MP1B: MQSeries for OS/390 V5.2

Interpreting accounting and statistics data Version 1.3 MQSeries for OS/390 V5.2

Colin Paice

August 2002

Document Number MP1B

Take Note!
Before using this User's Guide and the product it supports, be sure to read the general information under "Notices".

Third Edition, August 2002

This edition applies to Version 1.3 of "MQSeries for OS/390 V5.2 - Interpreting accounting and statistics data" and to all subsequent releases and modifications until otherwise indicated in new editions.

A form for reader's comments is provided at the back of this publication. If the form has been removed, address your comments to:

IBM United Kingdom Laboratories

AIM WW Technical Sales (MP102)

Hursley Park

Hursley

Hampshire, SO21 2JN, England

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you. You may continue to use the information that you supply.

Contents

Notices

This report is intended to give guidance on the use and interpretation of the statistics and accounting in MQSeries for OS/390 Version 5.2. The information in this report is not intended as the specification of any programming interfaces that are provided by OS/390 or MQSeries.

References in this report to IBM products or programs do not imply that IBM intends to make these available in all countries in which IBM operates.

Information contained in this report has not been submitted to any formal IBM test and is distributed "as is". The use of this information, and the implementation of any of the techniques, is the responsibility of the customer, and depends on the customer's ability to evaluate and integrate them into their operational environment.

The following terms, used in this document, are trademarks of the IBM Corporation in the United States or other countries or both:

CICS

DFSORT

IMS/ESA

MQSeries

MVS/ESA

OS/390

Performance Reporter

The following term, used in this document, are trademarks of the SAS Corporation in the United States or other countries or both:

SAS

The following term, used in this document, are trademarks of the Lotus Corporation in the United States or other countries or both:

Lotus 123

Summary of Amendments

Date	Changes
November 2000	Initial release
June 2001	Inclusion of SAS definitions, minor textual updates, and update to Sample C program to resolve problem when processing buffer pool statistics.
August 2002	Additional explanations of how to interpret statistics and other editorial changes. SAS definitions updated, and the programs to print out the statistics have been improved, mainly to display times in hh:mm:ss format rather than internal STCK format. Document converted to WordPro.

This page intentionally left blank.

Introduction

MQSeries accounting and statistics

MQSeries for OS/390 provides statistics information about processing within the queue manager, and provides accounting information about individual application and channel usage. Both statistics and accounting information are written to the OS/390 SMF facility. For information about SMF see the *MVS System Management Facilities (SMF)* manual.

MQSeries statistics These are produced periodically, typically every half an hour, or every hour. Information is provided by the different resource managers (components) within the queue manager, and allows you to identify potential problems in the setup and usage of your queue manager. For example the buffer pool statistics can tell you that you need to increase the size of a buffer pool.

You should keep the data for several months and look for trends to see if changing usage patterns will cause problems. For information about processing and interpreting MQSeries statistics see Understanding and using MQSeries statistics data on page 35.

MQSeries accounting MQSeries produces accounting information about the activities and resources used by applications and channels. These can be used to analyze application activity and to charge users for their MQSeries usage. For information about processing and interpreting MQSeries accounting information see Understanding and using the new accounting data on page 23.

Summary of changes in Version 5.2

In Version 5.2 of MQSeries for OS/390 the accounting information has been significantly enhanced to provide information about the queues used by an application and the resources used when processing MQSeries requests. The accounting information available in previous releases is also still available.

There are two new sections in the statistics to reflect the shared queue usage of DB2 and the Coupling Facility.

Contents of this report

This document is complimentary to the *MQSeries for OS/390 System Setup Guide* and provides examples and additional information on how to use and interpret MQSeries accounting and statistics information. Some of the information in the System Setup guide is repeated in this document so as to have all the relevant information in one place.

If you find this SupportPac useful, have suggestions on improving it, or spot any errors please contact the author, PAICE@UK.IBM.COM.

Required fixes

The following APARS(PTF) fix various small problems.

- PQ43750(UQ50781)
- PQ56039(UQ61462)
- PQ56178(UQ62949)

Additional materials with this SupportPac

Included in the SupportPac are programs (including one written in C) and JCL which can be used to display the data. Sample SMF data is also included to allow these programs to be run without having to first collect real data.

These additional files are contained in *mp1b.zip* and are named as follows:

mp1b.loa	Load library
mp1b.src	Source library
mp1b.smf	Sample SMF data
mp1b.sas	SAS definitions for layout of SMF records

The files need to be transferred to the destination TSO system as sequential binary files with a record format of FB 80. Use one of the following methods to accomplish this:

1. Use the SEND commands below to send the files to TSO as sequential binary files:
   - **send mp1b.loa A:mp1b.loadseq**
   - **send mp1b.src A:mp1b.srcseq**
   - **send mp1b.smf A:mp1b.smfseq**
   - **send mp1b.sas A:mp1b.sasseq**

   where **A** is the TSO session ID.

2. To send them via ftp ensure the BINARY option is set then use the following commands:
   - **site fixrecfm 80** (Optional)
   - **put mp1b.loa mp1b.loadseq**
   - **put mp1b.src mp1b.srcseq**
   - **put mp1b.smf mp1b.smfseq**
   - **put mp1b.sas mp1b.sasseq**

3. With Personal Communications, use the "Send Files to Host" option under the Transfer menu item to transmit to TSO
   - PC File **mp1b.loa** etc
   - Host File **mp1b.loadseq** etc

- `Transfer Type` **`pds`**
- The Transfer type of **pds** may need to be correctly setup. To do this, use the "Setup.Define Transfer Types" option under the Transfer menu item and create the **pds** type with the ASCII, CRLF and Append checkboxes all **un**selected, the **Fixed** radio button selected and the **LRECL** set to **80**

On TSO, issue the following commands to unload these sequential files into TSO partitioned datasets:

- **receive indsname(mp1b.loadseq)**
  when prompted for a filename, reply **`dsn(mqload)`**
- **receive indsname(mp1b.srcseq)**
  when prompted for a filename, reply **`dsn(mqsource)`**
- **receive indsname(mp1b.smfseq)**
  when prompted for a filename, reply **`dsn(mqsmf)`**
- **receive indsname(mp1b.sasseq)**
  when prompted for a filename, reply **`dsn(mqsas)`**

Contents of MQLOAD

From MQLOAD you will get a load library with the following attributes: record format undefined, record length 0, and block size 6144. This dataset has the following members:

MQ1150	This prints out MQSeries statistics, see Supplied programs to print out the SMF records on page 83.
MQ116S	This prints out the new task and queue accounting records, see Supplied programs to print out the SMF records on page 83.
MQ1160	This prints out the accounting information which was also available in earlier releases, see Supplied programs to print out the SMF records on page 83
MQCSMF	The load module from "Sample C program for displaying statistics and accounting" on page 75.

Contents of MQSOURCE

From MQSOURCE you will get a PDS with the following attributes: record format FB, record length 80, block size 800. This dataset has some C structures and the following members:

MQCSAMP	This prints out MQSeries statistics, see Supplied programs to print out the SMF records on page 83.
RUNCSAMP	This runs the C MQCSAMP program which prints out the accounting and statistics information.
COPYSMF	This extracts the data from SMF into a temporary file and invokes MQ116S to process the statistics.

Contents of MQSMF

This dataset has some SMF data collected after a batch job put some messages to a batch server which sent the replies back to the originator. This file is provided so you can run the programs with this SupportPac without having to collect any data yourself.

From MQSMF you will get sequential file with the following attributes: record format VBS, record length 32767, block size 27998.

Contents of MQSAS

This dataset has some SAS definitions for the layout of the MQSeries accounting and statistics records. There is a $$README in the PDS which describes the contents.

Overview of Performance Reporter

This document gives examples of processing MQSeries accounting and statistics data using the IBM product Performance Reporter. This chapter describes the facilities available with Performance Reporter so that you can understand the examples, if you are not familiar with Performance Reporter.

Performance Reporter is an IBM program product which is now part of Tivoli Decision support. In the past it has also called EPDM (Enterprise Performance Data Manager) which replaced SLR (Service Level reporter). The IBM product number for Performance Reporter is 5695-101, and Tivoli Decision Support is 5698-TD9. Some Performance Reporter publications are listed in the bibliography.

Performance Reporter runs on OS/390 and takes data from various sources (including SMF), manipulates the data and stores the results in DB2 tables. The facilities of DB2 and QMF can then be used to display reports and charts from the data. The Performance Reporter language definitions are similar to DB2 command language.

Input data definition

Although Performance Reporter can process data from many sources, this chapter describes how SMF data is processed.

The source of the data is defined using LOG definitions, and the layout of the data is described using RECORD statements.

The Performance Reporter definitions are available via APAR PQ46511.

DB2 definitions

The input data is processed and put into DB2 tables. You need to decide what data you want to keep, and how you want to summarize it. This will in turn define the columns you want in your DB2 table. For example, when looking at trends it is better to combine accounting records into one DB2 table row than to have one DB2 row for every accounting record. You could have the data summarized by day, week, month (or even all of them), with one table for each date type. If you want to be able to report by queue manager, you will need to have a column for queue manager. If you are not currently interested in which queue manager the records came from, you can omit this column from your table, or you might choose to keep this column in case you want to examine usage by queue manager at a later date.

Processing the records

Performance Reporter has UPDATE definitions that describe how the input data updates the DB2 tables. You can have multiple UPDATE definitions using the same record if you want to update multiple tables with different information from the input record (for example, summarizing the data by week, and by month).

Consider the scenario where only the total number of commits per queue manager per day are wanted. The DB2 table is called MQSTATS_TABLE and has following columns:

DATE	The date the SMF record was produced
QMGR	The queue manager name
TOTALCOMMITS	The total number of commits issued by all applications

The following fields in the SMF_116_01 update definition are of interest:

SM116DTE	The date the SMF records were produced
SM116SSI	The MQSeries subsystem ID
WTASCMN	The number of commits issued by the application

Example Performance Reporter definitions

Some examples of Performance Reporter definitions that would update the DB2 table are as follows:

```
DEFINE UPDATE UPDATEMQ
 FROM SMF_116_01
 TO MQSTATS_TABLE
 GROUP BY
 (DATE = SM116DTE,
  QMGR = SM115SSI)
SET (TOTALCOMMITS =SUM(WTASCMN))
```

Where:

DEFINE UPDATE UPDATEMQ	This defines UPDATEMQ to Performance Reporter as definitions that update the DB2 tables.
FROM SMF_116_01	This is the RECORD definition that defines the layout of the SMF 116 records.
TO MQSTATS_TABLE	This is the DB2 table to be updated.
GROUP BY	Identifies how the data is summarized.
DATE = SM116DTE	Sets the DB2 column DATE to the value in SM116DTE. This will also do conversion from one data type to another if required, for example from 00yydddF (the format of SMF116DTE) to DB2 internal date format.
QMGR = SM115SSI	Sets the DB2 column QMGR to the value in SM116SSI.
SET (TOTALCOMMITS =SUM(WTASCMN))	This defines how the data is summarized.

How the records are processed

The processing is as follows:

1. The first record is read from the input.
2. The fields are extracted using the definitions in the RECORD definition.
3. The table MQSTAT_TABLE is read with key date=SM116DTE and QMGR=SM116SSI.
   If the record does not exist, a record is inserted into the table with date=SM116DTE, QMGR=SM116SSI and TOTALCOMMITS=WTASCMN.

   If the record does exist in the table, the value of WTASCMN is added to TOTALCOMMITS and the record rewritten to the table.
4. The next record is read from the input file.

Other processing that can be done includes the following:

column=MIN(inputfield)	The value of column is the minimum of the input field of all the records
column=MAX(inputfield)	The value of column is the maximum of the input field of all the records
hh=HOUR(SM116TME)	This extracts the hour value of a time
days=DAYS(date)	This gives the day number from January 1st 0001
date = DATE(indate)	Converts the input value (indate) to a date (an example is converting a day number obtained by DAYS back to a date)
Monday = DATE(((DAYS(date)/7)*7) + 1)	This gives the date of the Monday of that week (this makes it easy to report data on a weekly basis)
month = MONTH(date)	Extracts the month number from the date
time5 = TIME(ROUND(SM116TME,5 MINUTES))	This rounds down the date/time to the specified interval, in this case to the 5 minute boundary (this could be used to summarize accounting data to 5 minute periods over a day)
tran = substr(text,start,count)	This extracts a portion of a text string
string = part1||part2	This concatenates two strings
If.. then.. Else **answer = CASE** **WHEN A=1 then 'YES'** **WHEN A=2 then 'NO '** **ELSE '???'** **END**	This allow conditional processing. This could be used to extract different data depending on whether it is an IMS or a batch job.

Many of these are illustrated in the samples supplied with this SupportPac.

Sample query

Using the table above, you can use DB2 commands to extract and display the data. For example the following query displays the data in the table.

```
SELECT DATE,QMGR,TOTALCOMMITS from MQSTATS_TABLE
```

To summarize the total number of commits across all queue managers in a day you can use the following query.

```
SELECT DATE,SUM(TOTALCOMMITS)
   from MQSTATS_TABLE
   group by DATE
```

If you use QMF, you can design how you want the data laid out, including headings, and if you want totals and subtotals. You can also use QMF to display the data graphically using GDDM.

You can also use the ODBC inteface from products like Lotus 123 to extract and display the data in a spreadsheet.

Deleting old data

Performance Reporter provides the facility to remove unwanted data, so for example you can easily delete detailed accounting records that are older than a week.

Understanding and using MQSeries accounting data

Accounting information

MQSeries produces accounting information about the activities and resources used by applications and channels.

Who uses the data?

People with different roles might want different views of the data:

- Application architects might be interested in data about applications and queues.
- Systems programmers might be interested in the resources used, and response time of DASD.

People need different views of the data at different times, and so you might keep *all* data for only 24 hours, but keep only a summary of the data for long term analysis.

- You might want detailed information about the last 24 hours to be able to identify any out of line conditions, and display the data from individual accounting records to explain any unusual events.
- For the long term you might want to have the data summarized by week, so you can do trend analysis on the number of transactions, the amount of data processed, and the delay caused by writing to the log for example.

What can you do with the data?

The examples below show some ways in which the accounting information can be used.

- Charge users' departments for their MQSeries usage, by CPU and by bytes processed. This can be done using information about the application users and the remote destination by using the channel name and network address.
- Identify high use queues and perform trend analysis on throughput over time.
- Show those applications using **MQSET** on a queue. You can check that if an error occurs, these applications reset any attribute they might change. For example make sure they reset the trigger attribute if the application sets the queue to NOTRIGGER.
- Identify where the MQI calls are being delayed, for example waiting for log I/O or waiting for page set I/O. If the log I/O takes a long time, you might need to consider moving the log data sets to a volume that is used less heavily, or splitting the queue manager work into multiple queue managers.
- Show where queues have been set up incorrectly, for example a queue that is not indexed when all of the requests are to get with a specific message ID, or an indexed queue where only get next requests are used.
- Evaluate application changes to make sure that there is no unexpected increase in MQSeries usage. For example if an application puts additional persistent messages, the volume of data logged increases, and so larger or a greater number of logs might be needed.
- Determine why application response time is different between two days. For example after MQSeries startup, messages might have to be read from the page set rather than just accessed from a buffer.
- Correlate the MQSeries accounting information for a CICS transaction with CICS and DB2 accounting in order to understand the complete transaction picture.

"Some useful DB2 queries for processing accounting and statistics data" on page 69 has some example DB2 queries illustrating some of the above.

Accounting information available before Version 5.2

The following information was available in MQSeries releases before Version 5.2.

- Information to identify the task, but not channel names.
- The amount of CPU used on the application TCB.
- Number of **MQPUT** or **MQPUT1** requests for messages of length 0 through 99 bytes, 100 through 999 bytes, 1000 through 9999 bytes, and greater than or equal to 10000 bytes.

- Number of **MQGET** requests for where the message obtained is of length 0 through 99 bytes, 100 through 999 bytes, 1000 through 9999 bytes, and greater than or equal to 10000 bytes.

For more information see "Understanding and using the accounting data available in and before Version 5.2" on page 20.

The data is written to SMF when the application or channel ends.

What does it cost to collect accounting information?

The cost of collecting the "old" accounting records is about 2-3% CPU overhead.

The amount of data produced can be significant. An application that gets a message, puts a message to a different queue and ends, produces a 436 byte record. The space used by 160,000 of these transactions is about 100 cylinders of 3390 DASD.

Summary of the new accounting information available in Version 5.2

The new accounting information can be broken down into the following areas:

- Task identification. This now includes channel names in addition to other information, and allows you to correlate accounting records with CICS and DB2.
- CPU used per MQSeries call, by queue where appropriate.
- Reasons why calls were delayed, for example waiting for log I/O to complete.
- Other information, for example the time a message spent on a queue from the time it was put to the time it was got, and total number of bytes processed.

The data is written to SMF when the application or channel ends, or when OS/390 issues the SMF interval broadcast - typically every hour or half hour. This interval is defined by the INTVAL statement in the SMFPRMxx member of SYS1.PARMLIB, see OS/390 MVS Initialization and Tuning Reference.

What does it cost to collect accounting information?

The cost of collecting the "new" accounting records is between 5-10% CPU overhead.

The amount of data produced can be significant. An application that gets a message, puts a message to a different queue and ends produces 2260 byte records. The space used by 18,000 of these transactions is about 100 cylinders of 3390 DASD.

Understanding and using the accounting data available in and before Version 5.2

The accounting information described in this chapter is available in Version 5.2 and earlier releases.

The accounting data is in SMF type 116 records, subtype 0. For information about the SMF record layout, and how to locate the data in the records see Appendix A, "Overall layout of MQSeries SMF records" on page 89. The following tables show the format of the message manager accounting records that are available in Version 5.2 and earlier releases.

Table 1. Structure of the Common MQSeries SMF header record QWHS

Offsets		Type	Len	Name	Description
Dec	**Hex**				
0	***(0)***	***Structure***	***128***	***QWHS***	
0	(0)		6		Reserved.
6	(6)	Character	1	QWHSNSDA	Number of self defining sections in the SMF records. See **Table** on page 89
7	(7)		5		Reserved.
12	(0C)	Character	4	QWHSSSID	QWHSSSID
16	(10)		24		Reserved.
40	(28)	Character	8	QWHCAID	User ID associated with the OS/390 job.
48	(30)	Character	12	QWHCCV	Thread cross reference (see "Thread cross reference data" on page 21)
60	(3C)	Character	8	QWHCCN	Connection name.
68	(44)		8		Reserved.
76	(4C)	Character	8	QWHCOPID	User ID associated with the transaction.
84	(54)	Signed	4	QWHCATYP	Type of connecting system (1=CICS, 2=Batch or TSO, 3=IMS control region, 4=IMS MPP or BMP, 5=Command server, 6=Channel initiator, 7=RRS Batch).
88	(58)	Character	22	QWHCTOKN	Accounting token set to the OS/390 accounting information for the user.
110	(6E)	Character	16	QWHCNID	Network identifier
126	(7E)		2		Reserved.

Table 2. Structure of the message manager accounting record QMAC

Offsets		Type	Len	Name	Description
Dec	**Hex**				
0	***(0)***	***Structure***	***48***	***QMAC***	***Message manager accounting data***
0	(0)	Bitstring	2	QMACID	Control block identifier.
2	(2)	Unsigned	2	QMACLL	Control block length.
4	(4)	Character	4	QMACEYEC	Control block eye catcher (QMAC).
8	(8)	Character	8	QMACCPUT	CPU time used (TOD format).
16	(10)	Signed	4	QMACPUTA	Number of **MQPUT** requests for messages of length 0 through 99 bytes.
20	(14)	Signed	4	QMACPUTB	Number of **MQPUT** requests for messages of length 100 through 999 bytes.
24	(18)	Signed	4	QMACPUTC	Number of **MQPUT** requests for messages of length 1000 through 9999 bytes.

28	(1C)	Signed	4	QMACPUTD	Number of **MQPUT** requests for messages of length greater than or equal to 10000 bytes.
32	(20)	Signed	4	QMACGETA	Number of **MQGET** requests for messages of length 0 through 99 bytes.
36	(24)	Signed	4	QMACGETB	Number of **MQGET** requests for messages of length 100 through 999 bytes.
40	(28)	Signed	4	QMACGETC	Number of **MQGET** requests for messages of length 1000 through 9999 bytes.
44	(2C)	Signed	4	QMACGETD	Number of **MQGET** requests for messages of length greater than or equal to 10000 bytes.

Interpretation

The QWHC* fields gives you information about the user (for example, the user ID (QWHCAID) and the type of application (QWHCATYP)).

The QMAC* fields gives you information about the CPU time spent processing MQI calls, and counts of the number of **MQPUT** and **MQGET** requests for messages of different sizes.

Records containing zero CPU time

Records are sometimes produced that contain zero CPU time in the QMACCPUT field. These records occur when long running TCBs identified to MQSeries either terminate or are prompted to output accounting records by accounting trace being stopped. Such TCBs exist in the CICS adapter and in the channel initiator (for distributed queuing without CICS). The number of these TCBs with zero CPU time depends upon how much activity there has been in the system:

- For the CICS adapter, this can result in up to nine records with zero CPU time.
- For the channel initiator, the number of records with zero CPU time can be up to the sum of `Adapters + Dispatchers + 6`, as defined in the channel initiator parameters.

Thread cross reference data

The interpretation of the data in the thread cross reference (QWHCCV) field varies. This depends on what the data relates to:

- CICS (QWHCATYP=1) - see **Table 3** on page 21
- IMS (QWHCATYP=3 or 4) - see **Table 4** on page 22
- Batch, TSO, or RRS Batch (QWHCATYP=2 or 7) - this field consists of binary zeros
- Others - no meaningful data

Table 3. Structure of the thread cross reference record for a CICS system

Offsets		Type	Len	Name	Description
Dec	**Hex**				
48	(30)	Character	4	QWHCTNO	CICS thread number.
52	(34)	Character	4	QWHCTRN	CICS transaction name.
56	(38)	Packed Decimal	4	QWHCTASK	CICS task number.

Accounting records for the CICS adapter TCBs have a blank thread cross reference. You can identify which records apply to the adapter because the QHWCTRN field is blank, as records for CICS transactions have the transaction code in this field.

Table 4. Structure of the thread cross reference record for an IMS system

Offsets		Type	Len	Name	Description
Dec	Hex				
48	(30)	Character	4	QWHCPST	IMS partition specification table (PST) region identifier.
52	(34)	Character	8	QWHCPSB	IMS program specification block (PSB) name.

Special considerations when using IMS accounting records

A single IMS application might write two SMF records. In this case, the figures from both records should be added to provide the correct totals for the IMS application.

Understanding and using the new accounting data

The new accounting data is in SMF type 116 records, subtypes 1 and 2. For information about the SMF record layout, and how to locate the data in the records see Appendix A, "Overall layout of MQSeries SMF records" on page 89.

When is the queue related accounting data produced?

An SMF record is produced when the accounting trace class (3) has been activated (for example "+cpf START TRACE(A) CLASS(3)") and either:

- The job or application ends.
- The SMF statistics broadcast occurs, and it is a long running application, such as a channel. This means it was running prior to the last SMF statistics broadcast. Before Version 5.2 the records were produced only when the job or application ended. So for a channel that was active for a year, you would get one record. In Version 5.2 you can request that records are produced at the SMF broadcast, typically every 30 minutes, by setting STATIME=0 in the CSQ6SYSP system parameter macro.

What is in the accounting record?

In the accounting record there are three sections covering the new data: task identification, task accounting, and queue related.

Task identification

This information is in a structure called the WTID. The detailed description and layout of the fields is given in "Task identification (WTID)" on page 98 and includes the following:

- Job name
- User ID
- Transaction name, if applicable
- Channel name, if applicable, including the TCP/IP address or APPC LU

Task accounting data

This information is in a structure called the WTAS. It includes information about commit and backout verbs, and other information that is not specific to a particular queue. The detailed description and layout of the fields is given in "Task related information (WTAS)" on page 96 and includes the following:

- Number, accumulated elapsed time, and accumulated CPU time per verb for commit and backout requests.
- How many times a request was made to ensure data has been written to the log, and the accumulated time waiting for the write to the logs to complete for commit and backout requests.

