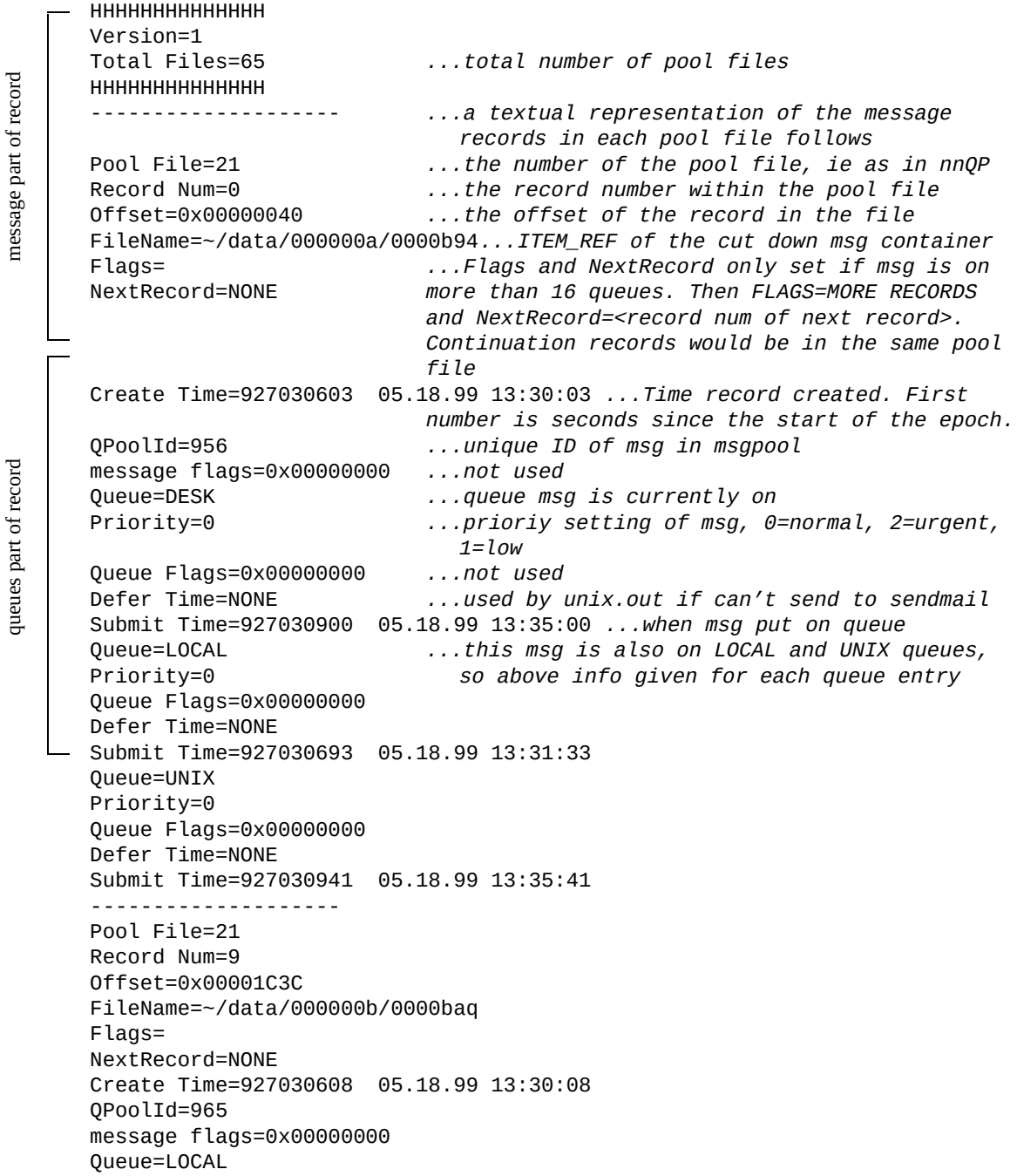
This option gives you two sub options, `m` or `q`. m creates a text description of what is in the Message Pool and q creates a text description of what is in the Queue Manager's memory. These provide particularly useful troubleshooting analysis of the system for the lab engineers.

In the following examples, I have annotated the output to help you understand what the figures relate to. We'll look at a text dump of the Message Pool first.

