IBM DB2 Universal Database

System Monitor Guide and Reference

Version 5

S10J-8164-00

IBM DB2 Universal Database

System Monitor Guide and Reference

Version 5

S10J-8164-00

Before using this information and the product it supports, be sure to read the general information under Appendix E, “Notices” on page 283.

Order publications through your IBM representative or the IBM branch office serving your locality or by calling 1-800-879-2755 in U.S. or 1-800-IBM-4YOU in Canada.

Contents

About This Book

Your DB2 Database Manager is instrumented to gather data on its operation and performance. You can use this data to:

- Monitor database activities
- Assist in problem determination
- Analyze performance
- Help configure the system.

The DB2 DBMS function that collects this data is called the database system monitor. This book describes how to use the database system monitor.

Various tools allow users to exploit the strengths of the database system monitor with minimal explicit knowledge of its associated commands, APIs, or data formats. Some of these tools, for example the Control Center, are described briefly, but for detailed information you should refer to the *Administration Getting Started*.

Who Should Use This Book

This book is for any users who require an understanding of the operation of the DB2 database system monitor, including how to program to its interface.

It is intended for database administrators, system administrators, security administrators and system operators who are maintaining a database accessed by local or remote clients. It is also for software developers who are interested in building software tools that use the DB2 database system monitor to assist in these administrative functions.

How This Book is Structured

This book starts with a description of the database system monitor and then details the data that you can collect with it.

Chapter 1, Introducing the Database System Monitor, introduces the database system monitor and describes its capabilities.

Chapter 2, Using the Database System Monitor, describes the information that is available from the database system monitor: how to collect it and how to work with it.

Chapter 3, Database System Monitor Data Elements, provides details of the information elements that you can collect with the database system monitor.

Chapter 4, Event Monitor Output, is for programmers who want to write applications that read records from an event monitor trace.

Appendix A, Database System Monitor Interfaces, contains detailed descriptions of the commands, SQL statements, APIs, and tools that you may need to use with the database system monitor. Detailed information is provided for each API routine.

Appendix B, Parallel Edition Version 1.2 Users, is intended for DB2 Parallel Edition Version 1.2. users of database system monitor who are upgrading their system to DB2 Version 5.

Appendix C, DB2 Version 1 sqlestat Users, is intended for DB2 Version 1 sqlestat users.

Appendix D, How the DB2 Library Is Structured describes the DB2 library; including books and online help.

Appendix E, Notices contains notice and trademark information.

Conventions

You will find this book easier to use if you look for these conventions:

- **Boldface type** indicates an important item or concept
- *Italics type* indicates new terms, data elements, configuration parameters, or book titles.
- `Monospace type` indicates an example of text that is displayed on the screen or contained in a file.
- UPPERCASE TYPE indicates a file name, command name, or acronym.

Text in examples can be black or a lighter type.

```
db2 commands and output associated with the
database system monitor are in black type
other db2 commands used are in lighter type
```

Chapter 1. Introducing the Database System Monitor

This chapter gives you a brief overview of the database system monitor's capabilities. It also discusses the integral role that the database system monitor plays in monitoring database activity and performance.

If you want to get started quickly, read this chapter and Chapter 2, "Using the Database System Monitor" on page 3. The information in these two chapters, combined with the reference material in Appendix A, "Database System Monitor Interfaces" on page 213, provides the information required to use the database system monitor.

Chapter 3, "Database System Monitor Data Elements" on page 31 provides complete details on all the data available with the database system monitor.

Database System Monitor Capabilities

The capabilities of the database system monitor opens several possibilities:

- **Activity monitoring**

  For example, using the database system monitor you can obtain:

  – The list of database connections:
    - The status of each connection.
    - The SQL that each is executing.
    - The locks that each holds.
  – The tables being accessed and the number of rows read and written for each.

  You can also track the progression of a query or application using information, such as:

  – The cursors that are currently open for this application.
  – The number of rows read or CPU consumed (if available from the operating system) by this application.
  – How long each query has been running.
  – How long an application has been idle.

- **Problem determination**

  You can collect data to help diagnose the cause of poor system and application performance. For example:

  – By tracing deadlocks you can determine conflicts between applications that lead to poor overall system performance.
  – By looking at the amount of time applications spent waiting for locks and which application is holding these locks you can identify applications that fail to commit their transactions, a common cause of poor system performance.