Queue related accounting data

The information is in a structure called the WQ. The detailed description and layout of the fields is given in "Queue records (WQ)" on page 93.

- Queue type, for example model queue or local queue.
- Queue name as used by the application, and the base queue name. These might be different, for example an alias queue maps to a base queue name.
- Number, accumulated elapsed time, and accumulated CPU time for **MQOPEN**, **MQCLOSE**, **MQPUT**, **MQPUT1**, **MQGET**, **MQINQ**, and **MQSET**.

- Number of successful **MQGET**, **MQPUT** and **MQPUT1** calls that successfully processed a message. For example an **MQGET** that returned 'no message found' is considered an unsuccessful get.
- How many times a request was made to ensure that data has been written to the log, and the accumulated time waiting for the write to the logs to complete. (**MQPUT**, **MQPUT1**, **MQGET**, and **MQSET**.)
- How many page set reads, and the accumulated time doing page set reads, and page set number. (**MQGET**, **MQPUT**, and **MQPUT1**.)
- Type of **MQGET** request, get by message ID or correl ID, or get first; destructive get or get browse. (**MQGET**.)
- Total number of bytes put or got, maximum and minimum message size. (**MQPUT** and **MQGET**.)
- Time on queue (TOQ). The time between the message arriving on the queue, and the get of the message. Total for all messages, the maximum and the minimum time for messages processed on the queue.
- Number of generated messages, such as trigger or event messages.

How to process the data

If you want to examine a few accounting records you can use programs described in "Supplied programs to print out the SMF records" on on page 83 to print out the contents of the records. With a large number of records the amount of output quickly becomes unmanageable.

If you want to write your own procedures to process the data you should read "Understanding the fields in the records" on page 34 for guidance on how to interpret the fields.

You could write a program that reads the SMF 116 records and inserts records into a DB2 table where the fields in the DB2 table match the field names in the SMF records. If you want one row in the table per SMF 116 record, insert the row into the database. If you want to accumulate the data, so that you have one row per transaction for a particular day, you need to read the record, update the fields, and rewrite the record.

The IBM product Performance Reporter does all of this for you, and you can easily migrate the SMF data into DB2, and have the power of DB2 and QMF to query the data. See "Overview of Performance Reporter" on page 13 for a summary of the facilities available with Performance Reporter.

Alternatively, instead of using a database you could write an application that summarized the data and displayed information when the end of the input file has been reached. This is not as flexible as storing the data in a DB2 database.

Other analysis tools, in addition to Performance Reporter include SAS from the SAS corporation.

The way you store the data depends on the analysis you want to perform. For example:

- If you are displaying trends over a long period of time, you might want to summarize the data so there is one row of data per application per week.
- If you want the raw, un-summarized data, you might want to have one row per accounting record, and include all of the fields. You might want one table for the task information, and another table for the individual queue records. You can use the fields in the WTID record to link the tables together.
- If you are displaying the data on an hourly basis, you might want to combine the records, but extract all of the fields.

We recommend that you summarize the information on a daily basis, and keep the last 28 days worth of data.

For these different analyses the data and database structure will be very different.

Typical long term analysis

When you are looking at data over the long term you are looking for trends rather than individual accounting records. The following section gives an example of the sort of data you can use for trend analysis.

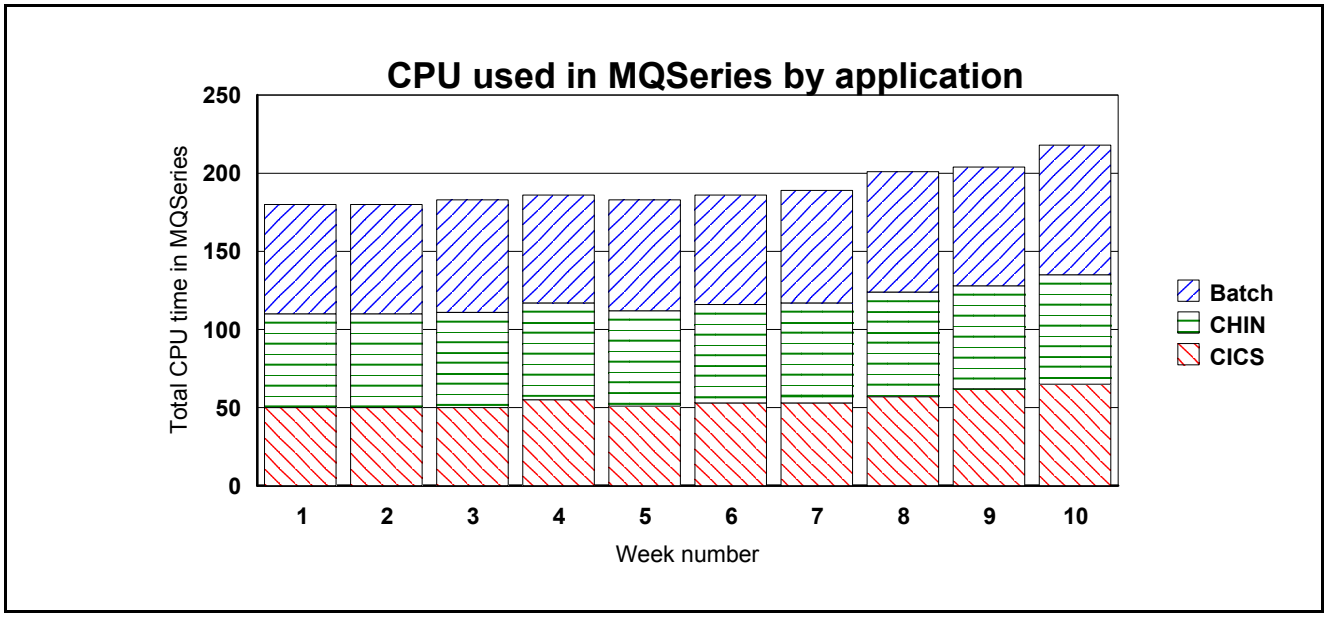
Table 5. Example columns in a DB2 table for long term analysis

This DB2 table has one row per transaction or channel and each row has the data for a whole weeks worth of data. So if your MQSeries system had only one batch job that puts messages to a queue, and a channel that sends them to a remote site there would be two records for each week.	
Date	Date of start of week (Monday)
MVS	OS/390 subsystem name
QMGR	Queue manager name
Jobtype	This is a character representation of the job type, for example 'CICS'
Jobname	This could be the batch job name, or CICS region
Tran	The CICS or IMS transaction
Channel	The channel name
Channelq	The channel connection name - the TCP/IP address or LU name
NCommits	Number of commits of this task, transaction, channel combination
ET	Total elapsed time
CT	Total CPU time used
Logtime	Time spent waiting for logging
PSTime	Time waiting for page set activity
DB2time	Time waiting for DB2 requests to be processed
CFtime	Time waiting for CF requests to be processed
Bytes	Bytes processed

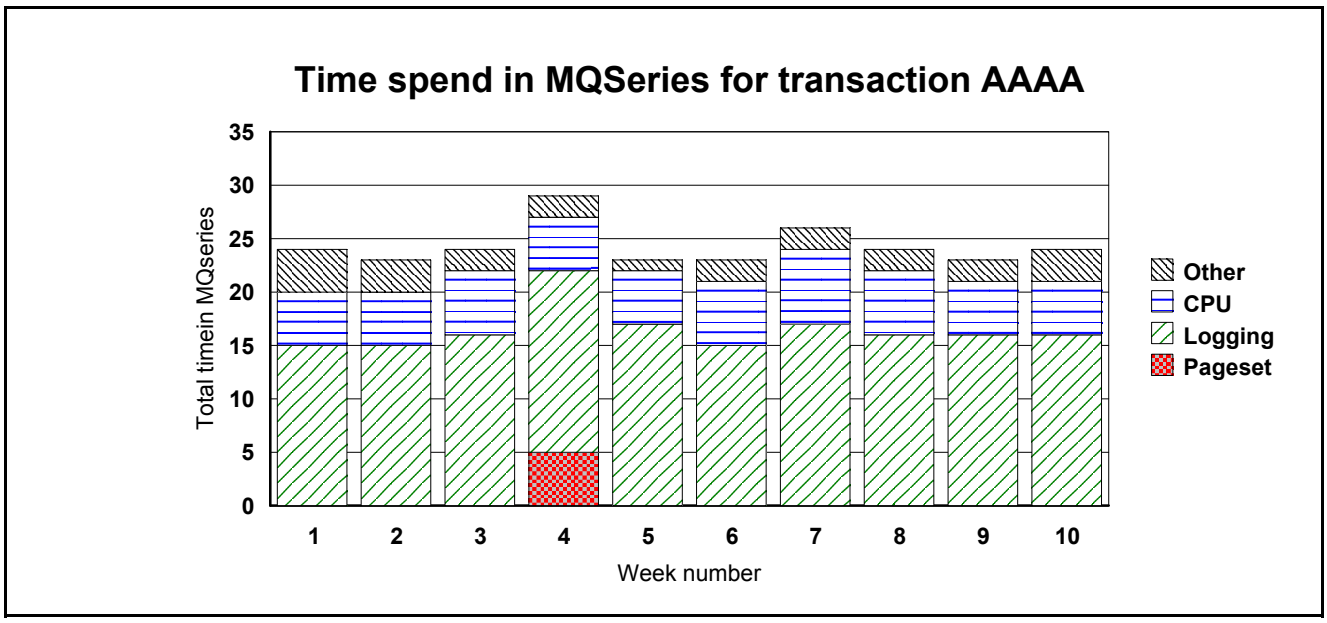
Some charts you might use for analysis might include

- Total CPU used by task per week, by week. For illustration see**1** page 26
- Total elapsed time in MQSeries per task per week, by week.
  If this is plotted in a bar chart, the height of the bar gives the total elapsed time for all transactions, and with in the bar, it shows how much was CPU and logging for example. For illustration see **2**page 26.
- Bytes processed per task, per week, by week.
- CPU used by the all tasks per week, by week.

1Growth in CPU over time by major application



2Breakdown of where time in MQSeries is spent for a transaction



Example Performance Reporter update definition for queue records

3The figure below shows the Performance Reporter definitions to update the DB2 table **DRL.MQ_Weekly_stats** using the field names of the SMF records in Appendix A, "Overall layout of MQSeries SMF records" on page 89. For information on Performance Reporter definitions see "Overview of Performance Reporter" on page 13

3Performance Reporter update definition for queue records

```
DEFINE UPDATE MQ_weekly_stats              - As known by Performance Reporter (1)
  FROM SMF_116_01 section QUEUE_BLOCK      - Which part of the SMF record (2)
  TO   DRL.MQ_Weekly_stats                 - DB2 table name              (3) GROUP BY
                           - These are the key to the records (4)
   (DATE             = DATE(DAYS(SM116DTE - 7 DAYS)/7*7 + 1),
                                           - Date of the start of the week (5)
    MVS              = SM116SID,           - OS/390 subsystem name (6)
    QMGR             = SM116SSI,           - Queue manager name (7)
    JOBNAME          = WTIDCCN,            - Region name        ( 8)
    TRAN             = WTIDCTRN,           - Transaction name (if any) (9)
    Channel          = WTIDCHL,            - Channel name (10)
    Channelq         = WTIDCHLC)           - Channel qualifier (10)
  SET                                                          (11)
   (
    CT = SUM(OPENCT/4096+                  - Total CPU used from queue records (12)
             CLOSECT/4096+
             GETCT/4096+
             PUTCT/4096+
             PUT1CT/4096+
             INQCT/4096+
             SETCT/4096),
    ET = SUM(OPENET/4096+                  - Total elapsed time-queue records (13)
             CLOSEET/4096+
             GETET/4096+
             PUTET/4096+
             PUT1ET/4096+
             INQET/4096+
             SETET/4096),
   Logtime =SUM(GETJWET/4096+              - Total time writing to the log (14)
            PUTJWET/4096+
            PUT1JWET/4096+
            SETJWET/4096),
   PStime  =SUM(GETPSET/4096+              - Total time processing a page set (15)
            PUTPSET/4096+
            PUT1PSET/4096),
   Bytes  = sum(Putbytes+getbytes)         - Total bytes processed (16)
   );
```

The numbers in brackets are explained below:

1. The `DEFINE UPDATE` statement defines how data will be taken from the SMF record and applied to the DB2 table. Each update definition has a name, this one is named `MQ_weekly_stats`.
2. The `SMF116_01` is the definition of the SMF 116 record layout. It has the same layout as the structures in Appendix B, "Detail layout of MQSeries accounting and statistics records" on page 93 `Section QUEUE_BLOCK` says to repeat this update for every queue record in the SMF record.

`DRL.MQ_Weekly_stats` is the name of the DB2 table. This table has columns that match **Table 5** on page 25.

The fields in `GROUP BY` define the unique key to the record, so all accounting records with the same combination of values in `GROUP BY` map to the same DB2 row.

1. `DATE(DAYS(SM116DTE - 7 DAYS)/7*7 + 1)` takes the date the record was created (SM116DTE), and calculates the date of the first day of the week, so all records for that week map to the same value.
2. The OS/390 subsystem name from the SMF record header.

3. The queue manager subsystem name from the SMF record header.
4. The jobname of the application, for example the CHINIT or the CICS region.
5. The transaction name, taken from the task identification block, see "Task identification (WTID)" on page 98.
6. The channel name taken from the WTIDCHL field. The WTIDCHLC field is the TCP/IP address or LU name.
7. The SET command specifies how the data is accumulated from the SMF record.
8. This calculation does the following SMF record into the DB2 record:
    1. Convert the times from STCK format into microseconds (by dividing by 4096).
    2. Accumulate the CPU used for the different verbs, and calculate the SUM of all the records processed.
    3. Save the accumulated value in CT.
9. Performs a similar calculation on the elapsed time.
10. Accumulates the time spent waiting for log I/O to complete.
11. Accumulates the time spent waiting for page set I/O to complete.
12. Accumulates the number of bytes put and got.

Example Performance Reporter update definition for task accounting

4Performance Reporter update definition for task records

```
DEFINE UPDATE MQ_weekly_stats_T            - As known by Performance Reporter
  FROM SMF_116_01                          - Use the task section by default
  TO   DRL.MQ_weekly_stats                 - DB2 table name
  GROUP BY                                 - These are the key to the records
   (DATE             = DATE(DAYS(SM116DTE - 7 DAYS)/7*7 + 1),
                                           - Date of the start of the week
    MVS              = SM116SID,           - MVS subsystem name
    QMGR             = SM116SSI,           - Queue manager name
    JOBNAME          = WTIDCCN,            - Region or channel name
    TRAN             = WTIDTRN,            - Transaction name (if any)
    Channel          = WTIDCHL,            - Channel name
    Channelq         = WTIDCHLC)           - Channel qualifier
  SET
   (
    CT = SUM(WTASCMCT/4096+                - Total CPU used
             WTASBACT/4096+
             WTASOTCT/4096),
    ET = SUM(WTASCMET/4096+                - Total Elapsed time
             WTASBAET/4096+
             WTASOTET/4096),
   Logtime =SUM(WTASJWET/4096),            - Total time writing to the log
   PStime  =SUM(WTASPSE0/4096),            - Total time processing a page set
   NCOMMITS = SUM(WTASCMN)                 - The number of records accumulated
   );
```

See **3**the previous page for a description of what the definition means.

The updates are made from the task accounting of the SMF record instead of the queue sections.

Typical daily analysis

When looking at the data over a short term, you are usually looking for out of line conditions, or exceptions to normal processing. In these cases you normally want individual records, or data summarized over a short interval, perhaps 5 minutes. You are likely to want to keep the detailed information about individual queues.

One way of holding this data is to have two tables, one similar to the table above, and a table containing the individual queue records for a task.

Update definition for the daily statistics for task information

Update definition for the daily statistics for task information

```
DEFINE UPDATE MQ_Daily_Task           - As known by Performance Reporter
  FROM SMF_116_01                     - Use the task section by default
  TO   DRL.MQ_Daily_Task              - DB2 table name
  GROUP BY                            - These are the key to the records
   (DATE             = SM116DTE,      - Date the record produced
    TIME             = SM116TME,      - Time the record was produced
    MVS              = SM116SID,      - OS/390 subsystem name
    QMGR             = SM116SSI,      - Queue manager name
    JOBNAME          = WTIDCCN,       - Region or channel name
    TRAN             = WTIDCTRN,      - Transaction name (if any)
    Channel          = WTIDCHL,       - Channel name
    Channelq         = WTIDCHLC)      - Channel qualifier
  SET(
  ...
     );
```

The definitions are the same as above except the date and time are different. There is one row in this DB2 table for every SMF 116 accounting record.

The SET part of the update definitions would be the same as above.

A typical query that could be issued against the above DB2 table is shown in below.

5Identify transactions whose average time in MQSeries is outside a value

```
Select Date,mvs,qmgr,jobname,tran,et/NCommits from
       DRL.MQ_Daily_Task
       Where tran='TRA1'              - Select specific CICS transaction
       and
       et/NCommits > 10000            - The average response time is
                                      - greater than 10000 microseconds
```

In this example the CICS transaction gets a message, puts a reply, commits the request and ends. In this case `et/Ncommits` is a valid measure. For long running jobs, long running transactions, or channels, `et/NCommits` is not valid.

Update definition for queue records for daily statistics

Update definition for queue records for daily statistics

```
DEFINE UPDATE MQ_Queue_stats          - As known to Performance Reporter
  FROM SMF_116_01 Section QUEUE_block - The queue section of the 116
  TO   DRL.MQ_Daily_Queue             - This DB2 table
   GROUP BY                           - These are the fields that
                                      - uniquely identify a record
   (DATE             = SM116DTE,      - Date record produced
    TIME             = SM116TME,      - Time record produced
    MVS_SYSTEM_ID    = SM116SID,      - OS/390 subsystem name
    MQ_SUBSYSTEM     = SM116SSI,      - Queue manager name
    JOBNAME          = WTIDCCN,       - Region or channel name
    TRANSACTION      = WTIDCTRN       - Transaction name (if any)
    Channel          = WTIDCHL,       - Channel name
    Channelq         = WTIDCHLC,      - Channel qualifier
    Queue            = Basename)      - Base queue name
    SET
     (
     openET   = SUM( openET/4096),    - Convert open elapsed time in stck to microseconds
     openCT   = SUM( openCT/4096),    - Convert open CPU time in stck to microseconds
     openN    = SUM( openN   ),       - Count of open
     closeET  = SUM( closeET /4096),  - Convert close elapsed time in stck to microseconds
     closeCT  = SUM( closeCT /4096),  - Convert close CPU time in stck to microseconds
     closeN   = SUM( closeN  ),       - Count of close
     getET    = SUM( getET    /4096), - Convert get elapsed time in stck to microseconds
     getCT    = SUM( getCT    /4096), - Convert get CPU time in stck to microseconds
     GetN     = SUM( GetN    ),
     GetBrwA  = SUM( GetBrwA  ),
     GetBrwS  = SUM( GetBrwS  ),
  ...
 );
```

This update updates rows in the DB2 table **DRL.MQ_Queue_stats**. There is one row in this DB2 table for queue for every task in the SMF 116 accounting record.

As the two tables **DRL.MQ_Daily_Task** and **DRL.MQ_Daily_Queue** have the similar key fields, these key fields can be used to join the two tables and produce detailed reports.

The partial query below shows how a DB2 can be used to process the data to produce reports, similar to the output on page 31:

Part of a DB2 query to extract data from the queue table

```
SELECT TRANSACTION,QUEUE,'OPEN  ',OPENN,OPENET/OPENN,OPENCT/OPENN
  FROM DRL.MQ_QUEUE_STATS
  WHERE OPENN > 0
UNION
SELECT TRANSACTION,QUEUE,'CLOSE ',CLOSEN,CLOSEET/CLOSEN,CLOSECT/CLOSEN
  FROM DRL.MQ_QUEUE_STATS
  WHERE CLOSEN > 0
UNION
 ....
```

Example report showing a transaction and queues used by it

TRAN	QUEUE	VERB	COUNT	AVG ET	AVG CT
A8EA	Q_INPUT	CLOSE	3	64	63
A8EA	Q_INPUT	GET	3	8251	237
A8EA	Q_INPUT	OPEN	3	250	181
A8EA	Q_OUTPUT	PUT1	3	9651	621

How to interpret the data

The way you analyze the data depends on the application. For example consider the following four scenarios:

- A simple CICS transaction that gets a message, puts a reply, commits and ends.
- A CICS transaction that drains a queue. This transaction loops doing {**MQGET** (with wait option), **MQPUT1**, commit} and ends when no further messages have arrived in a short period of time.
- A sender channel that gets messages from a transmission queue (and sends them to a remote queue manager).
- A receiver channel that receives messages from a remote queue manager and puts them to various queues.

The following sections show how the data could be interpreted.

A simple CICS transaction

This transaction starts, opens a queue for input, gets a message from the queue, puts a message using **MQPUT1** to the replyto queue, commits the request and ends.

In this case, each transaction will have one accounting record that shows one commit, and the record will have 2 queue subsections.

Some typical analysis and reports might include:

- The average of the total CPU used in MQI calls across all transactions during the day. Plot in a line graph this average and number of records against the day.
- The average of the total elapsed time of the MQSeries calls for a transaction, across all transaction during the day.
- A bar chart, showing the constituent parts of the elapsed time, CPU, log wait, page set I/O, and other suspend time.
- The total CPU time used for MQI calls by transaction, charged back to the user's department.
- The average number of bytes put and got per transaction per day, or the total divided by the number of commits per day.
- The number of transactions where the time spent in MQSeries was greater than a particular value per day.
- The maximum total time in MQSeries per transaction per day.

A CICS transaction that drains a queue

The CICS transaction is triggered, starts, opens the input queue, and loops, getting a message from the queue (with the wait option), putting a reply to the specified queue using **MQPUT1**, committing the work, and going round the loop again. When there are no more messages and the wait interval expires, the transaction terminates.

In this case, there might be multiple units of work within one transaction, and so there might be multiple commits in the accounting record. The elapsed time information is accumulated over many calls, so the average time per call can be obtained, but the maximum time for the calls cannot be obtained. If the transaction runs over half an hour, there might be multiple accounting records, which are produced on the SMF accounting broadcast.

Some typical analysis or reporting might include:

- A plot of total MQSeries CPU used across all transactions by day.
  Total CPU used divided by the number of commits give a measure of the transaction cost. This can be plotted by day to see if there is a trend to the resources used.
- A breakdown of the total elapsed time of the MQI calls across all transaction during the day.
- A bar chart showing the constituent parts of the elapsed time, CPU, log wait, page set I/O, and other suspend time (similar to that for the simple CICS transaction).

- A plot of total elapsed time/number of commits is approximately the same as for the simple CICS transaction.
  Because the program issues an **MQGET** with the wait option, there might be two **MQGET** calls for every message processed. The first **MQGET** finds no message, and so waits in the adapter. When a message arrives the **MQGET** is re-issued by the adapter. If there was a message to process, the adapter does not have to wait, and only one **MQGET** call is issued. So in this scenario there are more **MQGET** requests than in the simple CICS transaction.
- The total CPU time used for MQI calls, charged back to the user's department. This might be difficult to do because messages can come from many sources and be processed by this transaction. You might be able to charge back depending on the bytes put to specific application queues.
- The total number of bytes put and got per transaction per day, or the total divided by the number of commits per day.
- The number of transactions where the time spent in MQSeries was greater than a particular value per day.
- The "average response time per day" (`total of time in MQSeries calls/number of commits`) is a useful measure which gives the average amount of time in MQSeries each Unit of Work took. Strictly the "average response time per day" refers to the response time of the transaction.

A sender channel

A sender channel gets one or more messages from a transmission queue and sends them to a remote queue manager. There might be some processing to internal queues at the end of batch. Because there can be a variable number of messages per batch, the `cost/(number of commits)` does not give a very meaningful answer. The number of commits could be zero if only fast messages have been processed, so calculations such as `Elapsed time/Number of commits` are meaningless and could result in a divide by zero condition.