In `omqdump`, type:

```
Option?t
Message pool (m) or queues (q) ?  m
Msg File Name (/tmp/omqdmsg):
```

To accept the default file offered, just press `<RETURN>`. If you then `more` the file, the output will be similar to the following example:




```
HHHHHHHHHHHHHH
Version=1
Total Files=65             ...total number of pool files
HHHHHHHHHHHHHH
--------------------       ...a textual representation of the message
                              records in each pool file follows
Pool File=21               ...the number of the pool file, ie as in nnQP
Record Num=0               ...the record number within the pool file
Offset=0x00000040          ...the offset of the record in the file
FileName=~/data/000000a/0000b94...ITEM_REF of the cut down msg container
Flags=                     ...Flags and NextRecord only set if msg is on
NextRecord=NONE            more than 16 queues. Then FLAGS=MORE RECORDS
                           and NextRecord=<record num of next record>.
                           Continuation records would be in the same pool
                           file
Create Time=927030603  05.18.99 13:30:03 ...Time record created. First
                           number is seconds since the start of the epoch.
QPoolId=956                ...unique ID of msg in msgpool
message flags=0x00000000   ...not used
Queue=DESK                 ...queue msg is currently on
Priority=0                 ...prioriy setting of msg, 0=normal, 2=urgent,
                              1=low
Queue Flags=0x00000000     ...not used
Defer Time=NONE            ...used by unix.out if can't send to sendmail
Submit Time=927030900  05.18.99 13:35:00 ...when msg put on queue
Queue=LOCAL                ...this msg is also on LOCAL and UNIX queues,
Priority=0                    so above info given for each queue entry
Queue Flags=0x00000000
Defer Time=NONE
Submit Time=927030693  05.18.99 13:31:33
Queue=UNIX
Priority=0
Queue Flags=0x00000000
Defer Time=NONE
Submit Time=927030941  05.18.99 13:35:41
--------------------
Pool File=21
Record Num=9
Offset=0x00001C3C
FileName=~/data/000000b/0000baq
Flags=
NextRecord=NONE
Create Time=927030608  05.18.99 13:30:08
QPoolId=965
message flags=0x00000000
Queue=LOCAL
```

```
Priority=0
Queue Flags=0x00000000
Defer Time=NONE
Submit Time=927030695  05.18.99 13:31:35
--------------------
...etc
```

Now let's look at a textual description of the Queue Manager's memory. To create the file, you type:

```
Option?t
Message pool (m) or queues (q) ?  q
Msg File Name (~/temp/00008b0):
```

If you do not accept the default location, the filename you give must be an `ITEM_REF` in the data or temp domain, that is `~openmail/data` or `~openmail/temp`. We strongly recommend you put it in `~openmail/temp`.

In the output, he sections are in the following order and each section starts with '`====`':

`====`Active Messages (Active message list)

`====`Known messages (The message list)

`====`Queues - for each queue, status info followed by the msgs on the queue, sorted in order.

`====`Queue Readers (Registered readers)