- **Performance analysis**

  You can use the information available to analyze the performance of individual applications or SQL queries. For example, you can monitor for:

  – The CPU consumed by each individual statement or application.
  – The time it takes to run a statement.
  – The number of rows read and returned.
  – The use of database resources, such as buffer pool, prefetchers, and SQL cache.

  These run-time metrics are useful in tuning queries for optimal utilization of your database resources. Modifying a query or certain system parameters can result in dramatic performance improvements. The impact of your modifications can be measured with the database system monitor.

  You can also track the usage of indexes and tables, and in a partitioned database, the progression of a query on each partition. Adding indices or repartitioning the data often results in significant performance improvements.

  Carrying out some these performance analysis tasks may also require input that is obtained from the operating system, such as system load or the amount of free storage, or from other DB2 tools such as the **db2 explain facility**. For example, the db2expln application lets you analyze the *access plan* generated by the SQL compiler, which can then be compared with the run-time information available from the database system monitor.

- **System configuration**

  You can assemble the information necessary to evaluate and tune the effectiveness of your database manager and database configuration.

You can use the database system monitor to help monitor, tune, and manage your databases whether they are local or remote.

Chapter 2. Using the Database System Monitor

This chapter describes the data that is available from the DB2 Version 5 database system monitor. It explains how you can either take a **snapshot** of this data, or request the database manager to log information when certain **events** take place.

It describes the types of snapshots that you can take, and how they can be taken using CLP (command line processor) commands or APIs (application programming interfaces). It details the types of event monitors that can be used for data collection, and how to collect that information using commands or tools that come with DB2.

Database Manager Maintains Operation and Performance Data

Built into the database manager is the ability to collect data about its operation and performance, and that of the applications using it. The database manager maintains information at the following levels:

- Database manager
- Database
- Application (database connection)
- Table
- Table space
- Buffer pool
- Transaction
- Statement
- Subsection

Collecting some of this data introduces some processing overhead. For example, in order to calculate the execution time of an SQL statement, the database manager must make a call to the operating system to obtain timestamps before and after statement execution. These types of system calls are generally expensive. In order to minimize the overhead involved in maintaining monitoring information, **monitor switches** control the collection of potentially expensive data by the database manager.

Monitor Switches Control Data Collected by the Database Manager

The database system monitor will always collect some basic information, but you can use the switches to govern the amount of expensive data collected. Monitor switches can be set:

- **Explicitly**, this is usually done using the UPDATE MONITOR SWITCHES command.

  You can also set these switches in the database manager configuration file if you want data collection to start from the moment the server is started. You should note that setting switches in this way means that they cannot be turned off without stopping the database management system. Switches are explained in “Resetting Monitor Data” on page 21. For more information on configuration see the *dft_monswitches* configuration parameters in the *Administration Guide*.

- **Implicitly**, when an event monitor is activated. Event monitors are explained in "Event Monitors" on page 10.

To see if your database manager is currently collecting any monitor data issue the command:

```
db2 get database manager monitor switches
```

The resulting output indicates the database manager switch settings and the time that they were turned on.

```
               DBM System Monitor Information Collected

     Buffer Pool Activity Information (BUFFERPOOL) = OFF
     Lock Information                        (LOCK) = OFF
     Sorting Information                     (SORT) = ON   04-18-1997 10:11:01.738400
     SQL Statement Information          (STATEMENT) = OFF
     Table Activity Information             (TABLE) = OFF
     Unit of Work Information                 (UOW) = OFF
```

In this example, in addition to collecting basic-level information, the database manager is collecting all information under control of the sort switch.

Accessing Monitor Data

There are two ways to access the monitor data collected by the database manager:

- **Snapshot monitoring**

  Taking a snapshot gives you information for a specific point in time. A snapshot is a picture of the current state of activity in the database manager for a particular object or group of objects.

- **Event monitors**

  You can request the database manager to automatically log monitor data to files or a named pipe when specific events occur. This allows you to collect information about transient events that are difficult to monitor through snapshots, such as deadlocks and transaction completions.