Some of the typical analysis and reports might include:

- Total CPU used by the channel per day
- The total bytes read from the transmission queue per day
- The total number of messages got per day
- The average achieved batch size per day (or per interval)

With data for multiple applications on a channel it might not be possible to charge back usage to departments because you cannot identify who gets charged for what.

A receiver channel

A receiver channel receives one or more messages from a remote queue manager and puts them to one or more queues. There might be some processing to internal queues at the end of a batch. The number of commits could be zero if only fast messages have been processed, so calculations such as `Elapsed time/Number of commits` are meaningless and could result in a divide by zero condition.

Some of the typical analysis and reporting might include:

- Total CPU used by the channel per day
- The total bytes put to each queue per day
- The total number of messages put per day
- The average achieved batch size per day (or per interval)

Understanding the fields in the records

MQI calls like **MQPUT** and **MQGET** act on a queue, where a commit call applies to the whole transaction and is not queue specific.

Constants used in the records

For fields like the queue type (QTYPE in the WQ) the values used are in the cmq* member in the SAMPLIB library, depending on the programming language; for example CMQC.H for C.

How to interpret durations

Information on durations is usually stored as a cumulative time, and the number of times that event happened. The figures used in this section are for illustration and do not reflect real figures.

Consider an application that issued 2 **MQSET** requests:

1. The first request, which took 10 ms of elapsed time, and used 1 ms of CPU.
2. The second request on the same queue, which took 10 ms elapsed time and 3 ms of CPU.

This would be reported as:

- SETN - the number of SEQ requests = 2.
- SETET - the accumulated elapsed time of the calls = 20 ms
- SETCT - the accumulated CPU time of the calls = 4ms.

This can be interpreted as follows:

- The average time of the **MQSET** calls was 10 milliseconds.
- The average CPU time used for the **MQSET** calls was 2 milliseconds.

The time values are in S/390 Store Clock format (STCK), which is a double word where bit 53 is a microsecond. To convert a STCK value to microseconds ignore the bottom 12 bits. In C this can be done by treating the values as long long and dividing by 4096 to get to microseconds.

Other programming considerations

- Most fields are initialized to zero.
- The get minimum message size, and the put minimum message size are set to a large value. You should only use these fields if the number of valid puts (VALIDPUT) or valid gets (VALIDGET) is non zero.
- When an MQSeries application ends, it posts an asynchronous task to create the SMF accounting record. The time in the SMF record (SM116TME) is the time the record was produced (the number of hundredths of a second since midnight). It is usually close (within a second) to the time the transaction or channel ended. If you need more accurate times, you should use the time interval end time (WTASINTE), which is in STCK format.
- Time on queue is calculated when the **MQGET** was successful and it was a destructive get.

Understanding and using MQSeries statistics data

Statistics information

The statistics data is in SMF type 115 records, subtypes 1 and 2. For information on the SMF record layout, and how to locate the data in the records see Appendix A, "Overall layout of MQSeries SMF records" on page 93.

MQSeries produces statistics providing information on the resource managers (components) of the queue manager.

Note: The layout of the SMF header changed in MQSeries Version 5.2 to conform to the standard SMF layout. The subtype field is now 2 bytes long at offset decimal 22 (x'16'), in previous releases it used to be only one byte long. This is field SM115STF and SM116STF.

Who uses the data?

Usually the systems programmer and people responsible for monitoring performance use the MQSeries statistics.

What can you do with the data?

Some data should be reviewed daily, and some is used in the long term to identify potential problems early, to allow preventive actions to be taken.

Exceptions should be reviewed daily as these indicate a problem with your setup, for example a Coupling Facility structure filling up.

Usually statistics are reviewed weekly or monthly and any trends examined.

How to collect statistics

You start the statistics trace using the "+cpf START TRACE(S)" command.

Statistics information available before Version 5.2

The following is a summary of the information available in releases before Version 5.2

- Message manager statistics - counts of MQSeries verbs used
- Data manager statistics - count of access to objects
- Buffer manager statistics - information about the size and usage of buffer pools
- Log manager statistics - information about data written to the log data sets

The data is written to SMF every STATIME minutes (where STATIME is a value you set using the CSQ6SYSP system parameter macro).

New statistics information in Version 5.2

There are new statistics sections for DB2 and the Coupling Facility (the components needed to support shared queues).

There are new statistics sections on storage manager and lock manager, but these are for IBM's use and are not described further.

What does it cost to collect statistics?

The CPU cost of collecting statistics is negligible.

The amount of data produced is typically a few thousand bytes every hour.

The data is written to SMF every STATIME minutes, or if STATIME is zero, when OS/390 issues the SMF interval broadcast - typically every half hour or hour.

Many components of the queue manager produce statistics.

The follow list summarizes what is available and where to get more information:

- The number, and elapsed time of MQI requests issuing DB2 calls, as used by shared queues (see "Statistics on MQSeries' use of DB2" on page 39).
- The number, type, and duration of Coupling Facility calls as used by shared queues (see "Coupling Facility statistics" on page 47.
- The number of requests for each MQSeries call used (see "Message manager statistics" on page 51).
- The number of different requests to access data within the queue manager (see "Data manager statistics" on page 52).
- For each buffer pool, information about the usage of the buffer pool (see "Buffer manager statistics" on page 53).
- Information about the log (see "Log manager statistics" on page 61).

Statistics on MQSeries' use of DB2

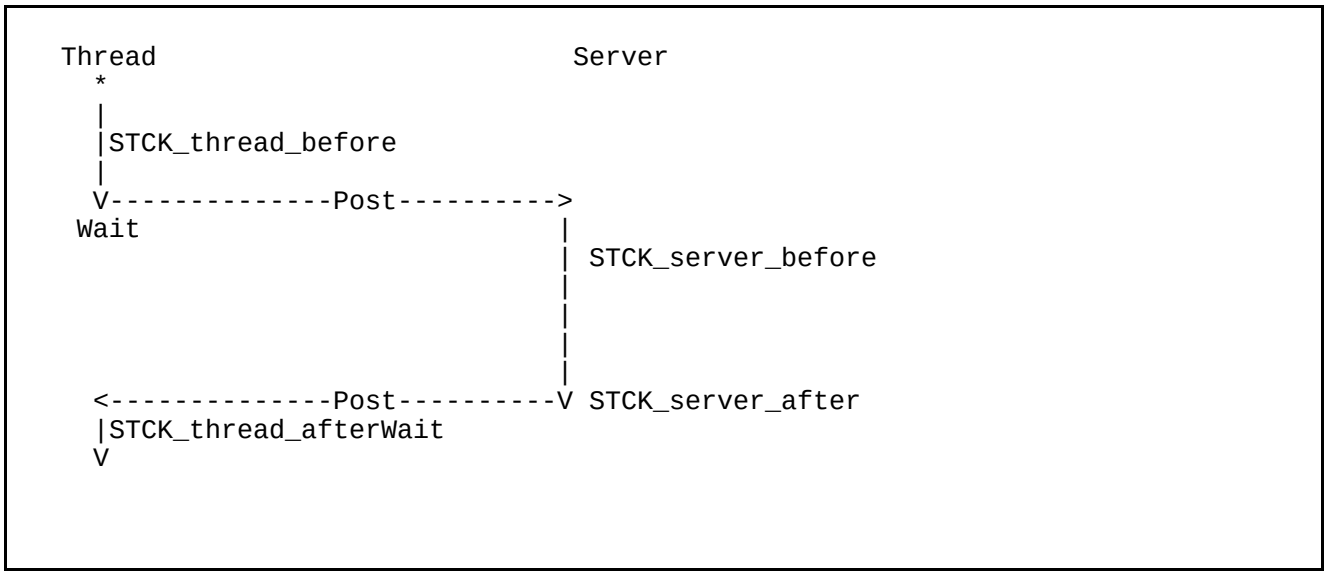
The DB2 manager manages the interface with the DB2 database that is used as the shared repository.

When using shared queues, object definitions and other information are stored in DB2 tables. This means you may be involved in tuning DB2.

DB2 requests are made from the queue manager by passing a request to a pool of server tasks that issue the DB2 request on behalf of the applications.

The figure below shows how DB2 requests are issued

Flow of a request for a DB2 service from a thread to server task



The processing for a thread wanting to issue a read request is as follows:

1. The thread puts a request onto a server work list.
2. The thread determines the current time (`STCK_thread_before`).
3. The thread posts a server task.
4. The thread waits.
5. The server task wakes up, and determines the current time (`STCK_server_before`).
6. The server takes the first request off the server work list and issues the DB2 request.
7. When the request has ended it posts the thread task.
8. The server task determines the current time (`STCK_server_after`) and updates the statistics:
    1. It increments the number of read requests `READCNT`.
    2. It calculates the time taken it took to process the request, `STCK_server_after-STCK_server_before` and adds this to the cumulative time `READSCUW`.
    3. If the time for the request was larger than the previous maximum if replaces the READSMXW with the delta.

    Note: For other request, other counters are updated. These are `LIST*`, `UPDT*`, `DELE*`, and `WRIT*`.
9. The original thread wakes up and determines the current time (`STCK_thread_after`) and updates the statistics:
    1. It calculates the time spent waiting (`STCK_thread_after - STCK_thread_before`) and adds this to the cumulative time `READTCUW`.

2. If the time spent waiting for the request was greater than the previous maximum if replaces the READTMXW with the larger value.

Note: For other request, other counters are updated. These are LIST*, UPDT*, DELE*, and WRIT*.

10. The thread continues processing.

The processing is similar for update, write, and delete requests. The list request is more complex and can result in reads being done from the server task issuing the list request.

Shared-channel-status and shared-sync-key tables

If you are using shared channels, shared-channel-status information and information about the shared-sync-queue are stored in DB2 tables. The fields with names starting SCS* are for DB2 selects, inserts, updates, and deletes from the shared-channel-status table. The fields with names starting SSK* are for DB2 selects, inserts, updates, and deletes for information about the the shared-synch-key table.

The shared-sync-key table is used to locate the message id for messages on the shared sync queue. The Shared Channel Sync queue is used when the channel NPMSPEED(NORMAL) is used. There are messages on the queue have information about the status of messages in a batch. The Shared Sync Key table, provides a mapping from channel name, XMITQ name, and remote queue manager name to the messages for the channel in the Shared Channel Sync queue. Information is inserted into the Shared Sync Key table, when a channel processes messages with NPMSPEED(normal) for the first time. Both of these have times for the thread and the server, as described above.

Interpreting the data

You should monitor the following:

- The average time for server requests.

  The average times in the server are a measure of the response time from DB2. Update requests to DB2 (update, write, and delete) have to wait for DB2 to log the changes, the response time will typically be between 5 and 10 milliseconds. If the average time is larger than this, you should investigate the DB2 system.

- The average difference between the wait time on thread and the time on the server.

  If the difference between the wait time on thread, and time on server (for example readtcuw-readscuw) is greater than a millisecond, this indicates that there was a delay before the server could process the request, and so you should increase the number of server tasks. You change the number of server task using the DB2Servers parameter of keyword QSGDATA in the CSQ6SYSP macro.

- The peak number of requests queued up for the server (field DHIGMAX).

  The field DHIGMAX tells you the maximum number of requests queued waiting for a server. If this value is greater than 10 you should consider increasing the number of servers. In our testing, this value was 20 when we started 1000 channels at the same time; the rest of the time it was usually 1.

Notes:

1. The maximum time waiting for a request can occasionally be large if the information is not in the DB2 buffers and so DB2 has to read it from disk.
2. The list request average and maximum values might be large if a lot of data is requested.

Copying the statistics data to DB2 tables

You can use Performance Reporter to copy the data from the SMF record into a DB2 table.

The figure below shows part of the some Performance Reporter statements that will copy the DB2 statistics to a DB2 table, where the columns in the DB2 table match the names in the MQSeries record.

Performance Reporter definition for MQSeries DB2 statistics

```
DEFINE UPDATE MQ_DB2_Calls
  FROM SMF_115_02
  TO   DRL.MQ_DB2_Calls
  GROUP BY
   (DATE             = SM115DTE,
    PERIOD_NAME      = VALUE(PERIOD(SM115SID,SM115DTE,
                             TIME(SM115TME)),'?'),
    TIME             = TIME(ROUND(SM115TME,5 MINUTES)),
    MVS              = SM115SID,
    QMGR             = SM115SSI)
  SET
   (
    NUMTASK             = MAX(NUMTASK) ,
    ACTTASK             = MAX(ACTTASK) ,
    CONNCNT             = MAX(CONNCNT) ,
    DISCCNT             = SUM(DISCCNT ),
    DHIGMAX             = MAX(DHIGMAX ),
    ...
    DELETCUW     = SUM( DELETCUW /4096),
    DELETMXW     = MAX( DELETMXW /4096),
    ...
    );
```

Performing queries on the data

You can use DB2 to perform queries on the data. The figure below shows part of a query that displays the following:

- The average difference between the time the thread had to wait and the time the server was executing
- The average time the thread had to wait
- The maximum time the thread had to wait
- The maximum time the server had to wait
- The maximum number of requests on the server work queue

Sample DB2 query for displaying statistics

```
SELECT 'READ  ',(READTCUW-READSCUW)/READCNT,READTCUW/READCNT,READTMXW,
                READSMXW,DHIGMAX

  FROM DRL.MQ_DB2_CALLS
  WHERE READCNT> 0
UNION
SELECT 'WRITE ',(WRITTCUW-WRITSCUW)/WRITCNT,WRITTCUW/WRITCNT,WRITTMXW,
                WRITSMXW,DHIGMAX
  FROM DRL.MQ_DB2_CALLS
  WHERE WRITCNT> 0
 ...
```

Part of the output from this query is displayed in below.

Sample output of DB2 statistics. Part of the output from the DB2 query in.

```
                    Times in uSecs for DB2 request
VERB        AVG T-S        AVG T        MAX T        MAX S  DHIGMAX
------  -----------  -----------  -----------  -----------  -------
DELETE           66        22363        77011        77011        1
LIST            281        62131       405399       405291        1
READ             72         3808         6032         5961        1
UPDATE          247         5791         6125         6011        1
WRITE            85         8390        59886        59796        1
```

Exception queries

The queries shown below illustrate how you can monitor out of line conditions described in "Interpreting the data" on page 40.

- The peak number of requests queued up for the server:

```
Select DATE,TIME,MVS,QMGR,DHIGMAX from DRL.MQ_DB2_CALLS
where DHIGMAX > 20
```

- The average difference between the wait time on thread and the time on the server:

```
SELECT 'WRITE ',(WRITTCUW-WRITSCUW)/WRITCNT
  from DRL.MQ_DB2_CALLS
  Where WRITCNT > 0 and (WRITTCUW-WRITSCUW)/WRITCNT > 1000
```

- The average time for server requests:

```
Select DATE,TIME,MVS,QMGR,'WRITE  ', WRITTCUW/WRITCNT
  where WRITCNT > 0 and WRITTCUW/WRITCNT > 10000
Union
  ....
Select DATE,TIME,MVS,QMGR,'WRITE  ', SCSUSCUW/SCSUPD
  from DRL.MQ_DB2_CALLS
  where SCSUPD  > 0 and SCSUSCUW/SCSUPD > 10000
Union
  ...
Select DATE,TIME,MVS,QMGR,'WRITE  ', SSKUSCUW/SSKUPD
  from DRL.MQ_DB2_CALLS
  where SSKUPD  > 0 and SSKUSCUW/SSKUPD > 10000
Union
  ...
```

DB2 statistics record (Q5ST)

Table 6. DB2 statistics record

Offsets		Type	Len	Name (Dim)	Description
Dec	Hex				
0	(0)	Structure	488	Q5ST	DB2 manager statistics
0	(0)	Bitstring	2	Q5STID	Control block identifier
2	(2)	Unsigned	2	Q5STLL	Control block length
4	(4)	Character	4	Q5STEYEC	Control block eye catcher
8	(8)	Character	480	Q5STZERO	QMST part cleared on occasion
8	(8)	Unsigned	4	NUMTASK	Number of server tasks
12	(C)	Unsigned	4	ACTTASK	Number of active server tasks
16	(10)	Unsigned	4	CONNCNT	Number of connect requests
20	(14)	Unsigned	4	DISCCNT	Number of disconnect requests
24	(18)	Unsigned	4	DHIGMAX	Max. request queue depth
28	(1C)	Unsigned	4	ABNDCNT	Number of DB2SRV task abends
32	(20)	Unsigned	4	REQUCNT	Number of requests requeued
36	(24)	Unsigned	4	DEADCNT	Number of deadlock timeouts
40	(28)	Unsigned	4	DELECNT	Number of delete requests
44	(2C)	Unsigned	4	LISTCNT	Number of list requests
48	(30)	Unsigned	4	READCNT	Number of read requests
52	(34)	Unsigned	4	UPDTCNT	Number of update requests
56	(38)	Unsigned	4	WRITCNT	Number of write requests
60	(3C)	Unsigned	4	SCSSEL	SCST selects (shared-channel-status)
64	(40)	Unsigned	4	SCSINS	SCST inserts (shared-channel-status)
68	(44)	Unsigned	4	SCSUPD	SCST updates (shared-channel-status)
72	(48)	Unsigned	4	SCSDEL	SCST deletes (shared-channel-status)
76	(4C)	Unsigned	4	SSKSEL	SSKT selects (shared-sync-key)
80	(50)	Unsigned	4	SSKINS	SSKT inserts (shared-sync-key)
84	(54)	Unsigned	4	SSKDEL	SSKT deletes (shared-sync-key)
88	(58)	Unsigned	4	SCSBFTS	SCST number of times buffer too small
92	(5C)	Unsigned	4	SCSMAXR	SCST maximum rows on query
96	(60)	Unsigned	4	* (2)	Reserved
104	(68)	Character	8	DELETCUW	Cumulative STCK difference - Thread delete
112	(70)	Character	8	DELETMXW	Maximum STCK difference - Thread delete
120	(78)	Character	8	DELESCUW	Cumulative STCK difference - SQL delete
128	(80)	Character	8	DELESMXW	Maximum STCK difference - SQL delete
136	(88)	Character	8	LISTTCUW	Cumulative STCK difference - Thread list
144	(90)	Character	8	LISTTMXW	Maximum STCK difference - Thread list
152	(98)	Character	8	LISTSCUW	Cumulative STCK difference - SQL list
160	(A0)	Character	8	LISTSMXW	Maximum STCK difference - SQL list
168	(A8)	Character	8	READTCUW	Cumulative STCK difference - Thread read
176	(B0)	Character	8	READTMXW	Maximum STCK difference - Thread read

184	(B8)	Character	8	READSCUW	Cumulative STCK difference - SQL read
192	(C0)	Character	8	READSMXW	Maximum STCK difference - SQL read
200	(C8)	Character	8	UPDTTCUW	Cumulative STCK difference - Thread update
208	(D0)	Character	8	UPDTTMXW	Maximum STCK difference - Thread update
216	(D8)	Character	8	UPDTSCUW	Cumulative STCK difference - SQL update
224	(E0)	Character	8	UPDTSMXW	Maximum STCK difference - SQL update
232	(E8)	Character	8	WRITTCUW	Cumulative STCK difference - Thread write
240	(F0)	Character	8	WRITTMXW	Maximum STCK difference - Thread write
248	(F8)	Character	8	WRITSCUW	Cumulative STCK difference - SQL write
256	(100)	Character	8	WRITSMXW	Maximum STCK difference - SQL write
264	(108)	Character	8	SCSSTCUW	Cumulative STCK difference - Thread select
272	(110)	Character	8	SCSSTMXW	Maximum STCK difference - Thread select
280	(118)	Character	8	SCSSSCUW	Cumulative STCK difference - SQL select
288	(120)	Character	8	SCSSSMXW	Maximum STCK difference - SQL select
296	(128)	Character	8	SCSITCUW	Cumulative STCK difference - Thread insert
304	(130)	Character	8	SCSITMXW	Maximum STCK difference - Thread insert
312	(138)	Character	8	SCSISCUW	Cumulative STCK difference - SQL insert
320	(140)	Character	8	SCSISMXW	Maximum STCK difference - SQL insert
328	(148)	Character	8	SCSUTCUW	Cumulative STCK difference - Thread update
336	(150)	Character	8	SCSUTMXW	Maximum STCK difference - Thread update
344	(158)	Character	8	SCSUSCUW	Cumulative STCK difference - SQL update
352	(160)	Character	8	SCSUSMXW	Maximum STCK difference - SQL update
360	(168)	Character	8	SCSDTCUW	Cumulative STCK difference - Thread delete
368	(170)	Character	8	SCSDTMXW	Maximum STCK difference - Thread delete
376	(178)	Character	8	SCSDSCUW	Cumulative STCK difference - SQL delete
384	(180)	Character	8	SCSDSMXW	Maximum STCK difference - SQL delete
392	(188)	Character	8	SSKSTCUW	Cumulative STCK difference - Thread select
400	(190)	Character	8	SSKSTMXW	Maximum STCK difference - Thread select
408	(198)	Character	8	SSKSSCUW	Cumulative STCK difference - SQL select
416	(1A0)	Character	8	SSKSSMXW	Maximum STCK difference - SQL select
424	(1A8)	Character	8	SSKITCUW	Cumulative STCK difference - Thread insert
432	(1B0)	Character	8	SSKITMXW	Maximum STCK difference - Thread insert
440	(1B8)	Character	8	SSKISCUW	Cumulative STCK difference - SQL insert
448	(1C0)	Character	8	SSKISMXW	Maximum STCK difference - SQL insert
456	(1C8)	Character	8	SSKDTCUW	Cumulative STCK difference - Thread delete
464	(1D0)	Character	8	SSKDTMXW	Maximum STCK difference - Thread delete
472	(1D8)	Character	8	SSKDSCUW	Cumulative STCK difference - SQL delete
480	(1E0)	Character	8	SSKDSMXW	Maximum STCK difference - SQL delete

Cross reference

Name	Hex Offset
ABNDCNT	1C
ACTTASK	C
CONNCNT	10
DEADCNT	24
DELECNT	28
DELESCUW	78
DELESMXW	80

Name	Hex Offset
DELETCUW	68
DELETMXW	70
DHIGMAX	18
DISCCNT	14
LISTCNT	2C
LISTSCUW	98
LISTSMXW	A0

Name	Hex Offset
LISTTCUW	88
LISTTMXW	90
NUMTASK	8
Q5ST	0
Q5STEYEC	4
Q5STID	0
Q5STLL	2

Name	Hex Offset
Q5STZERO	8
READCNT	30
READSCUW	B8
READSMXW	C0
READTCUW	A8
READTMXW	B0
REQUCNT	20

Name	Hex Offset
SCSBFTS	58
SCSDEL	48
SCSDSCUW	178
SCSDSMXW	180
SCSDTCUW	168
SCSDTMXW	170
SCSINS	40
SCSISCUW	138
SCSISMXW	140
SCSITCUW	128
SCSITMXW	130
SCSMAXR	5C

Name	Hex Offset
SCSSEL	3C
SCSSSCUW	118
SCSSSMXW	120
SCSSTCUW	108
SCSSTMXW	110
SCSUPD	44
SCSUSCUW	158
SCSUSMXW	160
SCSUTCUW	148
SCSUTMXW	150
SSKDEL	54
SSKDSCUW	1D8

Name	Hex Offset
SSKDSMXW	1E0
SSKDTCUW	1C8
SSKDTMXW	1D0
SSKINS	50
SSKISCUW	1B8
SSKISMXW	1C0
SSKITCUW	1A8
SSKITMXW	1B0
SSKSEL	4C
SSKSSCUW	198
SSKSSMXW	1A0
SSKSTCUW	188

Name	Hex Offset
SSKSTMXW	190
UPDTCNT	34
UPDTSCUW	D8
UPDTSMXW	E0
UPDTTCUW	C8
UPDTTMXW	D0
WRITCNT	38
WRITSCUW	F8
WRITSMXW	100
WRITTCUW	E8
WRITTMXW	F0

The field names in the record reflect the content, for example SCSSTCUW is

SCSSTCUW SCS is for Shared Channel Status

SCS**S**TCUW S is for Select, I for Insert, U for Update, D for Delete

SCSS**T**CUW T is for Thread wait, S is for SQL waits

SCSST**CUW** CUW is for CUmulative Wait, MXW is for MaXimum Wait

So SCSSSCUW is for the Shared Channel Status, Select request, SQL time, Cumulative wait time.