Not all the information will be useful to you. I've added the comments in italics to help you understand the potentially useful parts.

```
more ~openmail/temp/00008b0
==ACTIVE MESSAGES==
Table Size=4                  ...max. number of msgs active at any point
------------------      there are 2 msgs being processed at this point...
Record=0                      ...record number in table
Reader Id=3                   ...same as 'record' entry in Q Readers
                                 section at end of output
Queue Name=LOCAL
Active Time=05.18.99 13:35:11
Flags=0x00000001
Item Ref          = ~/data/000000a/0000b94 ...ITEM_REF of cut down ctner
Priority = 0
Item Type = 0
Submit Date/Time  = 05.18.99/13:31:33 ...when submitted to queue
Mail Date/Time    = 01.01.70/00:00:00 ...only used by unix.out for
                                          deferred msgs
Entry Number=29
Sequence Num=1442               ...seq num given to msg when it was put on
                                   this queue
Message Pool Id=956             ...QPoolId
Message Pool File=21            ...ie pool file is 21QP
Message Pool Rec=0              ...record in pool file
Failed processing attempts=0    ...if 3 them would be on POISON queue
------------------
Record=2
Reader Id=4
Queue Name=LOCAL
Active Time=05.18.99 13:37:25
Flags=0x00000001
Item Ref          = ~/data/000000e/0000be0
Priority = 0
Item Type = 0
```

```
Submit Date/Time  = 05.18.99/13:31:37
Mail Date/Time    = 01.01.70/00:00:00
Entry Number=170
Sequence Num=1368
Message Pool Id=982
Message Pool File=21
Message Pool Rec=26
Failed processing attempts=0
====End Active=====

====Known messages===          ...All the messages in the system
                                  (not sorted)
Table size=300                 ...Available slots in the table
Table Used Size=272            ...Slots used
Free list Size=192             ...A list referencing free table slots
-----------------------        ...Details of each msg follows...
Record=0                       ...record number in message table
Message Pool Id=101            ...Same as QPoolId
Message Pool File=25           ...i.e ~openmail/msgpool/25QP
Message Pool Rec=0             ...record within ~openmail/msgpool/25QP
Access Count=1                 ...no. of queues msg is on
Flags=0x00000001               ...Flag of 0x00000002 prevents msg being
                               deleted as creator stillhas it open
Msg File=~/data/000000p/00004l4 ..ITEM_REF of cut down msg container
-----------------------
Record=12
Message Pool Id=113
Message Pool File=51
Message Pool Rec=0
Access Count=1
Flags=0x00000001
Msg File=~/data/000000q/00004p4
-----------------------
Record=29
Message Pool Id=956
Message Pool File=21
Message Pool Rec=0
Access Count=3          ...NB this message is on 3 queues. Search
                        on the Message Pool Id to find it on queues
Flags=0x00000001
Msg File=~/data/000000a/0000b94
-----------------------
Record=31
Message Pool Id=954
Message Pool File=33
Message Pool Rec=0
Access Count=1
Flags=0x00000001
Msg File=~/data/0000009/0000b81
...

Record=271
Message Pool Id=1032
Message Pool File=21
Message Pool Rec=76
Access Count=1
Flags=0x00000001
Msg File=~/data/000000n/0000bnc
=====END Messages====

==== Queues =======          ...Info about each queue and msgs on them
                                (sorted)
Number of queues=23
```

```
Current Pool ID=1055          ...highest number in QPoolId sequence
----------------
Queue Name=SMINTFC
Flags=0x00000001              ...just indicates msg in use.
                                 Flags are cumulative.
                                 0x00000002 queue suspended
                                 0x00000004 queue shutdown
                                 0x00000008 queue locked by reader
Messages this hour=0
Messages last hour=0
Current Hour= 00:00:00
Messages processed=0
Active Messages=0

+++Load Average:              ...some load stats taken at intervals,
                                 s & T options of omqdump are more useful
Load=0.000000
Decay factor=0.920044
Averge interval=60
Sample period=5
Remaining Sample period=2
Next load value=0
Last average taken at= 13:37:25
Average running for= 23:37:28
+++Load Average:
...
------>>>> The Messages       ...no messages on the SMINTFC queue
----------------
...

Queue Name=LOCAL
Flags=0x00000001
Messages this hour=25
Messages last hour=121
Current Hour= 13:37:13
Messages processed=874
Active Messages=2             ...we saw these in Active Msgs section
+++Load Average:
Load=96.679782
Decay factor=0.920044
Averge interval=60
Sample period=5
Remaining Sample period=2
Next load value=0
Last average taken at= 13:37:25
Average running for= 23:37:28
+++Load Average:
...
------>>>> The Messages
++++++++++++++++++
Item Ref          = ~/data/000000e/0000be6
Priority = 0
Item Type = 0
Submit Date/Time  = 05.18.99/13:31:38
Mail Date/Time    = 01.01.70/00:00:00
Entry Number=172
Sequence Num=1369
Message Pool Id=983
Message Pool File=21
Message Pool Rec=27
Failed processing attempts=0
++++++++++++++++++
Item Ref          = ~/data/000000e/0000bec
```

```
Priority = 0
Item Type = 0
Submit Date/Time  = 05.18.99/13:31:38
Mail Date/Time    = 01.01.70/00:00:00
Entry Number=173
Sequence Num=1370
Message Pool Id=984
Message Pool File=21
Message Pool Rec=28
Failed processing attempts=0
++++++++++++++++++
Item Ref          = ~/data/000000e/0000bei
Priority = 0
Item Type = 0
Submit Date/Time  = 05.18.99/13:31:38
Mail Date/Time    = 01.01.70/00:00:00
Entry Number=174
Sequence Num=1371
Message Pool Id=985
Message Pool File=21
Message Pool Rec=29
Failed processing attempts=0
++++++++++++++++++
...

Queue Name=DESK
Flags=0x00000001
Messages this hour=0
Messages last hour=0
Current Hour= 00:00:00
Messages processed=0
Active Messages=0
+++Load Average:
...
------>>>> The Messages
++++++++++++++++++
Item Ref          = ~/data/000000a/0000b94
Priority = 0
Item Type = 0
Submit Date/Time  = 05.18.99/13:35:00
Mail Date/Time    = 01.01.70/00:00:00
Entry Number=29
Sequence Num=2
Message Pool Id=956          ....Here's the msg on 3 queues
Message Pool File=21
Message Pool Rec=0
Failed processing attempts=0
----------------
...
Queue Name=UNIX
Flags=0x00000001
Messages this hour=0
Messages last hour=0
Current Hour= 00:00:00
Messages processed=0
Active Messages=0
+++Load Average:
...
------>>>> The Messages
++++++++++++++++++
Item Ref          = ~/data/000000a/0000b94
Priority = 0
Item Type = 0
```

```
Submit Date/Time  = 05.18.99/13:35:41
Mail Date/Time    = 01.01.70/00:00:00
Entry Number=29
Sequence Num=2
Message Pool Id=956          ....Here's the msg on 3 queues
Message Pool File=21
Message Pool Rec=0
Failed processing attempts=0
----------------
...
----------------
Queue Name=IDEL
Flags=0x00000001
Messages this hour=25
Messages last hour=121
Current Hour= 13:37:14
Messages processed=873
Active Messages=0
+++Load Average:
...
------>>>> The Messages
----------------
Queue Name=POISON
Flags=0x00000001
Messages this hour=0
Messages last hour=0
Current Hour= 00:00:00
Messages processed=0
Active Messages=0
...

====Queue Readers====          ...Info about registered queue readers...
Readers Array Size=8
------------------------
Record=0                       ...=Reader Id in Active Messages
Pid=11620                      ...PID of reader process
Registered at=05.05.99 14:00:03 ...Time process registered with Qmgr
Last Request at=05.18.99 13:37:25 ...Info about latest communication
Last Response at=05.18.99 13:37:25   with qmgr
Current Active message=NONE
Queue Name=IDEL
Flags=0x00000003               ...cumulative flags: 1=structure in use,
                                  3=reader awaiting reply, 10=shutdown
                                  req (i.e. omoff), 40=reader has q
                                  locked (i.e.omscan probably running)
TimeOut=300                    ...queue reader timeout in seconds
------------------------
Record=1
Pid=16650
Registered at=05.18.99 13:31:33
Last Request at=05.18.99 13:36:58
Last Response at=05.18.99 13:36:58
Current Active message=NONE
Queue Name=ROUTER
Flags=0x00000003
TimeOut=30
------------------------
Record=2
Pid=16651
Registered at=05.18.99 13:31:33
Last Request at=05.18.99 13:31:33
Last Response at=01.01.70 00:00:00
Current Active message=NONE
```

```
Queue Name=DMM
Flags=0x00000003
TimeOut=3600
------------------------
Record=3
Pid=16661
Registered at=05.18.99 13:34:43
Last Request at=01.01.70 00:00:00
Last Response at=05.18.99 13:35:11
Current Active message=0          ...Notice there are 2 local del readers
Queue Name=LOCAL
Flags=0x00000001
TimeOut=0
------------------------
Record=4
Pid=16687
Registered at=05.18.99 13:37:12
Last Request at=05.18.99 13:37:25
Last Response at=05.18.99 13:37:25
Current Active message=2
Queue Name=LOCAL
Flags=0x00000001
TimeOut=0
------------------------
Record=5
Pid=16686
Registered at=05.18.99 13:37:12
Last Request at=05.18.99 13:37:12
Last Response at=01.01.70 00:00:00
Current Active message=NONE
Queue Name=ERRMGR
Flags=0x00000003
TimeOut=90
=====End Readers=====
```