Snapshot Monitoring

The snapshot monitor provides two categories of information for each level being monitored:

- State

  This includes information such as:

  – the current status of the database
  – information on the current or most recent unit of work
  – the list of locks being held by an application
  – the status of an application
  – the current number of connections to a database

  - the most recent SQL statement performed by an application
  - run-time values for configurable system parameters.

- Counters

  These accumulate counts for activities from the time monitoring started until the time a snapshot is taken. Such as:

  - the number of deadlocks that have occurred
  - the number of transactions performed on a database
  - the amount of time an application has waited on locks.

For example, you can obtain a list of the locks held by applications connected to a database by taking a database lock snapshot. First, turn on the LOCK switch (UPDATE MONITOR SWITCHES), so that the time spent waiting for locks is collected.

```
db2 connect to sample
db2 update monitor switches using LOCK on
db2 +c list tables for all             # this command will require locks
                                       # on the database catalogs
db2 get snapshot for locks on sample
```

Note: You can create and populate the sample database by running `sqllib/misc/db2sampl`.

Issuing the GET SNAPSHOT command returns the following.

```
         Database Lock Snapshot

Database name                               = SAMPLE
Database path                               = /home/bourbon/bourbon/NODE0000/SQL00005/
Input database alias                        = SAMPLE
Locks held                                  = 6
Applications currently connected            = 1
Applications currently waiting on locks     = 0
Snapshot timestamp                          = 04-11-1997 10:40:29.976539

Application handle                          = 1
Application ID                              = LOCAL.bourbon.970411143813
Sequence number                             = 0001
Application name                            = db2bp_32
Authorization ID                            = BOURBON
Application status                          = UOW Waiting
Status change time                          = Not Collected
Application code page                       = 850
Locks held                                  = 6
Total wait time (ms)                        = 0

Object Type     Tablespace Name      Table Schema         Table Name           Mode Status
--------------- -------------------- -------------------- -------------------- ---- ----------
Row             SYSCATSPACE          SYSIBM               SYSTABLES            NS   Granted
Table           SYSCATSPACE          SYSIBM               SYSTABLES            IS   Granted
Table           SYSCATSPACE          SYSIBM               SYSTABLESPACES       S    Granted
Row             SYSCATSPACE          SYSIBM               SYSPLAN              S    Granted
Table           SYSCATSPACE          SYSIBM               SYSPLAN              IS   Granted
Internal                                                                       S    Granted
```

Figure 1. Results of GET SNAPSHOT FOR LOCKS Command

From this snapshot, you can see that there is currently one application connected to the SAMPLE database, and it is holding six locks.

```
Locks held                                  = 6
Applications currently connected            = 1
```

Note that the time (`Status change time`) when the `Application status` became `UOW Waiting` is returned as `Not Collected`, because the UOW switch is OFF.

The lock snapshot also returns the total time spent waiting for locks (so far), by applications connected to this database.

```
Total wait time (ms)                        = 0
```

This is an example of an accumulating counter. “Resetting Monitor Data” on page 21 explains how counters can be reset to zero.

Authority Required for Snapshot Monitoring

To perform any of the snapshot monitor tasks, you must have SYSMAINT, SYSCTRL, or SYSADM authority for the database manager instance that you wish to monitor.

Snapshot Monitor Interface

Snapshot monitoring is invoked using the following application programming interfaces (APIs):

sqlmon()	set or query monitor switch settings
sqlmonrset()	reset system monitor counters
sqlmonss()	take a snapshot
sqlmonsz()	estimate the size of the data that would be returned for a particular invocation of sqlmonss()

The Command Line Processor (CLP) provides a convenient command-based front-end to the snapshot APIs. For example, the GET SNAPSHOT command invokes the sqlmonss() API. Appendix A, "Database System Monitor Interfaces" on page 213 contains detailed information on the commands and APIs associated with the database system monitor.