Examples of some MQSeries' DB2 statistics

A queue manager doing no work

The statistics are given below for a 15 minute period.

```
DB2 manager : Q5ST
  Tasks  : Servers  4 Active  5 Conns   0 Discs   0 High 1 Abend  0 Requeue  0
  Number of deadlock conditions :         0
  Lists :  #:180 Task avg m/s :  2 Task max m/s : 3   DB2 avg m/s :  2  DB2 max m/s :  2
```

1. There were 4 DB2 tasks requested(NUMTASK), and the Queue Manager started another task, so there are 5 active(ACTTASK).
2. The maximum number of requests queued was 1(DHIGMAX).
3. There were 180 read requests(READCNT) taking an average of 2 ms(READTCUW/READCNT)and a maximum of 3 ms(READTMXW). Most of this time was spent in DB2(READSCUW, READSMXW).
4. The 180 list requests were in a 15 minute period, this is one request every 5 seconds. This is due to the queue manager querying DB2 for any changed definitions every 5 seconds.

Defining a shared queue

The command "DEF QL(SQFRED) QSGDISP(SHARED) LIKE(SQ0000)" was issued. The statistics produced are given below.

```
DB2 manager : Q5ST
  Tasks  : Servers  4 Active  5 Conns   0 Discs   0 High 1 Abend  0 Requeue    0
  Number of deadlock conditions :         0
  Reads  : #:   3 Task avg m/s :  1 Task max m/s :   1 DB2 avg m/s :   1  DB2 max m/s :    1
  Writes : #:   1 Task avg m/s :110 Task max m/s : 110 DB2 avg m/s : 109  DB2 max m/s :  109
  Lists  : #:  12 Task avg m/s :  2 Task max m/s :   3 DB2 avg m/s :   2  DB2 max m/s :    3
```

1. There were 3 read requests taking a maximum and average of 1 ms, most of this time was spent in DB2.

2. There was 1 write request (WRITCNT). The (average) time was 110 milliseconds (WRITTCUW/WRITCNT) most of which (109 milliseconds) was spent in DB2 (WRITSCUW/WRITCNT).
3. The number of list requests was 12, which happen every 5 seconds, so a total time of 60 seconds, which matches the SMF interval of 61 seconds.

In the example below, a shared receiver channel was started. This accessed the Shared Channel Status table, and the Shared Key Table.

```
DB2 manager : Q5ST
  Tasks  : Servers 4 Active 5 Conns  0 Discs 0 High 1 Abend 0 Requeue  0 Reads  : #:  19 Task avg m/s
:  6 Task max m/s :
  Lists  : #:  56 Task avg m/s :  3 Task max m/s :  84
  SCS Maximum rows returned on query   :   2
  SCS Selects: #: 1 Task avg m/s: 2 Task max m/s:  2 DB2 avg m/s : 2 DB2 max m/s :  2
  SCS Inserts: #: 1 Task avg m/s: 4 Task max m/s:  4 DB2 avg m/s : 4 DB2 max m/s :  4
  SCS Updates: #: 1 Task avg m/s: 6 Task max m/s:  6 DB2 avg m/s : 5 DB2 max m/s :  5
  SCS Deletes: #: 1 Task avg m/s: 6 Task max m/s:  6 DB2 avg m/s : 5 DB2 max m/s :  5
  SSK Selects: #: 1 Task avg m/s: 1 Task max m/s:  1 DB2 avg m/s : 1 DB2 max m/s :  1
```

1. When a channel starts it inserts an record into the Shared Channel Status table - which may exist already, so the number of inserts is 1.
2. When a channel stops naturally (not as a result of a stop channel) then the record is deleted from the Shared Channel Status table. The channel disconnect interval expired, so the channel stopped. There was one delete request.
3. The maximum rows returned on a query was 2 (SCSMAXR).
4. There was one select from the Shared Channel Status table(SCSSEL) which took on average 2 ms(SCSSTCUW/SCSSEL), and a maximum of 2 ms (SCSSTMXW).
5. Of this time the amount time in DB2 was 2 ms(SCSSSCUW/SCSSEL), with a maximum of 2ms(SCSSSMXW).

Coupling Facility statistics

The Coupling Facility manager manages the interface with the Coupling Facility.

When using shared queues, messages are stored in the Coupling Facility (CF). During a commit of messages being processed within syncpoint, information about the messages is changed to indicate that the commit has occurred. The Coupling Facility can perform updates on many messages in one request. Updates to elements in the CF can be made one at a time, or as a group of changes that have to be made together. Conceptually, putting two messages in syncpoint followed by a commit is two requests to update an individual element followed by one request to act on both elements.

The requests to update one element are performed using the IXLLSTE request, and the requests to update multiple elements are performed using the IXLLSTM request.

When these requests are issued, the Coupling Facility might not be able to process the request completely, and so the request might have to be reissued. The number of times a request was reissued is recorded in the statistics as the number of redrives.

How the data is stored

The Coupling Facility statistics have a header and a number of records. Currently it is a 4096 byte data segment with 64 bytes for each of the possible 64 structures. Unused structures have a QESTSTR containing nulls.

What you should monitor

- You should monitor the average time for the various requests. For example `(QESTSSTC/4096)/QESTCSEC` (the division by 4096 converts the STCK value to microseconds).
- If the `Number of structure full` is greater than zero you need to determine if this is due to a transient problem or due to an increasing trend. You may want to increase the size of the structure. If this value is non zero, you will have got OS/390 console messages indicating capacity problems with the structure which you should act on.

Typical query

The query in below produced the output following it. This assumes the appropriate Performance Reporter definitions have been used to update the DB2 table.

Sample DB2 query on Coupling Facility statistics. The column names in DB2 table are the same as the names in the statistics record. There is one row per entry in the array, where there is data in the array.

```
SELECT 'SINGLE  ',QESTSTR,QESTSFUL,QESTSSTC/QESTCSEC,
                  QESTRSEC,QESTMNUS,QESTMLUS
  FROM DRL.MQ_CF_CALLS
  WHERE QESTCSEC  = 0
UNION
SELECT 'MULTIPLE',QESTSTR,QESTSFUL,QESTMSTC/QESTCMEC,
                  QESTRMEC,QESTMNUS,QESTMLUS
  FROM DRL.MQ_CF_CALLS
WHERE QESTCMEC    = 0
```

Sample output from Coupling Facility statistics

```
TYPE      STR NAME      FULL    AVGE TIME  REDRIVE     MAX ELEM      MAX
--------  ------------  ----  -----------  -------  -----------  --------
```

```
SINGLE     CSQ_ADMIN          0              33          0          1127          1
```

Coupling Facility record layout (QEST)

Table 6. Coupling Facility record layout

Offsets		Type	Len	Name	Description
Dec	Hex				
0	(0)	Structure	4104	QEST	CF manager statistics
0	(0)	Bitstring	2	QESTID	Control block identifier
2	(2)	Unsigned	2	QESTLL	Control block length
4	(4)	Character	4	QESTEYEC	Control block eye catcher
8	(8)	Character	4096	QESTZERO	QEST part cleared on occasion
8	(8)	Character	64	QESTSTUC (0:63)	Array (one entry per structure)
8	(8)	Character	12	QESTSTR	Structure name
20	(14)	Unsigned	4	QESTSTRN	Structure number
24	(18)	Unsigned	4	QESTCSEC	Number of IXLLSTE calls
28	(1C)	Unsigned	4	QESTCMEC	Number of IXLLSTM calls
32	(20)	Character	8	QESTSSTC	Time spent doing IXLLSTE calls
40	(28)	Character	8	QESTMSTC	Time spent doing IXLLSTM calls
48	(30)	Unsigned	4	QESTRSEC	Number of IXLLSTE redrives
52	(34)	Unsigned	4	QESTRMEC	Number of IXLLSTM redrives
56	(38)	Unsigned	4	QESTSFUL	Number of structure fulls
60	(3C)	Unsigned	4	QESTMNUS	Maximum number of entries in use
64	(40)	Unsigned	4	QESTMLUS	Maximum number of elements in use
68	(44)	Character	4	*	Reserved
4104	(1008)	Character	0	*	End of control block

The QESTSTUC is an array of 64 elements, each element is 64 bytes long.

Cross reference

Name	Hex Offset
QEST	0
QESTCMEC	1C
QESTCSEC	18
QESTEYEC	4
QESTID	0
QESTLL	2

Name	Hex Offset
QESTMLUS	40
QESTMNUS	3C
QESTMSTC	28
QESTRMEC	34
QESTRSEC	30
QESTSFUL	38

Name	Hex Offset
QESTSSTC	20
QESTSTR	8
QESTSTRN	14
QESTSTUC	8
QESTZERO	8

Examples of some MQSeries CF statistics

A transaction of put commit with a 1000 byte non persistent message was repeated 1000 times.

```
CF manager : QEST
  Structure #:        0, Name      CSQ_ADMIN   , Structure-fulls   0
  Single           1000, Avg time uS         53, Single   retries  0
  Multiple            0, Avg time uS          0, Multiple retries  0
  Max used entries      344, Max used elements       666
  Structure #:        1, Name      APPLICATION1, Structure-fulls   0
  Single           1000, Avg time uS         73, Single   retries  0
  Multiple         1000, Avg time uS        211, Multiple retries  0
  Max used entries     1313, Max used elements      7192
```

1. In the structure name(QESTSTR) corresponding to structure number 0 (QESTSTRN) the number of single requests(QESTCSEC) corresponds to the number of commit requests.
2. The average time for these request (QESTSSTC/QESTCSEC) is 53 micro seconds.
3. In the application structure APPLICATION1 there were 1000 single requests, corresponding to the number of puts, and 1000 requests where potentially multiple messages were committed in one call.
4. Before the run there were 313 used entries, so with 1000 messages the number of used entries is 313 + 1000 = 1313.
5. Before the run there were 1192 elements used.
6. A 1,000 byte message is composed of 6 256 byte segments.
7. The number of segments used is 6*1000 = 6000.
8. The number of segments before + number of segments used = 1192 + 6000 = 7192.
9. An ICS link was used for the Coupling Facility, which has a better response time than a CFS link.

A transaction of get commit with a 1000 byte non persistent message was repeated 1000 times.

```
CF manager : QEST
  Structure #:        0, Name      CSQ_ADMIN   , Structure-fulls   0
  Single           1000, Avg time uS         43, Single   retries  0
  Multiple            0, Avg time uS          0, Multiple retries  0
  Max used entries      344, Max used elements       389
  Structure #:        1, Name      APPLICATION1, Structure-fulls   0
  Single           1000, Avg time uS         52, Single   retries  0
  Multiple         1000, Avg time uS        228, Multiple retries  0
  Max used entries     1037, Max used elements      6090
```

1. The statistics are similar to the put example above.

A transaction of put commit with a 10,000 byte non persistent message was repeated 1000 times. Then another transaction issued get commit of the messages.

```
CF manager : QEST
  Structure #:        0, Name      CSQ_ADMIN   , Structure-fulls      0
  Single           2000, Avg time uS         50, Single   retries     0
  Multiple            0, Avg time uS          0, Multiple retries     0
  Max used entries      344, Max used elements       389
  Structure #:        1, Name      APPLICATION1, Structure-fulls      0
  Single           4000, Avg time uS        218, Single   retries  1000
  Multiple         1000, Avg time uS         63, Multiple retries     0
  Max used entries     1040, Max used elements     41248
```

1. Before the measurement the Max used entries was 40 and the Max used elements was 248.
2. There are 2000 single requests for the CSQ_ADMIN structure because there are 1000 for the put requests, and 1000 for the get requests.
3. 1000 messages were processed, and the Max used entries is 40 + 1000
4. A 10,000 byte message is stored in 256 segments, so 1000 messages used 41248 - 248 = 41,000 or 41 segments per message.
5. It is quicker to use a 4KB buffer than a 64KB buffer to get data There are 1000 single retries. When a message is got from the CF an attempt is made using a 4KB, if this is not large enough the message is got using a 64KB buffer. This getting a message using the larger buffer counts as a retry of reading the CF. As a 10,000 byte messages were being retrieved, the size is greater than 4KB so there was a retry for each message.
6. There were 4000 single CF requests, of which 1000 were retries so there were 3000 successful
7. The calculation of the average time uses the total number of requests. This includes the number and time for when the buffer was too small, in this case the response time was of the order of 10's of micro seconds, compared to the 100's of microseconds when the message was retrieved
   A more typical response time from the CF should exclude the count of the retries. So excluding the count of the retries from the calculation gives the response time = (4000 * 218)/(4000-1000) = 290 micro seconds. So the response time to actually get a message from the CF is between 218 and 290 micro seconds on average.

Message manager statistics

The message manager processes the MQI verbs.

The following table shows the format of the message manager statistics record. It is defined by member CSQDQMST.

Table 7. Structure of the message manager statistics record QMST

Offsets		Type	Len	Name	Description
Dec	Hex				
0	(0)	Structure	48	QMST	Message manager statistics
0	(0)	Bitstring	2	QMSTID	Control block identifier
2	(2)	Unsigned	2	QMSTLL	Control block length
4	(4)	Character	4	QMSTEYEC	Control block eye catcher (QMST)
8	(8)	Signed	4	QMSTOPEN	Number of **MQOPEN** requests
12	(C)	Signed	4	QMSTCLOS	Number of **MQCLOSE** requests
16	(10)	Signed	4	QMSTGET	Number of **MQGET** requests
20	(14)	Signed	4	QMSTPUT	Number of **MQPUT** requests
24	(18)	Signed	4	QMSTPUT1	Number of **MQPUT1** requests
28	(1C)	Signed	4	QMSTINQ	Number of **MQINQ** requests
32	(20)		4		Reserved
36	(24)	Signed	4	QMSTSET	Number of **MQSET** requests
40	(28)		4		Reserved
44	(2C)	Signed	4	QMSTCALH	Number of "close handle" requests

Interpretation

The data gives you counts of different MQI requests. There are no fields you should monitor on a regular basis.

Data manager statistics

The data manager manages the links between messages and queues. It calls the buffer manager to process the pages with messages on them.

The following table shows the format of the data manager statistics record. It is defined in member CSQDQIST.

Table 8. Structure of the data manager statistics record QIST

Offsets		Type	Len	Name	Description
Dec	Hex				
0	(0)	Structure	60	QIST	Data manager statistics
0	(0)	Bitstring	2	QISTID	Control block identifier
2	(2)	Unsigned	2	QISTLL	Control block length
4	(4)	Character	4	QISTEYEC	Control block eye catcher (QIST)
8	(8)	Unsigned	4	QISTMGET	Number of message get requests
12	(C)	Unsigned	4	QISTMPUT	Number of message put requests
16	(10)		4		Reserved
20	(14)	Signed	4	QISTDCRE	Number of object create requests
24	(18)	Signed	4	QISTDPUT	Number of object put requests
28	(1C)	Signed	4	QISTDDEL	Number of object delete requests
32	(20)	Signed	4	QISTDGET	Number of object get requests
36	(24)	Signed	4	QISTDLOC	Number of object locate requests
40	(28)		4		Reserved
44	(2C)	Signed	4	QISTALST	Number of Stgclass change requests
48	(30)		4		Reserved
52	(34)		4		Reserved
56	(38)		4		Reserved
60	(3C)		4		Reserved
64	(40)		4		Reserved
68	(44)		4		Reserved
72	(48)		4		Reserved

Interpretation

The data gives you counts of different object requests. There are no fields you should monitor on a regular basis.

Buffer manager statistics

The buffer manager manages the buffer pools in virtual storage and the writing of pages to, and reading pages from, page sets.

The following table shows the format of the buffer manager statistics record. It is defined in member CSQDQPST.

Note: If you have defined a buffer pool, but not used it, no values are set so the buffer manager statistics record will not contain any data.

Table 9. Structure of the buffer manager statistics record QPST

Offsets		Type	Len	Name	Description
Dec	Hex				
0	(0)	Structure	104	QPST	Buffer manager statistics.
0	(0)	Bitstring	2	QPSTID	Control block identifier.
2	(2)	Unsigned	2	QPSTLL	Control block length.
4	(4)	Character	4	QPSTEYEC	Control block eye catcher (QPST).
8	(8)	Signed	4	QPSTPOOL	Buffer pool identifier (0000-0003).
12	(C)	Signed	4	QPSTNBUF	Number of buffers in this buffer pool.
16	(10)	Signed	4	QPSTCBSL	Lowest number of available buffers.
20	(14)	Signed	4	QPSTCBS	Number of available buffers.
24	(18)	Signed	4	QPSTGETP	The number of page get requests where the current page contents are required. This might involve a read DASD operation if the page is not currently in the buffer pool.
28	(1C)	Signed	4	QPSTGETN	The number of **MQGET** requests for a new - or empty - page (that is, no read operation is necessary).
32	(20)	Signed	4	QPSTRIO	The number of page read DASD operations.
36	(24)	Signed	4	QPSTSTW	The number of page updates.
40	(28)	Signed	4	QPSTTPW	Number of pages written to DASD.
44	(2C)	Signed	4	QPSTWIO	The number of page write operations.
48	(30)	Signed	4	QPSTIMW	The number of synchronous page write operations.
52	(34)	Signed	4	QPSTDWT	The number of times the asynchronous write processor was started.
56	(38)	Signed	4	QPSTDMC	The number of times the synchronous page processor was started because the synchronous write threshold was reached.
60	(3C)	Signed	4	QPSTSTL	The number of times a page get request did not find the page already in the buffer pool.
64	(40)	Signed	4	QPSTSTLA	Number of times the hash chain has been changed during a buffer steal.
68	(44)	Signed	4	QPSTSOS	The number of times NO available buffers were found.
72	(48)		32		Reserved.

Interpreting buffer manager statistics

The buffer manager is the component of MQSeries that handles the movement of data between DASD and virtual storage.

Buffer pools are areas of MQSeries virtual storage reserved to satisfy the buffering requirements for MQSeries queues. Each buffer pool contains an installation defined number of 4 KB virtual storage pages or buffers. Page sets are VSAM linear data sets and each page set is associated with a buffer pool. Queues are mapped to

page sets via their storage class attribute. For more information on the relationship between these entities, see the *MQSeries for OS/390 Concepts and Planning Guide*.

Buffer pool 0 contains MQSeries objects and messages. Other buffer pools just contains messages. A buffer pool treats pages containing messages and objects the same way. To be able to estimate the required size of the buffer pools, you must understand their characteristics and how to interpret the buffer manager statistics generated by MQSeries.

A buffer pool can hold MQSeries object definitions, as well as messages, in 4 KB virtual storage pages. MQSeries is designed to keep pages in buffer pool virtual storage as long as possible in order to obtain the best performance.

However, if a buffer pool starts to fill up, pages in the buffer pool which have been updated are written out to their relevant DASD page sets to free up buffer pool space. This happens if, for example, messages are being put onto queues associated with the buffer faster than they are being taken off.

Information contained in pages that have been written out to DASD page sets can be read in again on demand.

Ideally, a transaction pattern should be such that messages do not spend a long time on a queue waiting to be retrieved. This means that messages never have to spill over to DASD because the pages used to hold them remain in virtual storage.

Buffer pool management

To manage your buffer pools efficiently, you must consider the factors that affect the buffer pool I/O operations and also the statistics associated with the buffer pools.

DASD operations

The following factors affect buffer pool I/O operations.

- If a page containing the required data is not found in the buffer pool, it is read synchronously from its DASD page set to an available buffer.
- Whenever a buffer pool page is updated, it is put on an internal queue of pages to be (potentially) written out to DASD. Once a buffer pool page has been updated, it cannot be reused until it has been written to DASD.
- If the number of pages queued to be written to DASD exceeds 85% of the total number of buffers in the pool, an asynchronous write processor is started in order to write the buffers to DASD.
  Similarly, should the number of buffers available for page get requests become less than 15% of the total number of buffers in the pool, then the asynchronous write processor is started in order to perform the write I/O operations. If the number of pages queued to be written to DASD is close to 85%, the number of free pages is likely to be just over the 15% limit, so if there are many applications concurrently browsing a queue, thus using pages, the number of free pages may drop below the 15%.

  The write processor stops when the number of pages queued to be written to DASD has fallen to 75% of the total number of buffer in the pool.
- If the number of pages queued for writing to DASD exceed 95% of the total number of buffers in the pool, all updates result in a synchronous write of the page to DASD.
- If the number of buffers available for page get requests ever reaches zero, a transaction that encounters this condition is suspended until the asynchronous write processor has finished.
- If a page is frequently updated, the page spends most of its time on the queue of pages waiting to be written to DASD. Because this queue is in least recently used order, it is possible that a frequently updated page placed on this least recently used queue will never be written out to DASD. For this reason, at the time of update, if the page is found to have been waiting on the write to DASD queue for at least 2 checkpoints, it will be synchronously written to DASD.
  The aim of the above algorithm is to maximize the time pages spend in buffer pool memory while allowing the system to function should system load put the buffer pool usage under stress.

Fields that you need to monitor daily

You should monitor the QPSTSOS field (the number of times that no buffers were available). If this value is non zero, you should increase the size of your buffer pool and check that the page data sets in the buffer pool are optimally placed to reduce contention.

Fields that you should monitor weekly

The values in the buffer manager statistics vary, depending on the applications that use the buffer pools. You should monitor the values listed below, and investigate any out-of-line conditions, and take the appropriate action. This might be:

- To make the buffer pool bigger.
- To move messages from one page set to another in order to move work to a different buffer pool. You can do this using the COPY and LOAD functions in CSQUTIL. This is described in *MQSeries for OS/390 System Administration Guide*
- To investigate why the message pattern has changed. For example a channel might not be working, so messages are accumulating on a transmission queue.

You should monitor the following:

- `QPSTCBSL/QPSTNBUF` (the ratio of how full the buffer is).
- `QPSTRIO` (the number of pages read from the page set). This might be non zero after a system restart, and zero the rest of the time
- `QPSTDMC` (how many times the task was started to move pages out from the buffer pool to the page set).

Examples of buffer pool statistics.

In the examples below the statistics are given for the buffer pool where the messages are located. There will also be activity in buffer pool 0 as information about the queue is updated, for example the current queue depth.

Putting 1000 1000 byte messages

```
Buffer manager : QPST
 ...
01 #buff      1000 #low       499 #now        499 #getp      2498 #getn       500
01 Rio           0 STW       2499 TPW          0 WIO           0 IMW           0
01 DWT           0 DMC          0 STL        500 STLA          0 SOS           0
```

1. Two 1000 byte message fit into a 4K page
2. 1000 messages require 500 pages, #getn is 500 (field QPSTGETN).
3. When there is space on the current(last) page for a second message, it can use the current page, so we have 1 request for get current page with contents.
   When there is not enough space to add the message to the current(last) page, a new page must be used. The typical flow for adding a page for a 1000 byte message is
   - ☐ get last page - determine there is not enough space on it
   - ☐ get a page which has a list of free pages and locate a free page
   - ☐ get the new page, and format it
   - ☐ get the previous last page and change it to point to the new page
   - ☐ get the new page and put the message in it

The 5 requests are 4 requests to get the page with contents, and one get new page.

Note: The get page requests are usually low cost calls, which just locates the page in memory. When the page has to be read in from the page set then the request is more expensive.

4. So for 2 messages we need 5+1 get page requests, of which 1 is get new, and 5 are get page contents. For 500 pairs of messages we need 500 get new page, and 2500 get old page requests. The exception is for the first message on the queue when there is no last page, so for the first message requires 2 fewer get requests, so we have 2500-2 = 2498. This is the #getp 2498 (field QPSTGETP). Note that this is a simplified description, it gets more complex when you have different sized messages and a busier system.
5. Whenever a buffer pool page is updated a flag is set called Set Write Intent. The number of times this flag is set is given in the STW (field QPSTSTW).
   In the above example, the current page is got
   - ☐ If there is space in the page then the Set Write Intent is set.
   - ☐ If there is no space, then the Set Write Intent is not set, and the flow above is followed. Each of these require an update to the page, so Set Write Intent is set for the rest of the requests.

   So for 2 messages we have 1 Set Write Intent when the new message fits in the existing page, and 4 write intents when a new page is needed. For 500 pairs of messages we have 500*(1+4) = 2500. This is approximately the same as the STW value in the statistics.

   This field gives a measure of how many pages were updated, and how many were browsed, or searched for a message, see "Getting messages from an indexed queue" on page 59 for an example.
6. These puts were done just after the queue manage started so there were no used buffers in the buffer pool, so before the puts, #low (the lowest number of free buffers) and #now (the current number of free buffers) were the same as the buffer pool size of 1000 pages
7. After the puts were done
   1. The number of buffers in the buffer pool is 1000 (field QPSTNBUF).
   2. The lowest number of unused buffers #low is 499 (field QPSTCBSL).
   3. The current number of unused buffers, #now is 499(field QPSTCBS).
      Note the number of buffers used is 1000-499 = 501. This is 500 for the pages containing messages, plus a page which is used internally for keeping track of free/used pages.
8. Because the Queue Manager had just been restarted, the buffer pool was full of unused pages. Whenever a new page was allocated, it did not have a buffer in the buffer pool, so one had to be stolen from the free page list. The number of stolen pages STL was 500 (field QPSTSTL).

Getting 1000 1000 byte messages

```
Buffer manager : QPST
 ...
01 #buff      1000 #low       499 #now        499 #getp      2003 #getn         0
01 Rio           0 STW       1003 TPW          0 WIO           0 IMW           0
01 DWT           0 DMC          0 STL             STLA          0 SOS           0
```

1. There are 0 requests for get new page, as we are getting existing messages, so we do not need any new pages.
2. The 2003 requests to get a page with current contents(#GETP) include the requests from an internal task which relocates empty pages to a free page list.
   - ☐ 1000 get requests each requested a page
   - ☐ In 500 cases there were no more messages on the page just obtained so the next page had to be got, so 500 more pages got for this.
     If the queue had been indexed, then the index would have enabled the correct page to be located directly, and so there would be 500 get pages requests less.

- ☐ An internal task removed empty pages from the queue. When the queue is empty, all 500 pages have been processed, and the last 3 represent when the internal task got a page, but could not free it, because it still had messages on it. This is typical.

3. The pages were all in the buffer pool, as the STL count is zero.

Putting 2000 1000 byte messages

The buffer pool had 188 pages free (#now) before this measurement.

```
Buffer manager : QPST
 ...
01 #Buff      1000 #Low       149 #Now        189 #Getp      6033 #Getn      2011
01 Rio           0 STW       8033 TPW        2012 WIO         503 IMW            0
01 DWT          35 DMC          0 STL        2011 STLA          0 SOS            0
```

1. The total number of pages used for 2000 messages is 1000 pages.
2. When the buffer pool has 15% or less pages then a background task is started to move old pages from the buffer pool to the page set. The task is called the Deferred Write Processor. The DWP stops when there are 25% free buffers in the buffer pool
3. DWT (field QPSTDWT) is 35, which shows that the DWP was started 35 times.
4. The fact that DWT is greater than 0 may or may not be an indicator of a problem. If you are writing messages to a queue for deferred processing, such as overnight, you want a small buffer pool, which will fill up frequently, and so DWT will be large. If you are processing short lived messages you expect to keep your messages in the buffer pool so the fact that DWT is greater than 0 indicates a possible problem.
5. The ratio #now/#buff is less than 25%, so some more pages were used since the last time the DWP ran.
6. The DWP did WIO(503) write requests (field QPSTWIO) and wrote out TPW(2012) pages to the page set (field QPSTTPW).
   The write request process up to 4 pages per I/O, so TPW/WIO is usually close to 4.

   STL (2011) pages were requested which were not in the buffer pool so had to be stolen

7. Before the messages were put to the queue, the buffer pool had 188 free pages. STL(2011) pages were taken from the buffer pool, and 2012 pages were written to the page set. The current number is pages in the buffer pool is 188 - 2011 + 2012 = 189 which is the number of current number of free pages in the buffer pool.
   Note: Normally TPW - STL is only approximately the change in the number of free pages from the start to the end of the interval. For example if a page in the buffer pool is being browsed then the number of free pages is decremented by 1, but the number of pages stolen and the number of pages written would be unchanged. When the browse has finished the page is put back onto the free list, and so the number of pages on is incremented back to the original value.

Getting 2000 1000 byte messages

```
Buffer manager : QPST
 ...
01 #buff      1000 #low       150 #now        195 #getp      4758 #getn         0
01 Rio        1000 STW       2005 TPW         972 WIO         243 IMW           0
01 DWT           9 DMC          0 STL        1000 STLA          0 SOS           0
```

1. The pages containing the oldest messages had been written out to the page set by the DWP task, and the buffers they had been using had been reused by other pages.

2. RIO show that 1000 pages were read from the page set(QPSTRIO), so as 1000 pages had been used for the 2000 messages, all the pages for the messages had been written to the page set
3. As pages were read from disk, they stole a buffer from the free list. As 1000 pages were read in, they needed 1000 pages from the free list, this is the STL (1000) value.
4. As pages are taken from the free list, the number of free pages decreases. When the number of free pages is 15% of the total buffer pool size the Deferred Write Processing task is started to move old pages from the buffer pool to the page set. The DWP task stops when there are 25% free pages available The DWP task was started DWP(9) times.
5. When the DWP task runs, it writes multiple pages to the page set. It had WIO(243) write requests, for a total of TPW(972) pages.
6. The #low reflects the lowest number of free pages in the buffer pool.

Comparing 2 similar scenarios

In the figures below are the buffer pool statistics for the same scenario when the system was running normally, and earlier during system set up when the applications were not started properly, and so messages built up on the input queue.

Normal scenario. Scenario running normally 1,030,000 MQGETs, 722,000 MQPUTs

```
Buffer manager : QPST
 ...
02 #Buff     99000 #Low     92147 #Now      92147 #Getp   3903501 #Getn    360863
02 Rio           0 STW    2526190 TPW          0 WIO           0 IMW           0
02 DWT           0 DMC          0 STL        228 STLA          0 SOS           0
```

Bad setup scenario. Scenario during setup 162,000 MQGETs, 200,000 MQPUTs

```
Buffer manager : QPST
 ...
02 #Buff     99000 #Low     14841 #Now      17287 #Getp    963356 #Getn     98996
02 Rio       35636 STW     657001 TPW      116504 WIO       29126 IMW           0
02 DWT          56 DMC          0 STL       75619 STLA          0 SOS           0
```

1. The normal scenario processed many more message than the bad setup scenario, 722,000 messages put compared to 200,000.
2. In the normal scenario
    1. The lowest number of free pages is 92147 or 99000 - 92147=6853 have been used.
    2. There were 360863 requests for new pages
    3. As 360863 is much greater than 6853 this means that the messages in the pages were short lived, because the same pages were frequently reused, as the number of used buffers is low.
    4. The number of stolen pages (STL) is 228 which means that most of the pages were already in the buffer pool, and 228 had to be acquired. This could be due to the queue depth increasing slightly. In the steady state STL will typically be 0.
    5. The buffer pool did not fill up as the lowest number free pages 92147/99000 = 93% which is much greater than 15%. Also DWT is 0.
    6. The total pages written to disk was 0, so if there was a checkpoint in the statistics interval, no pages were written to disk.
3. In , the bad setup scenario
    1. The lowest number of free pages is 14841 (15%) or 99000 - 14841=84159 have been used.
    2. There were 98996 requests for new pages

3. As 98996 is close to 84159 this means that the messages in the pages were long lived, because the pages were not reused.
4. The number of stolen pages (STL) is 75619 which means that most of the pages were not already in the buffer pool. This could be due to the queue depth increasing significantly
5. The buffer pool did fill up as DWT is 56.
6. The total pages written to disk was 116504.
7. 35636 pages were read from disk.
8. As the number of disk writes is close to the number of disk reads, this indicates that the buffer pool was too small for the work.
   - If there had been few disk reads, but many disk writes, then this is typical of an application putting messages for "overnight" processing.
   - If there are many reads from disk, and few writes then this indicates that these "overnight" messages are being processed.
   - If there are approximately the same number disk writes as disk reads then this could be caused by the buffer pool being too small, or that the "overnight" processing has started, and messages are still being put to the queue.
   - A long period between statistics intervals.

Getting messages from an indexed queue

1000 messages were put to an indexed queue. Then for the measurement a message was put and got by message id, and this repeated 1000 times.

Indexed queue

```
Buffer manager : QPST
 ...
02 #Buff     90000 #Low     89498 #Now      89498 #Getp      2498 #Getn       500
02 Rio           0 STW       2499 TPW           0 WIO           0 IMW           0
02 DWT           0 DMC          0 STL         500 STLA          0 SOS           0
```

1. The number of STW is as expected from item 5 on page 56.

1000 messages were put to an non-indexed queue. Then for the measurement a message was put (to the end of the queue) and got by message id (from the end of the queue), and this repeated 1000 times. Because the queue was not indexed the whole queue has to be searched to find the matching message.

Non Indexed queue

```
Buffer manager : QPST
 ...
02 #Buff     90000 #Low     88790 #Now      88790 #Getp    595913 #Getn       500
02 Rio           0 STW       3512 TPW           0 WIO           0 IMW           0
02 DWT           0 DMC          0 STL         207 STLA          0 SOS           0
```

1. The number of get pages is much larger in comparison to the indexed queue 595913 vs 2498.
2. The number of Set Write Intent STW(3512) is much smaller than the number of get pages (595913), this indicates that there was a lot of browse, or sequential searching activity.
3. The number of pages used in the indexed queue case is getp+getn = 2498+500.
4. The number of pages used in the non indexed queue case is getp+getn = 595913+500.
5. The difference in the number of pages is 593415.
6. For every message got, it had to search 1000 messages, or 500 pages so when processing 1000 messages the total of pages scanned, when looking for messages is 500 * 1000 = 500,000.

7. The difference in the total number of pages used, and the number used by the application is is 593,415-500,000 = 93,415. This is the number of pages got by the internal task which moves empty pages to the free list. The internal task starts at the front of the queue, and moves to the end. Most of the 93,415 pages processed had messages on them, and could not be freed.

Log manager statistics

The log manager is responsible for managing recovery data on log datasets.

- Logging is done for persistent messages, and not for non persistent messages
- Data is written to log buffers
- Data is written from log buffers to active log data sets.
  This occurs
  - During a commit request - for two phase commit there will be two requests
  - During a put or get out of syncpoint
  - When an installation specified number of buffers have been filled and there has been no request to force the buffers to disk.
  - Any define, delete or alter command.
  - An MQSET verb is issued.

The following table shows the format of the log manager statistics record. It is defined in member CSQDQJST.

Table 10. Structure of the log manager statistics record QJST

Offsets		Type	Len	Name	Description
Dec	Hex				
0	(0)	Structure	120	QJST	Log manager statistics.
0	(0)	Character	2	QJSTID	Control block identifier.
2	(2)	Signed	2	QJSTLL	Control block length.
4	(4)	Character	4	QJSTEID	Control block eye catcher (QJST).
8	(8)	Signed	4	QJSTWRW	Write_request count - Wait. This request is converted to a write_force requests, so this value is always zero.
12	(C)	Signed	4	QJSTWRNW	Write_request count - No wait. Data is written to log buffers, these buffers are not explicitly written to the active log data sets, and the requestor is not suspended.
16	(10)	Signed	4	QJSTWRF	Write_request count - Force. Data is written to log buffers, these buffers are then written to the active log data sets, and the requesting task is suspended until the write to active log data sets is complete.
20	(14)	Signed	4	QJSTWTB	Wait count for unavailable buffers. Number of times a task was suspended because all the buffers were waiting to be written to the active log data set.
24	(18)	Signed	4	QJSTRBUF	Number of read log requests satisfied from in-storage buffers.
28	(1C)	Signed	4	QJSTRACT	Number of read log requests satisfied from the active log data set.
32	(20)	Signed	4	QJSTRARH	Number of read log requests satisfied from an archive log data set.
36	(24)	Signed	4	QJSTWTL	See QJSTTVC.
36	(24)	Signed	4	QJSTTVC	Number of read log requests delayed because the number of archive log data sets that could be used was limited by the the MAXRTU parameter in the CSQ6LOGP system parameter macro.
40	(28)	Signed	4	QJSTBSDS	Total number of bootstrap data set (BSDS) access requests.
44	(2C)	Signed	4	QJSTBFFL	The number of active log control intervals (CIs) created (log pages used).
48	(30)	Signed	4	QJSTBFWR	Number of calls made that wrote to active log buffers.
52	(34)	Signed	4	QJSTALR	Number of times an archive log data set was allocated for a read request.
56	(38)	Signed	4	QJSTALW	Number of times an archive log data set was allocated for a write request.
60	(3C)	Signed	4	QJSTCIOF	Count of CIs off-loaded to the archive data set.
64	(40)	Signed	4	QJSTLLCP	Number of times that checkpoint was invoked because the number

					of requests to write to the log buffers was the LOGLOAD value specified in the CSQ6SYSP macro.
68	(44)	Signed	4	QJSTWUR	Number of read accesses delayed due to unavailable resource. This occurs during restart or rollback when using archive logs.
72	(48)	Signed	4	QJSTLAMA	Number of look-ahead tape volume mounts attempted. QJSTLAMA-QJSTLAMS shows how many times look-ahead mounting failed. This value applies during restart or rollback when using archive logs.
76	(4C)	Signed	4	QJSTLAMS	Number of look-ahead tape volume mounts performed. This value applies during restart or rollback when using archive logs.
80	(50)	Signed	4	QJSTLSUS	Number of times a request to write data to buffers was suspended. The request can be suspended because it has to wait until a log buffer has been written to the log data sets, or for example, there were insufficient log buffers - see QJSTWTB.
84	(54)	Signed	4	QJSTLOGW	Total number of write requests to active log data sets.
88	(58)	Signed	4	QJSTCIWR	Total number of log CIs written to active log data sets.
92	(5C)	Signed	4	QJSTSERW	For dual logging this is the number of requests to rewrite a CI, when the I/O was done to each log in series, rather than in parallel. For single logging this is the number of requests to rewrite a CI.
96	(60)	Signed	4	QJSTTHRW	Number of times a log write request was scheduled because the log write threshold (WRTHRSH in the CSQ6LOGP system parameter macro) was reached.
100	(64)	Signed	4	QJSTBPAG	Number of times a log-write buffer had to be paged in before it could be used.
104	(68)	Signed	16		Reserved.
120	(78)	Character	0	QJSTEND	End of log manager statistics block

Cross reference

Name	Hex Offset
QJST	0
QJSTALR	34
QJSTALW	38
QJSTBFFL	2C
QJSTBFWR	30
QJSTBPAG	64
QJSTBSDS	28
QJSTCIOF	3C

Name	Hex Offset
QJSTCIWR	58
QJSTEID	4
QJSTEND	78
QJSTHEAD	0
QJSTID	0
QJSTLAMA	48
QJSTLAMS	4C
QJSTLL	2

Name	Hex Offset
QJSTLLCP	40
QJSTLOGW	54
QJSTLSUS	50
QJSTRACT	1C
QJSTRARH	20
QJSTRBUF	18
QJSTSERW	5C
QJSTTHRW	60

Name	Hex Offset
QJSTTVC	24
QJSTWRF	10
QJSTWRNW	C
QJSTWRW	8
QJSTWTB	14
QJSTWUR	44

Interpreting log manager statistics

- QJSTWRF is the number of times a request was made to force the log buffers to disk. This occurs when:
  - ☐ Persistent messages are put or got out of syncpoint.
  - ☐ A commit or backout request is issued where persistent messages have been processed in syncpoint.
  - ☐ An **MQSET** call has been issued.
  - ☐ An object has been changed using the DEFINE, DELETE or ALTER commands.
- Updates to QJSTRBUF, QJSTRACT, and QJSTRACH occur when work is backed out or at system restart. The number of backouts you have should be small. If you do have backouts, you should try to have the data in log buffers, or on active logs; you should not have tasks needing archive logs.
- QJSTTVC is the number of delays because the MAXRTU limit was reached. (MAXRTU is the maximum number of tape units that can be allocated for archive read.)
- QJSTWUR is the number of delays that were not due to MAXRTU (QJSTTVC). For example this can be caused by not having allocated enough tape units, or a delay due to a WTOR.
- QJSTTHRW is the number of times a log-write request was scheduled because the log write threshold (WRTHRSH in the CSQ6LOGP system parameter macro) was reached. If this value is greater than 0, you might increase throughput by increasing the value of WRTHRSH. Depending on the work load mix, increasing the value of WRTHRSH can occasionally cause an increase in response time. You can monitor this using the queue level statistics.
- QJSTBSDS is the number of requests to write to the BSDS. The BSDS is updated periodically, such as when an active log switches and when the log buffer is about to wrap.

Fields you need to monitor

You should monitor the following fields daily or weekly:

QJSTWTB If this field is non zero you should increase the value of OUTBUFF in the CSQ6LOGP system parameter macro.

QJSTRBUF If this value is large, applications are backing out and not committing requests; this might be an application problem. You can use the queue level statistics to identify which tasks are backing out rather than committing.

QJSTRACT If this value is non zero this means that you have long running tasks that are backing out.

QJSTRARH If this value is non zero this means that you have long running tasks that are backing out and you are having to read from archive logs. You should determine why you have long running tasks backing out, and consider increasing the size or number of your active log data sets.

QPSTBPAG If this field is non zero it indicates a possible problem with your OS/390 system. You might get benefit by decreasing the number of log buffers (OUTBUFF in the CSQ6LOGP system parameter macro) provided that QJSTWTB is zero.

QJSTTVC If this value is on zero then you should investigate why you need so many archive logs, and consider increasing the value of MAXRTU to allow more devices to be used.

QJSTLLCP On a busy system you would expect to see typically 10 checkpoints an hour. If the QJSTLLCP value is larger than this, it indicates a problem in the setup of the queue manager.

The most likely reason for this is that the LOGLOAD parameter of the CSQ6SYSP system parameter macro is too small. The other event that causes a checkpoint is when an active log

fills up and switches to the next active log data set. If your logs are too small, this can cause frequent checkpoints.

You should increase the value of the LOGLOAD parameter, or increase the size of your log data sets as required.

"Display out of line log manager statistics" has an example DB2 query displaying some of the above fields from the log statistics DB2 datbase produced by Performance Reporter.

Examples of some log manager statistics

The examples below are to illustrate the use of common log manager statistics. Many of the statistics are not useful for day to day monitoring, and are not discussed.

The statistics are displayed with the supplied program described in "Supplied programs to print out the SMF records".

Putting 1000 1000 byte messages

A batch application put a 1000 byte message and issued a commit, then repeated this 1000 times.

In the observations following the log statistics the following interesting figures are discussed

- The number of pages used to hold the messages is 467
- There were 2934 write requests, each writing one page

```
Log manager : QJST
  Write_Wait          0 Write_Nowait     10030 Write_Force           0 WTB     0
  Read_Stor           0 Read_Active          0 Read_Archive         0 TVC     0
  BSDS_Reqs          13 CIs_Created       467 BFWR               1000 ALR     0
  ALW                 0 CIs_Offload         0 Checkpoints           0
  WUR                 0 LAMA                0 LAMS                  0
  Write_Susp       1000 Write_Reqs       2934 CI_Writes          2934
  Write_Serl       2000 Write_Thrsh         0 Buff_Pagein           0
```

1. Description of activities involved in putting a message
    1. Two 1000 byte messages fit into a 4K page
    2. When there is space on the current(last) page for a second message, it can use the current page.
        - The current queue depth is incremented, and the change logged.
        - The data is put into the page and the insertion logged.

        When there is not enough space to add the message to the current(last) page, a new page must be used and associated with the queue.
        - The current queue depth is incremented, and the change logged.
        - Allocate a page from the free page list, update the list to reflect the change, and log the changes
        - Format the page, and log data to say the page has been formatted.
        - Insert the page at the end of the queue, adjust various pointers to the new page, and log the changes
        - Put the message into the page and log the data inserted.
    3. For every start Unit Of Work data will be written to the logs buffers

4. For every commit or backout data will be written to the log buffers, and a request made to write the buffers to disk.
5. For the put of a 1000 byte message, the number of writes to the log buffers is typically about 4 + 6* number of messages per unit of work.

2. The number of log pages used (CIs_created) is 467 (field QJSTBFFL). The number bytes used in log buffers is 467 * 4096 = 1,912,832 or about 1900 bytes per message
   This number depends on message size, and how much data needs to be logged.
3. There were 10030 Write_Nowait requests (field QJSTWRNW). These are requests to put data in to log buffers. For example, as well as inserting the message into the page, the current depth of the queue is updated, pages have to be allocated, and pages have to be chained together.
4. There were 1000 Write_susp requests (field QJSTLSUS). This corresponds to the commit request where data is forced out to the log data sets. With 2 phase commit there will be two write suspends per commit.
5. CI_Writes (field QJSTCIWR) shows 2934 pages were written to the log data sets. As there was dual logging this is 1467 per log.
6. Although there were 467 CIs created, there were 1467 pages written to a log data set. This is because some pages were rewritten one or more times. A CI is rewritten if it was only partially filled with log records on the previous write.
7. There were 2934 Write_Reqs (field QJSTLOGW) and 2934 CI_Writes (field QJSTCIWR), or 1467 to each log. As the number of pages written equals the number of write requests this shows that only one page was written for each disk write request. As the rate of persistent messages processed increases you may get more pages per I/O
8. Write_serl (field QJSTSERW) is the number of times a page was rewritten. 2000 with dual logging is 1000 per log data set. In a busy system this value is usually different from the number of number of write suspends.
   The first time a page is written it is written to both logs in parallel. If a CI is rewritten it is written to each log in series. 467 CIs were created. So the first time these are written they will be written in parallel. So 467 of the 1467 CI_write request to a log are parallel, the rest are serial, so 1467-467 is 1000 which is the number of Write_serl requests. On average each page was written 1467/467, or 3-4 times, or rewritten 2-3 times. With PTF UQ61496 on V5.2, if the DASD is cached the I/Os are done in parallel, and not in series.
9. The BFWR write requests is the number of requests to write data out to the log data sets (QJSTBFWR). Internal tasks also issue these requests, and this number is typically higher than the number of application commits.
10. When the last page in the log buffers is used then it causes the BSDS to be updated with information about the information in the active logs. If the last page is rewritten then BSDS information is rewritten. The system had OUTBUFF defined as 400KB, so there were 100 4K pages of log buffers allocated. With 467 CIs created the last page in the log buffers was used 4 or 5 times. So if the last page was written 3-4 (rewritten 2-3 times, see above) times on average, we would expect the number of BSDS requests to be around the range of 3*4 to 4*5 or 12 to 20. The value of 15 matches this.

    When an active log fills up, or a checkpoint occurs the BSDS is updated with information about the active and archive logs, as well as checkpoint information.

Getting 1000 1000 byte messages

A batch application got a 1000 byte message and issued a commit, then repeated this 1000 times.

```
Log manager : QJST
  Write_wait         0 Write_Nowait      6511 Write_Force           2 WTB         0
  Read_stor          0 Read_Active          0 Read_archive         0 TVC         0
  BSDS_reqs         11 CIs_created         97 BFWR               1004 ALR         0
  ALW                0 CIs_offload          0 Checkpts             0
  WUR                0 LAMA                 0 LAMS                 0
```

```
  Write_susp      1004 Write_Reqs        2202 CI_Writes          2202
  Write_serl      2008 Write_Thrsh          0 Buff_Pagein           0
```

1. 1000 messages got had 97 CIs_created. This is about 97*4096/1000 or about 400 bytes per message
2. There were 2202 pages written, or 1101 pages per log data set.
3. The average number of times a page was written is 1101/97 or about 11 times.
4. The number of Write_susp requests is the number of application commits plus some requests from internal tasks.
5. 97 pages were written. The first time these pages were written they were written in parallel. The remained were written in serial so each log wrote 2008/2 = 1004 pages, in series. With the 97 in parallel we have total CIs written = 1004 + 97 = 1101 which matches the number of CI_Writes requests.