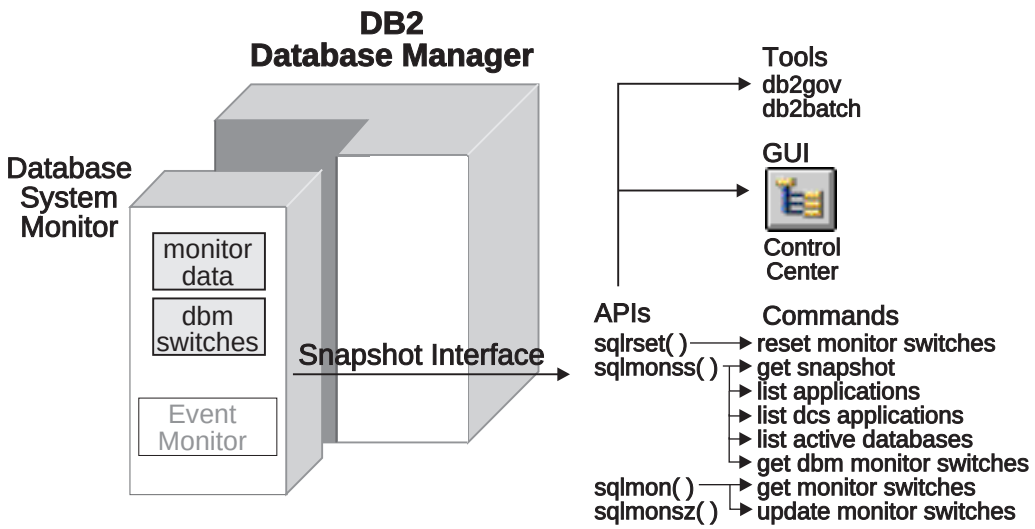


Figure 2. Snapshot Monitoring Interfaces

Information Available by Taking Snapshots

The following table lists all the supported snapshot request types. For certain request types, some information is returned only if the associated monitor switch is set ON. See Chapter 3, "Database System Monitor Data Elements" on page 31 to determine if a required counter is under switch control.

In the table, the API Request type column identifies the value that is supplied as input to the SQLMA input structure in the sqlmonss() Snapshot API routine.

API request type	CLP command	Information returned
	List of connections	
SQLMA_APPLINFO_ALL	list applications [show detail]	Application identification information for all applications currently connected to a database that is managed by the DB2 instance on the node where snapshot is taken.
SQLMA_DBASE_APPLINFO	list applications for database *dbname* [show detail]	Same as SQLMA_APPLINFO_ALL for each application currently connected to the specified database.
SQLMA_DCS_APPLINFO_ALL	list dcs applications	Application identification information for all DCS applications currently connected to a database that is managed by the DB2 instance on the node where snapshot is taken.
	Database manager snapshot	
SQLMA_DB2	get snapshot for dbm	Database manager level information, including internal monitor switch settings.
	get dbm monitor switches	Returns internal monitor switch settings.
	Database snapshot	
SQLMA_DBASE	get snapshot for database on *dbname*	Database level information and counters for a database. Information is returned only if there is at least one application connected to the database.
SQLMA_DBASE_ALL	get snapshot for all databases	Same as SQLMA_DBASE for each database active on the node.
	list active databases	The number of connections to each active database. Includes databases that were started using the ACTIVATE DATABASE command, but have no connections.
	Application snapshot	
SQLMA_APPL	get snapshot for application applid *appl-id*	Application level information, includes cumulative counters, status information, and most recent SQL statement executed (if statement switch is set).
SQLMA_AGENT_ID	get snapshot for application agentid *appl-handle*	Same as SQLMA_APPL.
SQLMA_DBASE_APPLS	get snapshot for applications on *dbname*	Same as SQLMA_APPL, for each application that is connected to the database on the node.
SQLMA_APPL_ALL	get snapshot for all applications	Same as SQLMA_APPL, for each application that is active on the node.
	Table snapshot	