Putting 100 100,000 byte messages

A batch application put a 100,000 byte message and issued a commit, then repeated this 100 times.

```
Log manager : QJST
  Write_wait         0 Write_Nowait      8214 Write_Force           0 WTB         0
  Read_stor          0 Read_Active          0 Read_archive         0 TVC         0
  BSDS_reqs         26 CIs_created       2550 BFWR                200 ALR         0
  ALW                0 CIs_offload          0 Checkpts             0
  WUR                0 LAMA                 0 LAMS                 0
  Write_susp       100 Write_Reqs         844 CI_Writes          5300
  Write_serl       200 Write_Thrsh        100 Buff_Pagein           0
```

1. For 100 messages there were 2550 CIs_created. This equates to 2550/100 pages per message or 2550*4096/100 or about 105000 bytes per message.
2. Each log wrote 5300/2 = 2650 pages.
3. 2550 of these 2650 were writing the data to the log, so 2650-2550 = 100 is the number of rewrite requests.
4. 2650 pages were written in 844/2 write requests, or about 6 pages per write request. This shows more data is written for each I/O request
5. Each message needs more than 26 pages of log buffers. The WRTHRSH was specified as 15 pages. This means that no more than 15 pages will be written in each I/O request. So for each message the WRTHRSH value was exceeded, and can be seen in the value of Write_Thrsh(100) field(QJSTTHRW) which in this case matches the number of messages processed.
6. There were 100 pages of log buffers, so the 2550 CIs created means that each page was used about 2550/100 times, or 25-26 times. The number of BSDS requests (QJSTBSDS) is 25, which matches the 25-26 times.

Getting 100 100,000 byte messages

A batch application got a 100,000 byte message and issued a commit, then repeated this 100 times.

```
Log manager : QJST
  Write_wait         0 Write_Nowait      3108 Write_Force           1 WTB         0
  Read_stor          0 Read_Active          0 Read_archive         0 TVC         0
  BSDS_reqs          5 CIs_created         30 BFWR                102 ALR         0
  ALW                0 CIs_offload          0 Checkpts             0
  WUR                0 LAMA                 0 LAMS                 0
  Write_susp       102 Write_Reqs         264 CI_Writes          264
  Write_serl       204 Write_Thrsh          0 Buff_Pagein          0
```

1. Getting 100 messages had 30 CIs_created. This is about 1200 bytes per message

2. There were 132 pages written per log data set, so the pages were written 4 or 5 times.

Processing messages concurrently

10 Batch jobs each put a 1000 byte message to a server queue and waited for a reply. Each job did this 1000 times, so there were 10,000 messages requests and 10,000 reply messages processed.

There were 3 server jobs getting from a server queue and sending a reply back to the originator. The processing was get commit, put commit (to simulate two phase commit).

```
Log manager : QJST
  Write_wait          0 Write_Nowait    302738 Write_Force         247 WTB         0
  Read_stor           0 Read_Active          0 Read_archive         0 TVC         0
  BSDS_reqs         271 CIs_created      11010 BFWR              41263 ALR         0
  ALW                 0 CIs_offload          0 Checkpts              0
  WUR                 0 LAMA                 0 LAMS                  0
  Write_susp      41260 Write_Reqs       72084 CI_Write          72298
  Write_serl      50278 Write_Thrsh          0 Buff_Pagein           0
```

1. 10,000 messages put to the server and 10,000 replies is 20,000 messages. The number of write_susp is 41260 which reflects 40,000 commits from the applications, and 260 commits from the internal task which removes empty pages from queues.
2. There were 11010 CIs_created. This equates to 11010*4096/20,000 or 2255 bytes per message. From the figures above for puts and gets the number of bytes per message is 1900 + 400 = 2300 which is approximately the same.
3. A write to a log data set can process one or more CIs. Each log processed 72298/2= 36149 pages, in 72084/2= 36042 write requests, so most I/O requests processed only one CI.
4. Each log processed 50278/2 = 25139 Write_serial request, so in most cases each page was rewritten 25139/11010 times - or written in parallel once and rewritten 2-3 times serially

After an archive log command was issued

```
Log manager : QJST
  Write_Wait          0 Write_Nowait         73 Write_Force           1 WTB         0
  Read_Stor           0 Read_Active           0 Read_Archive          0 TVC         0
  BSDS_Reqs         222 CIs_Created           2 BFWR                  3 ALR         0
  ALW                 2 CIs_Offload        6020 Checkpoints          0
  WUR                 0 LAMA                 0 LAMS                  0
  Write_Susp          1 Write_Reqs          10 CI_Writes            10
  Write_Serl          6 Write_Thrsh          0 Buff_Pagein           0
```

1. The number of CIs offloaded is 6020, (field QJSTCIOF)
2. Data was written to active logs, Write_Nowait is > 0, and one Write_force. This is checkpoint information, such as the status of applications, and other tasks.
3. There were 222 BSDS requests (field QJSTBSDS), these include request to read and update records.

Some useful DB2 queries for processing accounting and statistics data

This chapter has some example DB2 queries using data in DB2 databases produced by Performance Reporter from SMF records.

Display which shared queues were used, with their attributes

DB2 query to display the shared queues used with their attributes

```
SELECT QMGR,BASENAME,QSGDISP,QTYPE,INDXTYPE,CFSTRUCNAME,COUNT(*)
FROM DRL.MQ_DAILY_QUEUE
GROUP BY QMGR,BASENAME,QSGDISP,QTYPE,INDXTYPE,CFSTRUCNAME
ORDER BY BASENAME,QSGDISP
```

Example output showing shared queues used with their attributes

```
QMGR  BASENAME              QSGDISP   QTYPE     INDXTYPE  CFSTRUCNAME   COUNT
----  --------------------  --------  --------  --------  ------------  -----
V52F  CCP                   Q_MGR     LOCAL     NONE                        2
V521  CPV521                Q_MGR     LOCAL     MSGID                      13
V521  CPV521                Q_MGR     LOCAL     MSGID                      14
V520  CSIM_COMMON_REPLY_Q   COPY      LOCAL     CORELID   APPLICATION1      2
MQ07  MQ08.001              UNKNOWN   REMOTE    NONE                        7
MQ07  MQ08.002              COPY      LOCAL     NONE      APPLICATION1      4
```

Display the queues which had I/O to a page set

Query to display the queues which had I/O to a page set

```
SELECT QMGR,QSGDISP,QUEUE,NBUFFPOOL,PAGESET,PUTPSN+PUT1PSN+GETPSN
FROM DRL.MQ_DAILY_QUEUE
WHERE PUTPSN+PUT1PSN+GETPSN> 0
```

Example output showing which queues which had I/O to a page set

```
QMGR  QSGDISP   QUEUE                    NBUFFPOOL      PAGESET      NUM I/O
----++--------++---------------------++-----------++-----------++-----------
V521  Q_MGR     TOV52F                            2            2            8
V521  Q_MGR     SYSTEM.CHANNEL.SYNCQ              1            1           12
V520  Q_MGR     SYSTEM.ADMIN.CHANNEL              3            3            4
V520  Q_MGR     SYSTEM.CLUSTER.REPOS              1            1           12
```

Display MQI verbs used by transaction by queue

Query to display MQI verbs used by transaction and queue

```
SELECT TRAN,QUEUE,'OPEN',OPENN,OPENET/OPENN,OPENCT/OPENN
FROM DRL.MQ_DAILY_QUEUE
WHERE OPENN > 0 AND TRAN =' '
UNION
SELECT TRAN,QUEUE,'CLOS',CLOSEN,CLOSEET/CLOSEN,CLOSECT/CLOSEN
FROM DRL.MQ_Daily_queue
WHERE CLOSEN > 0 AND TRAN =' '
UNION
SELECT TRAN,QUEUE,'PUT ',PUTN,PUTET/PUTN,PUTCT/PUTN
FROM DRL.MQ_Daily_queue
WHERE PUTN > 0 AND TRAN =' '
UNION
SELECT TRAN,QUEUE,'PUT1',PUT1N,PUT1ET/PUT1N ,PUT1CT/PUT1N
FROM DRL.MQ_Daily_queue
WHERE PUT1N > 0 AND TRAN =' '
UNION
SELECT TRAN,QUEUE,'GET ',GETN,GETET/GETN,GETCT/GETN
FROM DRL.MQ_Daily_queue
WHERE GETN > 0 AND TRAN =' '
UNION
SELECT TRAN,QUEUE,'GETV',VALIDGET,GETET/VALIDGET,GETCT/VALIDGET
FROM DRL.MQ_DAILY_QUEUE
WHERE VALIDGET > 0 AND TRAN =' '
UNION
SELECT TRAN,' ','COM ',WTASCMN,WTASCMET/WTASCMN,WTASCMCT/WTASCMN
FROM DRL.MQ_DAILY_TASK
WHERE WTASCMN > 0 AND TRAN =' '


ORDER BY TRAN,2
```

The calculations like PUTET/PUTN calculate the average response time in microseconds.

The line with GETV is for valid gets, those that returned a message.

Example output showing MQI verbs used by transaction and queue

```
TRAN      QUEUE          VERB        COUNT       AVG ET       AVG CT
--------  ------------  ----  -----------  -----------  -----------
CP17                    COM             2         4599           52
CP17      CP0000        CLOS            2           24           23
CP17      CP0000        GET             1          187          186
CP17      CP0000        GETV            1          187          186
CP17      CP0000        OPEN            2           93           92
CP17      CP0000        PUT             1          400          369
```

Where:

AVG ET	Is the average time for the call in microseconds
AVG CT	Is the CPU time used by the call in microseconds
GETV	Is the number of valid **MQGET** calls
GET	Is the total number of **MQGET** calls

Display where a queue is used

When a queue alias, queue remote, or dynamic queues are used, the name used when opening the queue is different from the actual queue used. By selecting the basename field, you can display the queue name used by the application.

Query to display the usage of Queue alias, queue remote and dynamic queues

```
SELECT BASENAME,JOBNAME,JOBTYPE,QTYPE,QUEUE
FROM DRL.MQ_DAILY_QUEUE
```

Output showing the usage of Queue alias, queue remote and dynamic queues

```
BASENAME              JOBNAME   JOBTYPE  QTYPE     QUEUE
--------------------++--------++-------++--------++-------
V52F                  PAICE4P   MVS      REMOTE    TOV52F
V52F                  V521CHIN  CHIN     LOCAL     V52F
```

Where:

Queue	Is the name of the queue used by the application
Basename	Is the actual queue used after any indirection

Display the length of time messages were on a queue

Query to display the message time on a queue

```
SELECT QUEUE,MAXTOQ,MINTOQ,TOTTOQ/VALIDGET,VALIDGET
FROM DRL.MQ_DAILY_QUEUE
WHERE VALIDGET > 0
```

Where TOTTOQ/VALIDGET is the total time on queue divided by the number of gets which returned messages, to give the average time on queue.

Output showing the message time on queue

```
CHANNEL   QUEUE                    MAXTOQ    MINTOQ   AVG TOQ  VALIDGET
--------  ---------------------  --------  --------  --------  --------
          CP0000                 4.9E+03   3.2E+03   4.1E+03         2
MQ1       SYSTEM.CHANNEL.SYNCQ   8.6E+09   3.2E+04   1.5E+06      5873
MQ2       SYSTEM.CHANNEL.SYNCQ   1.1E+06   9.3E+03   2.7E+04     11741
```

Where:

MAXTOQ	is the maximum time on queue in microseconds. This could potentially be a very large number. A message which was on a queue for a day would have a value of 8.6E+10
MINTOQ	is the minimum time on queue in microseconds
AVG TOQ	is the average time on queue in microseconds

The channel MQ1 was not active for a couple of hours. When the channel restarted it read its message from the SYSTEM.CHANNEL.SYNCQ queue. In this case the maximum time on queue represents 2.3 hours (8.6E+09 microseconds).

Display count of get specific and get first message

Query to display count of Get specific and get first messgage

```
SELECT QMGR,QUEUE,QTYPE,QSGDISP,QTYPE,GETA,GETS,GETBRWA,GETBRWS
FROM DRL.MQ_DAILY_QUEUE
```

Sample output from the query

```
QMGR  QUEUE                 INDXTYPE       GETA      GETS   GETBRWA   GETBRWS
----  --------------------  --------  --------  --------  --------  --------
V52F  SYSTEM.CHANNEL.SYNCQ  MSGID            0     23488         4         4
V52F  SERVER                NONE        108004         0         0         0
```

Where:

GETS is the number of get specific requests

GETBRWS is the number of get browse specific requests

GETA is the number of get any(first) requests

GETBRWA is the number of get browse any(first) requests

Display out of line log manager statistics

```
SELECT MQSERIES_SUB_ID,WAIT_COUNT_NO_BUF,
          READ_REQ_BUF,
          READ_REQ_ACTIVE,READ_REQ_ARCHIVE,READ_REQ_DELAYED,
          LOGBUF_PAGEDIN
FROM DRL.MQS_LOGMGR_T
WHERE READ_REQ_BUF+
         READ_REQ_ACTIVE+READ_REQ_ARCHIVE+READ_REQ_DELAYED+
         LOGBUF_PAGEDIN > 0
```

Example output showing the count of get specific and get first messages

```
MQSERIES          READ         READ         READ         READ
  SUB             REQ          REQ          REQ          REQ         LOGBUF
   ID             BUF        ACTIVE       ARCHIVE      DELAYED      PAGEDIN
--------  -----------  -----------  -----------  -----------  -----------
V52G                0          314            0            0            0
V52G                0          157            0            0            0
V52G                0          157            0            0            0
V52G                0          314            0            0            0
V520                0          147            0            0            0
```

This page intentionally left blank.

Sample C program for displaying statistics and accounting

A C program is provided as an executable and as a source file to display MQSeries statistics and accounting data. The program may be used as-is, but it is intended to be a starting point so you can tailor it to meet your requirements.

This progam and the header files provided with this SupportPac are not supported by IBM. But if you tell the author (PAICE@UK.IBM.COM) of any problems, improvements may be incorporated in any future updates - if there are any future updates.

Upload the MQSOURCE in binary to TSO and issue `Receive indsn(MQSOURCE)`

Some of the key members of this data set are

MQCSAMP The source of the C program which will read MQSeries SMF records and display information in many forms. It contains JCL to compile and linkedit it to create a module MQCSMF.

RUNCSMF This executes the C program to print MQSeries SMF data.

This program is provided to illustrate ways that the data can processed. The output has been designed primarily so that it fits in this report, rather than for functional value. For example if the reports were to print out channel name, channel qualifier, queue used, there would be very little space left to display other information such as number of bytes processed. In particular timestamps have been omitted from most reports.

Using the sample program

If you change the program beware of the following

1. APAR PQ43750 fixes some problems in the accounting information see "Required fixes" on page 10.
2. Some of the fields are 64 bit long, for example those containing time values. These can be processed using "long long" variables available in the OS/390 C compiler. On one of our MVS systems, calculations using long-long variables gave incorrect values. On other systems where the service included a 2000 PUT tape the calculations worked properly. By using floating point, instead of long-long, in calculations this problem can be circumvented.
3. Some numbers can be very large and will not format properly using integer arithmetic in printf. You should consider displaying the data in floating point, like 6.221E+04. This applies to bytes processed, and "time on queue" where some messages could be on a queue for a long period, and one day is 86400000000 microseconds. You could convert the data to other units such as MegaBytes processed instead of bytes processed, and seconds instead of microseconds.
4. You can use the facilities of the ICETOOL facility of DFSORT to do simple accumulation and reporting of maximum and minimum values, see *DFSORT R13 ICETOOL Mini-User Guide GC26-7140-01* for more information.

Execute the sample code

Example JCL to run the C program is in given below

```
//PAICEC2  JOB '1',MSGCLASS=H,MSGLEVEL=(0,0),COND=(0,LT)
//S1 EXEC PGM=MQCSMF
//STEPLIB DD DISP=SHR,DSN=MQM.LOAD
//SMFIN    DD DISP=SHR,DSN=MQM.STATS
//SYSPRINT DD SYSOUT=*,DCB=(LRECL=132,RECFM=F)
//SUMMARY  DD SYSOUT=*,DCB=(LRECL=133,RECFM=F,BLKSIZE=133)
//STATS    DD SYSOUT=*,DCB=(LRECL=133,RECFM=F,BLKSIZE=133)
//PUT      DD SYSOUT=*,DCB=(LRECL=133,RECFM=F,BLKSIZE=133)
//GET      DD SYSOUT=*,DCB=(LRECL=133,RECFM=F,BLKSIZE=133)
//DB2      DD SYSOUT=*,DCB=(LRECL=133,RECFM=F,BLKSIZE=133)
//CF       DD SYSOUT=*,DCB=(LRECL=133,RECFM=F,BLKSIZE=133)
//SCF      DD SYSOUT=*,DCB=(LRECL=133,RECFM=F,BLKSIZE=133)
//MM       DD SYSOUT=*,DCB=(LRECL=133,RECFM=F,BLKSIZE=133)
//BM       DD SYSOUT=*,DCB=(LRECL=133,RECFM=F,BLKSIZE=133)
//SDB2     DD SYSOUT=*,DCB=(LRECL=133,RECFM=F,BLKSIZE=133)
//THREAD   DD SYSOUT=*,DCB=(LRECL=133,RECFM=F,BLKSIZE=133)
//LOG      DD SYSOUT=*,DCB=(LRECL=133,RECFM=F,BLKSIZE=133)
```

The files referenced in the JCL are explained below. All of the files need to be defined, but they can be set to "DD DUMMY" if required.

SMFIN data set

The SMFIN data set is the SMF records which have been extracted from SMF using a job like that on page 83. This data set is typically Variable Blocked Spanned with a 32760 record length.

SYSPRINT contents

The records in this data set give notification of any major problems identified, as a buffer pool too small.

```
2000293 V52A Buffer pool 3 is too small make larger
2000293 V52A Log stats - make OUTBUFF larger.
2000293 V52A Archive logs read.
```

SUMMARY contents

The records in this data set give a summary of the the usage of MQI verbs acting on a queue, and where time was spent.

```
Jobname   et_ms %open %clos %get %put %put1 %inq %set %cpu %log %pset %other
V52FCHIN      0     0     0  100    0     0    0    0|  97    0     0     3
PAICE4P    3329     7     1   85    4     0    0    0|  19   76     0     5
```

Where

Jobname Is the job name

et_ms	Is the total time spent doing MQSeries calls against a queue in *milli*seconds
%open	The percentage of the time spent doing **MQOPEN** calls
%clos	The percentage of the time spent doing **MQCLOSE** calls
%get	The percentage of the time spent doing **MQGET** calls
%put	The percentage of the time spent doing **MQPUT** calls
%put1	The percentage of the time spent doing **MQPUT1** calls
%inq	The percentage of the time spent ding **MQINQ** calls
%set	The percentage of the time spent doing **MQSET** calls
%cpu	The percentage of the time using CPU
%log	The percentage of the time waiting for log I/O to complete
%pset	The percentage of the time waiting for page set I/O to complete
%other	This is calculated as 100 - (%log + %pset + %cpu).

The et_ms for V52FCHIN is 0 because the total time was less than 1 millisecond.

STATS contents

The records in this data set give an overview of the MQI verbs used and the average elapsed time and average CPU time user per call.

```
Jobname  Queue                          Verb   Number Avg et Avg CT
V52FCHIN                                Commit   5872   4358     49
V52FCHIN                                Other       6     63     62
V52FCHIN SERVER                         Open        1    103    102
V52FCHIN SERVER                         Put      5872    155    150
V52FCHIN SYSTEM.ADMIN.CHANNEL.EVENT     Put1        1   3254    262
V52FCHIN SYSTEM.CHANNEL.SYNCQ           Open        2     71     70
V52FCHIN SYSTEM.CHANNEL.SYNCQ           Close       1     28     27
V52FCHIN SYSTEM.CHANNEL.SYNCQ           Put      5872    117    109
V52FCHIN SYSTEM.CHANNEL.SYNCQ           Get      5874     94     92
V52FCHIN                                Commit   5871   3485     46
V52FCHIN                                Other       6     54     54
PAICE4P                                 Commit   2000   2143     28
PAICE4P  CPV521                         Open     1000     70     68
PAICE4P  CPV521                         Close    1000     26     25
PAICE4P  CPV521                         Get      2000   1431     93
PAICE4P  V52F                           Open     1000    195    187
PAICE4P  V52F                           Close    1000     19     19
PAICE4P  V52F                           Put      1000    154    149
```

Where

Jobname	Is the job name
Queue	Is the queue name, or blank for commit, backout and "Other".
Verb	The MQI verb used. "Other" is used for some internal verbs, and when data cannot be

collected, for example an when an **MQOPEN** fails.

Number The number of times this verb was used

Avg et The average elapsed time per call in microseconds

Avg CT The average cpu time per call in microseconds

PUT contents

The records in this data set give an overview of **MQPUT** and **MQPUT1** verbs.

```
Jobname  Queue                          Valid Put      Put     Put1 Put_Bytes   PutMax   PutMin
V52FCHIN SERVER                              5872     5872        0  5872000     1000     1000
V52FCHIN SYSTEM.ADMIN.CHANNEL.EVENT             1        0        1      124      124      124
V52FCHIN SYSTEM.CHANNEL.SYNCQ                5872     5872        0  2513216      428      428
PAICE4P  V52F                                1000     1000        0  1000000     1000     1000
```

Where

Jobname Is the jobname

Queue Is the queue name used

Valid Put Is the number of valid **MQPUT** and **MQPUT1** requests

Put Is the number of **MQPUT** requests

Put1 Is the number of **MQPUT1** requests

Put_Bytes Is the number of bytes put to the queue.

Note: This is potentially a very large number so this may not display properly. You may decide to modify this to be a floating point number or use MegaByte units.

PutMax Is the maximum message size in bytes

PutMin Is the minimum message size in bytes.

GET contents

The records in this data set give an overview of the **MQGET** verb.

There are entries for multiple channels and applications. There is a jobname/queue record for each. If the channel name was specified it would be clearer.

```
Jobname  Queue                            Get  ValidGet   Bytes    MaxGet   MinGet    MaxTOQ   MinTOQ   AvgTOQ
V52FCHIN SYSTEM.CHANNEL.SYNCQ             5874     5873  2513644      428      428   8.6E+09  3.2E+04  1.5E+06
V52FCHIN SYSTEM.CHANNEL.SYNCQ            11743    11741  5166052      452      428   1.1E+06  9.3E+03  2.7E+04
V52FCHIN V521                            17616    11744  8385216     1428        0   1.7E+04  2.7E+03  4.0E+03
V52FCHIN SYSTEM.CHANNEL.SYNCQ                6        3     1284      428      428   1.3E+09  1.3E+09  4.2E+08
V52FCHIN SYSTEM.CLUSTER.TRANSMIT.QUEUE      17       16    34008     2764      780   0.0E+00  0.0E+00  0.0E+00
V52FCHIN SYSTEM.CHANNEL.SYNCQ                6        3     1284      428      428   1.3E+09  1.3E+09  8.2E+08
V52FCHIN SYSTEM.CLUSTER.TRANSMIT.QUEUE       3        2     5528     2764     2764   0.0E+00  0.0E+00  0.0E+00
V52FCHIN SYSTEM.CHANNEL.SYNCQ                2        1      428      428      428   8.6E+09  3.1E+04  1.7E+05
PAICE4P  CPV521                           2000     2000  1000000     1000        0   2.1E+05  6.5E+03  4.1E+04
```

Where

Jobname	Is the jobname
Queue	Is the queue name used
Valid Get	Is the number of valid gets. See "Required fixes" on page 10.
Get	Is the number of **MQGET** requests
Get_Bytes	Is the number of bytes got from the queue. Note this is potentially a very large number so this may not display properly.
Maxget	Is the maximum message size in bytes
Minget	Is the minimum message size in bytes
MaxTOQ	Is the maximum time a message was on the queue in microseconds
MinTOQ	Is the minimum time a message was on the queue in microseconds
AvgTOQ	Is the average time a message was on the queue in microseconds.

The time on queue is the difference in time from when the message was put onto the queue to the time it was got. This can be a very large number so it is displayed in floating point format.

DB2 contents

The records in this data set are for the time a task spent in DB2.

```
Jobname      Count Avg_ET_T  Avg_ET_S Max_Ti_T Max_Ti_S
V521CHIN         5     3179      3083     3344     3241
V521CHIN         5     3366      3260     3571     3462
```

Where

Jobname	Is the jobname
Count	Is the count of requests
Avg_ET_T	Is the average wait time for the task in microseconds
Avg_ET_S	Is the average wait time for the server in microseconds
Max_Ti_T	Is the maximum wait time for the task in microseconds
Max_Ti_S	Is the maximum wait time for the server in microseconds.

For information on task time and server time, see "Statistics on MQSeries' use of DB2" on page 39

CF contents

The records in this data set are for time spent processing Coupling Facility requests by jobname.

See "Coupling Facility statistics" on page 47 for what the terms mean.

```
Jobname  E_calls E_redrive Avg_E_time M_calls M_redrive Avg_M_time
V520CHIN      32         0        51       16         0        288
V5201       4667         0        43     1333         0         55
```

Where

Jobname Is the jobname

E_calls Is the count of IXLLSTE requests to the Coupling Facility

E_redrive Is the number of redrives for IXLLSTE

Avg_E_time Is the average time for the IXLLSTE in microseconds

M_calls Is the count if IXLLSTM requests to the Coupling Facility

M_redrive Is the number of redrives for IXLLSTM

Avg_M_time Is the average time for the IXLLSTM in microseconds.

SCF contents

The records in this data set are for the time spent processing Coupling Facility requests summarized by Coupling Facility strucure name.

```
Date    QMGR CFN CFname       |    Num_E  Avg_E_T %Redrive    Num_M  Avg_M_T %Redrive Num_full
2000293 V52D   0 CSQ_ADMIN         11914       32       0        0        0       0        0
2000293 V52D   1 APPLICATION1      35082       44       0    17531      156       0        0
2000293 V52D   0 CSQ_ADMIN         74621       33       0        0        0       0        0
2000293 V52D   1 APPLICATION1     223852       43       0   111937      156       0        0
```

Where

Date Is the date in yyyyddd format

QMGR Is the queue manager name

CFN Is the Coupling Facility number

CFname Is the name of the Coupling Facility Structure

Num_E Is the count if IXLLSTE requests

Avg_E_T Is the average time for the IXLLSTM in microseconds

%Redrive Is the number of redrives for IXLLSTE

Num_M Is the count if IXLLSTM requests

Avg_M_T Is the average time for the IXLLSTM in microseconds

%Redrive Is the number of redrives for IXLLSTM

Num_full Is the number of times the structure was full.

MM contents

The records in this data set are for the message manager statistics

```
Date        Open    Close      Get      Put     Put1      Inq     Inql      Set
2000293       42        0    23193     5637    11914        0        0        0
2000293        0        0        5        0        0        0        0        0
```

BM contents

The records in this data set are for the buffer manager statistics.

```
Date... QMGR BP  NumBuff %now %low   dwt    dmc    stl  stla   sos
2000293 V52D  0    50000    0  100     0      0      0     0      0
2000293 V52D  1    99000    0  100     0      0      0     0      0
2000293 V52D  2    99000    0  100     0      0      0     0      0
2000293 V52D  3    99000    0  100     0      0      0     0      0
```

See "Buffer manager statistics" on page 53 for information on the meaning of the columns.

SDB2 contents

The records in this data set are for the MQSeries usage of DB2 statistics

```
Date    Time         QMGR Max_Depth Num_deadlock
2000308 00:30:00.40 V52G         4            0
2000308 01:00:00.00 V52G         1            0
2000308 01:30:00.00 V52G         0            0
```

See "Statistics on MQSeries' use of DB2" on page 40 for information on the meaning of the columns.

THREAD contents

The records in this data set show the job name, jobtype and channel name of active MQSeries threads.

```
Jobname  type        Channel             channel qualifier
V52F     RRS
V52FCHIN Channel
V52FCHIN Channel     TO.V52G              WINMVS2A(2163)
V52FCHIN Channel     V521.TO.V52F         9.20.101.14
V52FCHIN Channel     V52F.V521            WINMVS25(2171)
V52FCHIN Channel
V52FCHIN Channel     TO.V520              WINMVS25(2170)
V52F     IMS Control
V52FCHIN Channel     V521.TO.V52F         9.20.101.14
V52FCHIN Channel     V52F.V521            WINMVS25(2171)
V52FCHIN Channel
V52FCHIN Channel     TO.V520              WINMVS25(2170)
V52FCHIN Channel     TO.V52G              WINMVS2A(2163)
```

See "Meaning of the channel names" on page 100 for information on the meaning of the columns.

LOG contents

The records in this data set are for the log manager statistics

```
Date    QMGR  wr_wait wr_nwait wr_force Wait_Buf read_buf read_act read_arc  r_delay N_CheckP Num I/O Num_CI_W   paging
2000293 V52F        0   273944      157        0        0        0        0        0        0   82188    82188        0
2000293 V52D        0        0        0        0        0        0        0        0        0       0        0        0
```

Supplied programs to print out the SMF records.

Three program executables (no source) are available with this SupportPac.

MQ1150 This prints out MQSeries statistics

MQ116S This prints out the new task and queue accounting records

MQ1160 This prints out the accounting information which was also available in earlier releases.

These programs are suitable for displaying the contents of a few records, but are impractical for processing a large number of records as they can generate a large amount of output.

You should use products like Performance Reporter® or SAS® for long term processing of the statistics or accounting data.

To install these on OS/390, upload the file MQLOAD to OS/390 in binary, for example using FTP, and use the TSO command "Receive indsn(xxx.xxx)" where xxx.xxx is the data set name on OS/390

To extract the SMF records from the SMF datasets you can use a job similar to the one below.

Sample job to extract the MQSeries SMF records from the SMF data sets.

You can specify a time range in hhmm format using START() and END().

```
//RUNPROG  JOB 1,CLASS=A
//*
//* Extract the records from the SMF database
//*
//SMFDUMP  EXEC PGM=IFASMFDP,REGION=0M
//DUMPOUT  DD   DSN=PAICE.MQSMF,DISP=(NEW,CATLG),SPACE=(CYL,(10),RLSE)
//SYSPRINT DD   SYSOUT=*
//SYSIN    DD   *
  INDD(DUMPIN,OPTIONS(DUMP))
  START(0000)
  END(2359)
  OUTDD(DUMPOUT,TYPE(115,116))
/*
//DUMPIN  DD   DSN=SYS1.MAN1,DISP=SHR,AMP=('BUFSP=65536')
```

Example JCL to run the supplied programs to print the SMF records is given below

```
//PAICESM1 JOB NOTIFY=PAICE,MSGCLASS=H,MSGLEVEL=(0,0)
//S1 EXEC PGM=CSQW1150
//STEPLIB  DD DISP=SHR,DSN=MQM.MQLOAD
//SYSPRINT DD SYSOUT=*
//SMFIN    DD DISP=SHR,DSN=PAICE.MQSMF

//PAICESM2 JOB NOTIFY=PAICE,MSGCLASS=H,MSGLEVEL=(0,0)
//S1 EXEC PGM=CSQW1160
//STEPLIB  DD DISP=SHR,DSN=MQM.MQLOAD
//SYSPRINT DD SYSOUT=*
//SMFIN    DD DISP=SHR,DSN=PAICE.MQSMF

//PAICESM3 JOB NOTIFY=PAICE,MSGCLASS=H,MSGLEVEL=(0,0)
//S1 EXEC PGM=CSQW116S
//STEPLIB  DD DISP=SHR,DSN=MQM.MQLOAD
//SYSPRINT DD SYSOUT=*
//SMFIN    DD DISP=SHR,DSN=PAICE.MQSMF
```

Example output and description of the MQSeries statistics printout

Example print out of MQSeries statistics - part 1

```
MVS:MV25 MQ_Subsys:V520 Date 2000297  Time 16:40:34.25
Storage manager : QSST
  fix. pools created:         1, Deallocated:         4
  fix. segments freed:         0, expanded:         0, contracted         0
  var. pools created:         3, Deallocated:         3
  var. segments freed:         4, expanded:         4, contracted         0
  # Getmains:      2003, # Freemains:      2007, # non-zero Rcs         0
  # SOS bit:         0, # contractions         0, # Abends         0
Log manager : QJST
  Write_wait          0 Write_Nowait          0 Write_Force           0 WTB          0
  Read_stor           0 Read_Active           0 Read_archive          0 TVC  0
  BSDS_reqs           0 CIs_created           0 BFWR                  0 ALR          0
  ALW                 0 CIs_offload           0 Checkpts              0
  WUR                 0 LAMA                  0 LAMS                  0
  Write_susp          0 Write_Reqs            0 CI_Writes             0
  Write_serl          0 Write_Thrsh           0 Buff_Pagein           0
```

The header *MVS:MV25 MQ_Subsys:V520 Date 2000297 Time 16:40:34.25* is taken from the SMF header.

The *Storage manager : QSST* section is only of interest to IBM.

For section Log manager: QJST see Log manager statistics on page 61. The fields are in the same order as **Table 10** on page 61, and the fields have been given more meaningful names.

Example print out of MQSeries statistics - part 2

```
MVS:MV25 MQ_Subsys:V520 Date 2000297  Time 16:40:34.25
Message manager : QMST
  #MQOPEN      6001 #MQCLOSE 2001 #MQGET  9051 #MQPUT  4000 #MQPUT1  0 #MQINQ  0 #MQSET      0
Data manager : QIST
  #Create         0 #Put          0 #Delete 0 #Get    7000 #Locate   18003    #stgclass      0
Buffer manager : QPST
  00 #buff     50000 #low     49967 #now      49967 #getp     13944 #getn         0
  00 Rio           0 STW      13944 TPW           0 WIO           0 IMW           0
  00 DMC           0 STL          0 STLA          0 SOS           0
  01 #buff     99000 #low     98664 #now      98664 #getp      8334 #getn       672
  01 Rio           0 STW       6018 TPW           0 WIO           0 IMW           0
  01 DMC           0 STL          0 STLA          0 SOS           0
  02 #buff     99000 #low     94876 #now      94876 #getp     16176 #getn      1227
  02 Rio           0 STW      11725 TPW           0 WIO           0 IMW           0
  02 DMC           0 STL          0 STLA          0 SOS           0
  03 #buff     99000 #low     99000 #now          0 #getp         0 #getn         0
  03 Rio           0 STW          0 TPW           0 WIO           0 IMW           0
  03 DMC           0 STL          0 STLA          0 SOS           0
Lock manager : QLST
  #lock gets    105821, already held      2998 releases     32873
DB2 manager : Q5ST
  Tasks  : Servers   10 Active   11 Conns    0 discs    0     High    1 Abend         0 Requeue   0
  Number of deadlock conditions :    0
  Reads  : #:  304 Task Avg m/s :    2 Task Max m/s :    8 DB2 Avg m/s :    2  DB2 Max m/s :    8
  Writes : #:   24 Task Avg m/s :    9 Task Max m/s :   68 DB2 Avg m/s :    8  DB2 Max m/s :   68
  Lists  : #:   60 Task Avg m/s :   20 Task Max m/s :   38 DB2 Avg m/s :   20  DB2 Max m/s :   38
  Deletes: #:   24 Task Avg m/s :   28 Task Max m/s :  104 DB2 Avg m/s :   27  DB2 Max m/s :  104
  SCS Selects: #:   21 Task Avg m/s :    7 Task Max m/s :   92 DB2 avg m/s :    7  DB2 Max m/s :   91
  SCS Inserts: #:   71 Task Avg m/s :   20 Task Max m/s :  259 DB2 avg m/s :   20  DB2 Max m/s :  259
  SCS Updates: #:   71 Task Avg m/s :   13 Task Max m/s :   31 DB2 avg m/s :   13  DB2 Max m/s :   31
CF manager : QEST
  Structure #:        0, Name      CSQ_ADMIN   , Structure-fulls   0
  Single           1000, Avg time uS         53, Single   retries  0
  Multiple            0, Avg time uS          0, Multiple retries  0
  Max used entries        344, Max used elements       666
  Structure #:        1, Name      APPLICATION1, Structure-fulls   0
  Single           1000, Avg time uS         73, Single   retries  0
  Multiple         1000, Avg time uS        211, Multiple retries  0
  Max used entries      1313, Max used elements      7192
```

The header *MVS:MV25 MQ_Subsys:V520 Date 2000297 Time 16:40:34.25* is taken from the SMF header.

For section *Message manager : QMST* see "Message manager statistics" on page 51. The fields are in the same order as in and the fields have been given more meaningful names.

For section *Data manager : QIST* see "Data manager statistics" on page 52. The fields are in the same order as in **Table 8** on page 52 and the fields have been given more meaningful names.

For section *Buffer manager : QPST* see "Buffer manager statistics" on page 53 The fields are in the same order as in **Table 9** on page 53 and the fields have been given more meaningful names, and the first field in the line is the buffer pool number.

The *Lock manager : QLST* statistics are only of interest to IBM.

For section *DB2 manager : Q5ST* see "Statistics on MQSeries' use of DB2" on page 39. The fields are taken from **Table 6** on page 43 . Rows which have all zero values are omitted, for example there was no Shared Sync Queue activity so all of the SSK* counters are zero, and so the rows are omitted.

For section *CF manager : QEST* see "Coupling Facility statistics" on page 47. There are 0 or more subsections taken depending on the number of Coupling Facility structures which have had any activity The fields are taken from **Table 6** on page 48. Row which have all zero values are omitted.

Example output and description of the MQSeries accounting printout

Example print out of the old MQSeries accounting records

```
 MVS:MV25 MQ_Subsys:V520 Date 2000297  Time 16:39:42.04
 QWHS: Job IMSV     Job_Userid IMSCREG  Tran_Userid IMSCREG  Connection IMS  Thread 4042322161
CSQQTRMN
 QWHC: Accounting token : , Network identifier : IMSV
 QMAC: CPU 00000318 MQPUT 0-99        0 100-999        0 1000-9999        0 >9999        0
 QMAC:              MQGET 0-99        0 100-999     2002 1000-9999        0 >9999        0
```

The header *MVS:MV25 MQ_Subsys:V520 Date 2000297 Time 16:39:42.04* is taken from the SMF header.

The fields are described in "Understanding and using the accounting data available in and before Version 5.2" on page 20. The CPU field is the CPU time used in milliseconds.

Example output and description of the new MQSeries accounting printout

The statistics are discussed in "Understanding and using the new accounting data" on page 23, and the records have information on

- Task identification, described in "Task identification (WTID)" on page 98. See page 86 for example printout.
- Task accounting data, described in "Task related information (WTAS)" on page 96. See page 86 for example printout.
- Queue accounting data, described in "Queue records (WQ)" on page 93. See page 86 for example printout.

Example print out of the new MQSeries accounting records - task identification

```
MVS:MV25 ,MQSeries sys:V520 ,Jobname:PAICEIMS ,Userid:PAICE1 ,Date 2000297  ,Time 16:37:36.61
          ====> New task record found     <==========
== Thread type.............> MVS/TSO
== Connection name........> PAICEIMS
== Operator id............> PAICE1
== User identifier........> PAICE1
== Channel name...........>
== Chl Connection.........>
== Correlator Id..........>
== Correlator Id.....(HEX)> 4040404040404040404040
== Context token..........>
== Context token.....(HEX)> 00000000000000000000000000000000
== NID....................>
== NID................(HEX)> 40404040404040400000000000000000
== accounting token.......>
== accounting token..(HEX)> 00000000000000000000000000000000000000000000000000
== UOW identifier.........>                          ¸O|·x□
== UOW identifier....(HEX)> 40404040404040404040404040404040B4D611B3A73D0001
```

The header *MVS:MV25 MQ_Subsys:V520 Date 2000297 ,Time 16:37:36.61* is taken from the SMF header.

Example print out of the new MQSeries accounting records - task accounting

```
== Task Token : 23/10/2000, 15:36:47.79, 7f14b7b8,   2741f038
== Interval : START 23/10/2000, 15:36:47.79
== Interval : END   23/10/2000, 15:37:56.61
== Commit  : Count       2000 ,avg elapse        130 ,avg CPU         34
== Suspend : Count       2000 ,avg elapse        100
== Pages   : New      20878 ,Old       1672
== Task Token : 23/10/2000, 15:36:47.79, 7f14b7b8,   2741f038
```

The row with == *Commit* gives the number of commit requests, and the average elapsed time for the commit requests in microseconds and the average CPU time used on the users TCB in microseconds of CPU.

Example print out of the new MQSeries accounting records - task accounting

```
Open name CP0000                                        , Base name CP0000
First opened  : 23/10/2000, 15:36:47.80
Last closed   : 23/10/2000, 15:37:56.60
Pageset identifier:          2, Bufferpool:          2
Current opens :          0 , Total Requests :       6000
GETs : Valid       4000 Max size         70 Min Size   0 Total bytes   00000000000222E0
GETs : Dest-S          0, Dest-G       4000, Brow-S          0 Brow-G          0
Gets : Maximum latency 00000  0:00:00.940
Gets : Minimum latency 00000  0:00:00.002
GETs : Average latency 00000  0:00:00.003
Generated messages :          0
MQCall            N         ET         CT     Susp   LOGW  PSET Epages skip expire
Open           1000         62         60
Close          1000         26         25
Get            4000         63         62        0     0    0     2218   0     0
```

The row with *GETs : Valid* gives the number of gets which returned a message(valid gets) the maximum and minimum message sizes in bytes, and the total number of bytes processed as an 8 character hex number. It is displayed in hex because this number could be very large and normal 31 bit arithmetic may not work.

The row with *GETs : Dest-S* gives

- the number of destructive gets where a message id or correlation id was specified (Dest-S)
- the number of destructive gets where a message id or correlation id was not specified (Dest-G)
- the number of browse requests where a message id or correlation id was specified (Brow-S)

- the number of browse requests where a message id or correlation id was not specified (Brow-G)

The rows giving latency times give the value in yyddd hh:mm:ss.ttt format.

The row with *MQCall N ET CT Susp LOGW PSET Epages skip expire* is a heading for information about the verbs following. The columns have the following meaning

MQCall	**MQOPEN**, **MQPUT** etc
N	Number of times the verb was issued
ET	Average elapsed time in microseconds
CT	Average CPU time used in microseconds
Susp	Some verbs can be suspended, this gives the average time in microseconds if the verb was suspended
LOGW	This is the average time in microseconds the verb was suspended waiting for log I/O
PSET	The average time in microseconds waiting for page set I/O in microseconds (when the page set was not page set 0)
Epages	The number of empty pages scanned during a get
skip	the number of messages that were skipped when looking for the required message
expire	the number of expired messages that were skipped when looking for the required message

This page intentionally left blank.

Appendix A. Overall layout of MQSeries SMF records

SMF record layout

The standard layout for SMF records involves three parts:

Part of record	What it is used for
SMF header	Provides format, identification, and time and date information about the record itself.
Self-defining section	Defines the location and size of the individual data records within the SMF record.
Data records	The actual data from MQSeries that you want to analyze.

For more information about SMF record formats, see the *MVS System Management Facilities (SMF)* manual.

SMF record header description

The SMF header is the same for subtypes 115 and 116 and the layout is given in **Table 11**. The fields are labelled SM116* to match the description of other SMF records in the *MVS System Management Facilities (SMF)* manual.

Note: The length of the record subtype(SM116ST) is now two bytes instead of 1 byte, to conform with the standard SMF record layout.

Table 11 SMF record header description

Offsets		Type	Len	Name	Description
Dec	Hex				
0	(0)	Structure	28	SM116	SMF record header.
0	(0)	Unsigned	2	SM116LEN	SMF record length.
2	(2)		2		Reserved.
4	(4)	Unsigned	1	SM116FLG	System indicator.
5	(5)	Unsigned	1	SM116RTY	Record type. The SMF record type, for MQSeries accounting records this is always 116 (X'74'). For MQSeries statistics this is 115 (X'73').
6	(6)	Unsigned	4	SM116TME	Time when SMF moved record.
10	(A)	Unsigned	4	SM116DTE	Date when SMF moved record.
14	(E)	Character	4	SM116SID	OS/390 subsystem ID. Defines the OS/390 subsystem on which the records were collected.
18	(12)	Character	4	SM116SSI	MQSeries subsystem ID.
22	(16)	Unsigned	2	SM116STY	Record subtype.
24	(18)		4	SM116SEQ	Reserved.
28	(1C)	Character	0	SM116END	End of SMF header and start of self-defining section.

Processing accounting records (SMF type 116)

Each SMF 116 record has a subtype, field SM116STY. The subtypes used are

Subtype	Description
0	Message manager accounting. This is available in Version 5.2 and earlier, and is described in "Understanding and using the accounting data available in and before Version 5.2" on page 20
1	Queue level accounting. This was introduced in Version 5.2. It has data on task identification, task related statistics and queue records. It is described in "Understanding and using the new accounting data" on page 23.
2	Queue level accounting extension records. This was introduced in Version 5.2. It is used when there are too many queue records to fix into a subtype 1 record. It has sections on task identification and queue records. It is described in "Understanding and using the new accounting data" on page 23.

Processing statistics records (SMF type 115)

The SMF header is identical in structure to the SMF type 116 records, and these fields(SM116*) can be used to access the fields.

Each SMF 115 record has a subtype. The subtypes used are

Subtype	Description
1	Log manager statistics. These are described in "Log manager statistics".
2	Message manager, data manager, buffer manager, DB2 manager and Coupling Facility manager These are described in "Message manager statistics", "Data manager statistics", "Buffer manager statistics", "Statistics on MQSeries' use of DB2" and "Coupling Facility statistics".

Self-defining sections

A self-defining section of an SMF record tells you where to find the different records, how long they are, and how many times that type of record is repeated. The self-defining sections follow the header, at a fixed offset from the start of the SMF record.

The table below summarizes the offsets from the start of the SMF record header.

Table 12. Offsets to self-defining sections

Offsets are from the start of the SMF record and are fixed for each type of accounting source.			
Source of accounting data	Offsets		See...
	Dec	Hex	
Accounting SMF type 116, subtype 0			
Common MQSeries SMF header	28	(X'1C')	Table 1 on page 20
Message manager	44	(X'2C')	Table 2 on page 20
Accounting SMF type 116, subtype 1			
Common MQSeries SMF header	28	(X'1C')	SMF record header description on page 89
Task identification	36	(X'24')	Task identification(WTID) on page 98
Task accounting	44	(X'2C')	"Task related information (WTAS)" on page96
Queue records	52	(X'34')	Queue records (WQ)Present if QWHSNSDA is larger than 3. See Table 17 on page 101

			Note that this section may not be present, if the application did not process any queues in the time period.
Accounting SMF type 116, subtype 2			
Common MQSeries SMF header	28	(X'1C')	Structure of the MQSeries SMF header QHWS on page 101
Task identification	36	(X'24')	Task related information (WTAS) on page 96
			There is no task accounting section in the subtype 2 record
Queue records	44	(X'2C')	Queue records (WQ) on page 93
Statistics SMF type 115, subtype 1			
Storage manager	100	(X'64')	This is of use only to IBM
Log manager	116	(X'74')	Log manager statistics on page 61
Statistics SMF type 115, subtype 2			
Message manager	36	(X'24')	Message manager statistics on page 51
Data manager	44	(X'2C')	Data manager statistics on page 52
Buffer manager	52	(X'34')	Buffer manager statistics on page 53
Lock manager	60	(X'3C')	This is of use only to IBM
DB2 manager	68	(X'44')	Statistics on MQSeries' use of DB2 on page 39
Coupling Facility manager	76	(X'4C')	Coupling Facility statistics on page 47
Note: Other self-defining sections refer to data for IBM use only.			

Each self-defining record is two fullwords long and has this format: `ssssssssllllnnnn`

where:

ssssssss Fullword containing the offset from start of the SMF record.

llll Halfword giving the length of this data record.

nnnn Halfword giving the number of data records in this SMF record.

Note: Always use offsets in the self-defining sections rather than the absolute position in SMF record to locate the accounting records, because if the length of a section or the number of sections change the absolute position in the SMF record will change.

Table 13. Structure of the Common MQSeries SMF header record QWHS

This structure is the same as Structure of the MQSeries SMF header QHWS101except the fields, such as connection name, refer to the task that creates the SMF records, and not to the application.					
Offsets		**Type**	**Len**	**Name**	**Description**
Dec	**Hex**				
0	***(0)***	***Structure***	***128***	***QWHS***	
0	(0)		6		Reserved.
6	(6)	Character	1	QWHSNSDA	Number of self defining sections in the SMF records.
7	(7)		5		Reserved.
12	(0C)	Character	4	QWHSSSID	Subsystem name.

Appendix B: Detail layout of MQSeries accounting and statistics records

Queue records (WQ)

This member is defined in CSQDWQ.

This data is present in SMF 116 subtype 1 records, (if the number of self defining sections is greater than 3. See Structure of the MQSeries SMF header QHWS on page 101) and in SMF 116 subtype 2 records. See Self-defining sections on page 90.

Offsets					
Dec	Hex	Type	Len	Name	Description
0	0	Structure	576	WQSTAT	
0	0	Signed	2	WQID	Control block hex ID
2	(2)	Signed	2	WQLL	Length of the block
4	(4)	Character	4	WQEYE	Eye catcher (WQST)
8	(8)	Signed	4	WQVER	Version number
12	(C)	ADDRESS	4	WQNEXT	Reserved
16	(10)	Character	16	CORREL	Correlator to tie block to owning WTAS
32	(20)	Character	48	OBJNAME	Object name as opened
80	(50)	Character	48	BASENAME	Base name or generate name if applicable
128	(80)	Character	8	OPENTIME	Time queue opened (this is the first time if data is accumulated)
136	(88)	Character	8	CLOSTIME	Time the queue was closed (this is the last time if data is accumulated)

Object information

144	(90)	Signed	4	QTYPE	Queue type (for example, local)
148	(94)	Signed	4	INDXTYPE	Index type of queue
152	(98)	Signed	4	QSGDISP	QSGDISP (for example, SHARED or GROUP)

MQOPEN

156	(9C)	Character	4	OPENEYE	Eye catcher (OPEN)
160	(A0)	Character	8	OPENET	Total elapsed time for **MQOPEN** processing
168	(A8)	Character	8	OPENCT	Total amount of CPU time processing **MQOPEN** calls
176	(B0)	Unsigned	4	OPENN	Number of **MQOPEN** calls

MQCLOSE

180	(B4)	Character	4	CLOSEEYE	Eye catcher (CLOS)
184	(B8)	Character	8	CLOSEET	Total elapsed time for **MQCLOSE** processing
192	(C0)	Character	8	CLOSECT	Total CPU times used for **MQCLOSE** processing
200	(C8)	Unsigned	4	CLOSEN	Number of **MQCLOSE** calls

MQGET

204	(CC)	Character	4	GETEYE	Eye catcher (GET)
208	(D0)	Character	8	GETET	Elapsed time processing **MQGET** calls
216	(D8)	Character	8	GETCT	CPU times used processing **MQGET** calls

224	(E0)	Unsigned	4	GETN	Total number of **MQGET** calls
228	(E4)	Unsigned	4	GETBRWA	Number of **MQGET** browses (any)
232	(E8)	Unsigned	4	GETBRWS	Number of **MQGET** browses (specific)
236	(EC)	Unsigned	4	GETA	Number of **MQGET** calls (any)
240	(F0)	Unsigned	4	GETS	Number of **MQGET** calls (specific)
244	(F4)	Unsigned	4	GETERR	Number of unaccountable **MQGET**s
248	(F8)	Character	8	GETJWET	Elapsed time waiting for a journal write to complete This is for getting persistent messages out of syncpoint.
256	(100)	Unsigned	4	GETJWN	Number of journal write requests This is for getting persistent messages out of syncpoint.
260	(104)	Character	8	GETPSET	Elapsed time waiting for a read from a page set
268	(10C)	Unsigned	4	GETPSN	Number of reads from a page set
272	(110)	Character	8	GETSUSET	Total suspend time for **MQGET** calls
280	(118)	Unsigned	4	GETSUSN	Number of times suspended
284	(11C)	Unsigned	4	GETEPAGE	Number of empty pages skipped over when doing an **MQGET**
288	(120)	Unsigned	4	GETSMSG	Number of messages skipped when doing an **MQGET**, either by *MsgId* or *CorrelId*
292	(124)	Unsigned	4	GETEXMSG	Number of expired messages processed (this causes an increase in time because the event messages need to be produced)

MQPUT

296	(128)	Character	4	PUTEYE	Eye catcher (PUT)
300	(12C)	Character	8	PUTET	Total elapsed time for the **MQPUT** calls
308	(134)	Character	8	PUTCT	CPU time used during **MQPUT** processing
316	(13C)	Unsigned	4	PUTN	Number of **MQPUT** requests
320	(140)	Character	8	PUTJWET	Elapsed time waiting for a journal write request. This is for putting persistent messages out of syncpoint.
328	(148)	Unsigned	4	PUTJWN	Number of journal write requests. This is for putting persistent messages out of syncpoint.
332	(14C)	Character	8	PUTSUSET	Elapsed time the task was suspended for
340	(154)	Unsigned	4	PUTSUSN	Number of times suspended
344	(158)	Character	8	PUTPSET	Time taken to read from a page set for **MQPUT**
352	(160)	Signed	4	PUTPSN	Number of page set put requests

MQPUT1

356	(164)	Character	4	PUT1EYE	Eye catcher (PUT1)
360	(168)	Character	8	PUT1ET	Total elapsed time for the **MQPUT1** calls
368	(170)	Character	8	PUT1CT	CPU time used during **MQPUT1** processing
376	(178)	Unsigned	4	PUT1N	Number of **MQPUT1** requests
380	(17C)	Character	8	PUT1JWET	Elapsed time waiting for a journal write request. This is for putting persistent messages out of syncpoint.
388	(184)	Unsigned	4	PUT1JWN	Number of journal write requests. This is for putting persistent messages out of syncpoint.

392	(188)	Character	8	PUT1SUSET	Elapsed time the task was suspended
400	(190)	Unsigned	4	PUT1SUSN	Number of times suspended
404	(194)	Character	8	PUT1PSET	Time taken to read from a page set for **MQPUT1**
412	(19C)	Signed	4	PUT1PSN	Number of page set **MQPUT1** requests

MQINQ

416	(1A0)	Character	4	INQEYE	Eye catcher (INQ)
420	(1A4)	Character	8	INQET	Total elapsed time for the **MQINQ** calls
428	(1AC)	Character	8	INQCT	CPU time used during **MQINQ** processing
436	(1B4)	Unsigned	4	INQN	Number of **MQINQ** requests
MQSET					
440	(1B8)	Character	4	SETEYE	Eye catcher (SET)
444	(1BC)	Character	8	SETET	Total elapsed time for the **MQSET** calls
452	(1C4)	Character	8	SETCT	CPU time used during **MQSET** processing
460	(1CC)	Unsigned	4	SETN	Number of **MQSET** requests
464	(1D0)	Character	8	SETJWET	Elapsed time waiting for journal write requests
472	(1D8)	Unsigned	4	SETJWN	Number of journal write requests

Other statistics

476	(1DC)	Unsigned	4	NPS	Page set number
480	(1E0)	Character	12	CFSTRUCNAME	Name of CF structure
492	(1EC)	Unsigned	4	NBUFFPOOL	Buffer pool number
Putbytes/validput = average message size put. Getbytes/validget = average size of message got. Failed put and get count in the total above, but only those successful calls get counted below					
496	(1F0)	Character	8	PUTBYTES	Total number of bytes put successfully
504	(1F8)	Character	8	GETBYTES	Total number of bytes got successfully
512	(200)	Unsigned	4	VALIDPUT	Number of **MQPUT**s writing data
516	(204)	Unsigned	4	VALIDGET	Number of **MQGET**s with data
520	(208)	Unsigned	4	NGEN	Number of messages generated (including COA, COD, event, and expiry messages)
524	(20C)	Signed	4	GETMAXMS	Get maximum messages size
528	(210)	Signed	4	GETMINMS	Get minimum messages size
532	(214)	Signed	4	PUTMAXMS	Put maximum messages size
536	(218)	Signed	4	PUTMINMS	Put minimum messages size
540	(21C)	Character	8	MAXLATNT	Maximum latency of message
548	(224)	Character	8	MINLATNT	Minimum latency of message
556	(22C)	Character	8	TOTLATNT	Total latency of messages
564	(234)	Unsigned	4	*	Reserved
568	(238)	Signed	4	USE_COUNT	Use count (plus 1 for **MQOPEN**, minus 1 for **MQCLOSE**)
572	(23C)	Signed	4	TOTAL_USE	Total number of calls using this queue
576	(240)	Character	0	*	End of Structure

Cross reference

Name	Hex Offset
BASENAME	50
CFSTRUCNAM E	1E0
CLOSECT	C0
CLOSEET	B8
CLOSEEYE	B4
CLOSEN	C8
CLOSTIME	88
CORREL	10
GETA	EC
GETBRWA	E4
GETBRWS	E8
GETBYTES	1F8
GETCT	D8
GETEPAGE	11C
GETERR	F4
GETET	D0
GETEXMSG	124
GETEYE	CC
GETJWET	F8
GETJWN	100
GETMAXMS	20C
GETMINMS	210
GETN	E0
GETPSET	104
GETPSN	10C
GETS	F0
GETSMSG	120
GETSUSET	110
GETSUSN	118
INDXTYPE	94

Name	Hex Offset
INQCT	1AC
INQET	1A4
INQEYE	1A0
INQN	1B4
MAXLATNT	21C
MINLATNT	224
NBUFFPOOL	1EC
NGEN	208
NPS	1DC
OBJNAME	20
OPENCT	A8
OPENET	A0
OPENEYE	9C
OPENN	B0
OPENTIME	80
PUTBYTES	1F0
PUTCT	134
PUTET	12C
PUTEYE	128
PUTJWET	140
PUTJWN	148
PUTMAXMS	214
PUTMINMS	218
PUTN	13C
PUTPSET	158
PUTPSN	160
PUTSUSET	14C
PUTSUSN	154
PUT1CT	170
PUT1ET	168
PUT1EYE	164

Name	Hex Offset
PUT1JWET	17C
PUT1JWN	184
PUT1N	178
PUT1PSET	194
PUT1PSN	19C
PUT1SUSET	188
PUT1SUSN	190
QSGDISP	98
QTYPE	90
SETCT	1C4
SETET	1BC
SETEYE	1B8
SETJWET	1D0
SETJWN	1D8
SETN	1CC
TOTAL_USE	23C
TOTLATNT	22C
USE_COUNT	238
VALIDGET	204
VALIDPUT	200
WQBACK	234
WQEYE	4
WQID	0
WQLL	2
WQNEXT	C
WQSTAT	0
WQVER	8

Task related information (WTAS)

This data is present in SMF 116 subtype 1 records. See Self-defining sections on page 90.

Offsets					
Dec	Hex	Type	Len	Name	Description
0	(0)	Structure	712	WTAS	
0	(0)	Signed	2	WTASSHEX	Hex ID of block
2	(2)	Signed	2	WTASLEN	Length of block
4	(4)	Character	4	WTASEYEC	Eye catcher
8	(8)	Character	16	WTASCORR	Correlator identifier
8	(8)	Character	8	WTASSTRT	When WTAS allocated
16	(10)	Character	8	WTASHASH	Reserved
16	(10)	Signed	4	WTASMTHR	Reserved
20	(14)	Signed	4	WTASWTAS	Reserved
24	(18)	Character	8	WTASLATC	Reserved
32	(20)	Signed	4	WTASHSHI	Reserved
36	(24)	ADDRESS	4	*	Reserved

40	(28)	Bitstring	4	*	Reserved
44	(2C)	Character	4	*	Reserved
48	(30)	Character	384	WTASTHST	Thread stats, of interest to all users of accounting
48	(30)	Character	8	*	Reserved
304	(130)	INTEGER	4		Reserved

Non-queue 'other' statistics

432	(1B0)	Character	8	WTASOTET	Other MQI calls elapsed time
440	(1B8)	Character	8	WTASOTCT	Other MQI calls CPU time
448	(1C0)	Unsigned	4	WTASOTN	Number of other calls
452	(1C4)	Character	8	WTASMLW	Maximum latch wait time
460	(1CC)	Signed	4	WTASMLWN	Maximum wait latch number
464	(1D0)	Character	4	*	Reserved
468	(1D4)	Signed	4	*	Reserved

Commit statistics

472	(1D8)	Character	8	WTASCMET	Commit elapsed time
480	(1E0)	Character	8	WTASCMCT	Commit CPU time
488	(1E8)	Signed	4	WTASCMN	Commit number of calls

Backout statistics

492	(1EC)	Character	8	WTASBAET	Backout elapsed time
500	(1F4)	Character	8	WTASBACT	Backout CPU time
508	(1FC)	Signed	4	WTASBAN	Backout number of calls
512	(200)	Character	4	*	Reserved

Journal and logging information

516	(204)	Character	8	WTASJWET	Log write elapsed time in STCK format
524	(20C)	Unsigned	4	WTASJWN	Number of log writes WTASJWB if required
528	(210)	Unsigned	4	WTASJWB	Number of bytes written to the log
532	(214)	Character	8	WTASJCET	Elapsed time waiting for log data to be forced to DASD
540	(21C)	Unsigned	4	WTASJCN	Number of times the log was forced
544	(220)	Unsigned	4	WTASSUSN	Number of times the task was suspended
548	(224)	Character	8	WTASSUSE	Total suspend time

Page set 0 logging activity

556	(22C)	Character	8	WTASPSE0	Elapse time logging page set 0
564	(234)	Unsigned	4	WTASPSN0	Logging requests page set 0

DB2 manager

568	(238)	Character	8	WTASDBET	DB2 elapse thread
576	(240)	Character	8	WTASDBES	DB2 elapse server
584	(248)	Character	8	WTASDBMT	DB2 maximum elapse thread

592	(250)	Character	8	WTASDBMS	DB2 maximum elapse server
600	(258)	Signed	4	WTASDBCT	DB2 requests

CF manager

604	(25C)	Unsigned	4	WTASCSEC	Number of IXLLSTE calls
608	(260)	Unsigned	4	WTASCMEC	Number of IXLLSTM calls
612	(264)	Unsigned	4	WTASRSEC	Number of IXLLSTE redrives
616	(268)	Unsigned	4	WTASRMEC	Number of IXLLSTM redrives
620	(26C)	Character	8	WTASSSTC	Time spent IXLLSTE calls
628	(274)	Character	8	WTASMSTC	Time spent IXLLSTM calls
640	(280)	Character	8	* (3)	Reserved

Interval data, page counts and chain pointers

664	(298)	Character	8	WTASINTS	Interval start - for post processing
672	(2A0)	Character	8	WTASINTE	Interval end - for post processing
680	(2A8)	Signed	4	WTASGPO	Get pages old
684	(2AC)	Signed	4	WTASGPN	Get rages new

Cross reference

Name	Hex Offset
WTAS	0
WTASBACT	1F4
WTASBAET	1EC
WTASBAN	1FC
WTASCMCT	1E0
WTASCMEC	260
WTASCMET	1D8
WTASCMN	1E8
WTASCORR	8
WTASCQB	2B4
WTASCQF	2B0
WTASCSEC	25C
WTASDBCT	258

Name	Hex Offset
WTASDBES	240
WTASDBET	238
WTASDBMS	250
WTASDBMT	248
WTASEYEC	4
WTASGPN	2AC
WTASGPO	2A8
WTASHASH	10
WTASHSHI	20
WTASINTE	2A0
WTASINTS	298
WTASJCET	214
WTASJCN	21C

Name	Hex Offset
WTASJWB	210
WTASJWET	204
WTASJWN	20C
WTASLATC	18
WTASLEN	2
WTASLWET	30
WTASLWN	130
WTASMLW	1C4
WTASMLWN	1CC
WTASMSTC	274
WTASMTHR	10
WTASOTCT	1B8
WTASOTET	1B0

Name	Hex Offset
WTASOTN	1C0
WTASPSE0	22C
WTASPSN0	234
WTASRMEC	268
WTASRSEC	264
WTASSHEX	0
WTASSSTC	26C
WTASSTRT	8
WTASSUSE	224
WTASSUSN	220
WTASTHST	30
WTASWTAS	14

Task identification(WTID)

Offsets					
Dec	Hex	Type	Len	Name	Description
0	(0)	Structure	208	WTID	
0	(0)	Signed	2	WTIDSHEX	Hex ID of block
2	(2)	Signed	2	WTIDLEN	Length of block
4	(4)	Character	4	WTIDEYEC	Eye catcher
8	(8)	Character	186	WTASID	

8	(8)	Signed	4	WTIDATYP	CCBCTCOD 1=CICS etc
12	(C)	Character	8	WTIDCCN	CCBNAME connection name
20	(14)	Character	8	WTIDOPID	CCBOPID operator ID
28	(1C)	Character	16	WTIDNID	NID
44	(2C)	Character	12	WTIDCORI	Correlator
56	(38)	Character	24	WTIDUOWI	LUWID
80	(50)	Character	22	WTIDACCT	Accounting token
102	(66)	Character	20	WTIDCHL	Channel name
122	(7A)	Character	48	WTIDCHLC	Channel connection name
170	(AA)	Character	16	WTIDCTXT	Current context token
186	(BA)	Character	8	WTIDTRAN	CCBUSER MVS user ID
194	(C2)	Character	2	*	Reserved
196	(C4)	ADDRESS	4	WTIDCFWD	Reserved
200	(C8)	ADDRESS	4	WTIDCBWD	Reserved
204	(CC)	ADDRESS	4	WTIDWTAS	Reserved
208	(D0)	Character	0	*	Reserved

Cross reference

Name	Hex Offset
WTASID	8
WTID	0
WTIDACCT	50
WTIDATYP	8
WTIDCBWD	C8

Name	Hex Offset
WTIDCCN	C
WTIDCFWD	C4
WTIDCHL	66
WTIDCHLC	7A
WTIDCORI	2C
WTIDCTXT	AA

Name	Hex Offset
WTIDEYEC	4
WTIDLEN	2
WTIDNID	1C
WTIDOPID	14
WTIDSHEX	0
WTIDTRAN	BA

Name	Hex Offset
WTIDUOWI	38
WTIDWTAS	CC

How to interpret the correlator field

Table 14. Structure of the WTIDCORI for a CICS system

Offsets		Type	Len	Name	Description
Dec	Hex				
44	(2C)	Hex	4	WTICTNO	CICS thread number.
48	(30)	Character	4	WTIDCTRN	CICS transaction name.
52	(34)	Packed Decimal	4	WTIDCTSK	CICS task number.

Table 15. Structure of WTIDCORI for an IMS system

Offsets		Type	Len	Name	Description
Dec	Hex				

44	(2C)	Character	4	WTIDPST	IMS partition specification table (PST) region identifier.
48	(30)	Character	8	WTIDPSB	IMS program specification block (PSB) name.

Meaning of the channel names

The channel name in the WTID has the following meaning. For a sender channel from queue manager V521 to V52A the following fields are set with examples of their contents

Field name	**Meaning**	**Example**
For queue manager V521 the sender channel has the following fields set:		
WTIDCCN	The job name	V521CHIN
WTIDCHL	The channel name	V521.V52A
WTIDCHLC	This is defined in the CONNAME of the channel	WINMVS2B(2162)
For the queue manager V52A the receiver channel has the following fields set:		
WTIDCCN	The job name	V52ACHIN
WTIDCHL	The channel name	V521.V52A
WTIDCHLC	Where the channel came from	9.20.101.14

Structure of the MQSeries SMF header QHWS

Table 16. Structure of the Common MQSeries SMF header record QWHS

Offsets					
Dec	**Hex**	**Type**	**Len**	**Name**	**Description**
0	***(0)***	***Structure***	***128***	***QWHS***	
0	(0)		6		Reserved.
6	(6)	Character	1	QWHSNSDA	Number of self defining sections in the SMF records.
7	(7)		5		Reserved.
12	(0C)	Character	4	QWHSSSID	Subsystem name.
16	(10)		24		Reserved.
40	(28)	Character	8	QWHCAID	User ID associated with the OS/390 job.
48	(30)	Character	12	QWHCCV	Thread cross reference (see Thread cross reference data on page 21)
60	(3C)	Character	8	QWHCCN	Connection name.
68	(44)		8		Reserved.
76	(4C)	Character	8	QWHCOPID	User ID associated with the transaction.
84	(54)	Signed	4	QWHCATYP	Type of connecting system (1=CICS, 2=Batch or TSO, 3=IMS control region, 4=IMS MPP or BMP, 5=Command server, 6=Channel initiator, 7=RRS Batch).
88	(58)	Character	22	QWHCTOKN	Accounting token set to the OS/390 accounting information for the user
110	(6E)	Character	16	QWHCNID	Network identifier
126	(7E)		2		Reserved.

Appendix C. Bibliography

This section describes the IBM documentation referred to in the document.

- *MQSeries for OS/390 System Setup Guide SC34-5651*
- *MQSeries for OS/390 System Administration Guide SC34-5652*
- *MQSeries for OS/390 Concepts and planning Guide GC34-5650*
- *MVS System Management Facilities(SMF) GC28-1783*
- *Performance Reporter R5 Language Guide and Reference SH19-6817-05*
- *Performance Reporter for OS/390 Release 5 Administration Guide SH19-6816-05*
- *OS/390 MVS Initialization and Tuning Reference SC28-1752*
- *DFSORT R13 ICETOOL Mini-User Guide GC26-7140-01*

Sending your comments to IBM

MP1B:MQSeries for OS/390 V5.2

Interpreting accounting and statistics data Version 1.3 MQSeries for OS/390 V5.2.

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM.

Feel free to comment on what you regard as specific errors or omissions, and on the accuracy, organization, subject matter, or completeness of this book. Please limit your comments to the information in this book and the way in which the information is presented.

To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate, without incurring any obligation to you.

You can send your comments to IBM in any of the following ways:

- By mail, to

  ```
  IBM United Kingdom Laboratories
  AIM WW Technical Sales (MP102)
  Hursley Park
  Hursley
  Hampshire, SO21 2JN, England
  ```

- By fax:
  - ☐ From outside the U.K., after your international access code use 44 1962 841409
  - ☐ From within the U.K., use 01962 841409
- Electronically, use the appropriate network ID:
  - ☐ IBMLink: IBMGB(AIMPACS)
  - ☐ Internet: aimpacs@uk.ibm.com

Whichever you use, ensure that you include:

- The publication number and title
- The page number or topic to which your comment applies
- Your name and address/telephone number/fax number/network ID.

Readers' Comments

MP1B:MQSeries for OS/390 V5.2
Interpreting accounting and statistics data Version 1.3 MQSeries for OS/390 V5.2.

Use this form to tell us what you think about this manual. If you have found errors in it, or if you want to express your opinion about it (such as organization, subject matter, appearance) or make suggestions for improvement, this is the form to use.

To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer. This form is provided for comments about the information in this manual and the way it is presented.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Be sure to print your name and address below if you would like a reply.

Name

Address

Company or Organization

Telephone

Email

MP1B:MQSeries for OS/390 V5.2
Interpreting accounting and statistics data Version 1.3 MQSeries for OS/390 V5.2..