API request type	CLP command	Information returned
SQLMA_DBASE_TABLES	get snapshot for tables on *dbname*	Table activity information at the database and application level for each application connected to the database, and at the table level for each table that **was accessed** by an application connected to the database. Requires the table switch.
	Lock snapshot	
SQLMA_APPL_LOCKS	get snapshot for locks for application applid *appl-id*	List of locks held by the application. Also, lock wait information if any and the lock switch is ON.
SQLMA_APPL_LOCKS_AGENT_ID	get snapshot for locks for application agentid *appl-handle*	Same as SQLMA_APPL_LOCKS.
SQLMA_DBASE_LOCKS	get snapshot for locks on *dbname*	Lock information at the database level, and application level for each application connected to the database. Requires the lock switch.
	Table space snapshot	
SQLMA_DBASE_TABLESPACES	get snapshot for tablespace on *dbname*	Information about table space activity at the database level, the application level for each application connected to the database, and the table space level for each table space that has been accessed by an application connected to the database. Requires the buffer pool switch.
	Buffer pool snapshot	
SQLMA_BUFFERPOOLS_ALL	get snapshot for all bufferpools	Buffer pool activity counters. Requires the buffer pool switch.
SQLMA_DBASE_BUFFERPOOLS	get snapshot for bufferpools on *dbname*	Same as SQLMA_BUFFERPOOLS_ALL, but for specified database only.

Snapshot Uses an Instance Connection

You do not need to be connected to a database in order to use the snapshot APIs. They are performed under an **instance connection**, which is a connection between an application and an instance of the DB2 database manager.

The instance attachment is usually done implicitly to the instance specified by the DB2INSTANCE environment variable when the first database system monitor API is invoked by the application. It can also be done explicitly, using the ATTACH TO NODE command.

Once an application is attached, all system monitor requests that it invokes are directed to that instance. This allows a client to monitor a remote server, by simply attaching to the instance on it.

Availability of Snapshot Monitor Data

If all applications disconnect from a database, then the system monitor data for that database is no longer available. To obtain monitor information for all database activity during a given period you may want to use an event monitor. Alternatively, you can keep the database active until your final snapshot has been taken, either by starting it with the ACTIVATE DATABASE command, or by maintaining a permanent connection to the database.

Event Monitors

In contrast to taking a point in time snapshot, an event monitor writes out database system monitor data to either a file or a named pipe, when one of the following events occurs:

- end of a transaction
- end of a statement
- a deadlock
- start of a connection
- end of a connection
- database activation
- database deactivation
- end of a statement's subsection (when a database is partitioned)

An event monitor effectively provides the ability to obtain a **trace** of the activity on a database.

For example, you can request that DB2 logs the occurrence of deadlocks between connections to a database. First, you must create and activate a DEADLOCK event monitor:

For UNIX systems

Monitor Session

```
db2 connect to sample
db2 "create event monitor dlockmon for
  deadlocks write to file '/tmp/dlocks'"
mkdir /tmp/dlocks
db2 "set event monitor dlockmon state 1"
```

For OS/2 and Windows systems

Monitor Session

```
db2 connect to sample
db2 "create event monitor dlockmon for
  deadlocks write to file 'c:\tmp\dlocks'"
mkdir c:\tmp\dlocks
db2 "set event monitor dlockmon state 1"
```

Now, two applications using the database enter a deadlock. That is, each one is holding a lock that the other one needs in order to continue processing. The deadlock is eventually detected and resolved by the DB2 deadlock detector component, which will rollback one of the transactions. The following figures illustrate this scenario.

Application 1

```
db2 connect to sample
db2 +c "insert into staff values (1, 'Ofer',
  1, 'Mgr', 0, 0, 0)"
DB20000I  The SQL command completed
successfully.
```

Note: The +c option turns autocommit off for CLP.

Application 1 is now holding an exclusive lock on a row of the staff table.

Application 2

```
db2 connect to sample
db2 +c "insert into department values ('1',
  'System Monitor', '1', 'A00', NULL)"
DB20000I  The SQL command completed
successfully.
```

Application 2 now has an exclusive lock on a row of the department table.

Application 1

```
db2 +c select deptname from department
```

Assuming cursor stability, Application 1 needs a share lock on each row of the department table as the rows are fetched, but a lock on the last row cannot be obtained because Application 2 has an exclusive lock on it. Application 1 enters a LOCK WAIT state, while it waits for the lock to be released.

Application 2

```
db2 +c select name from staff
```

Application 2 also enters a LOCK WAIT state, while waiting for Application 1 to release its exclusive lock on the last row of the staff table.

These applications are now in a deadlock. This waiting will never be resolved because each application is holding a resource that the other one needs to continue. Eventually, the deadlock detector checks for deadlocks (see the *dlchktime* database manager configuration parameter in the *Administration Guide*) and chooses a victim to rollback:

Application 2

```
SQLN0991N  The current transaction has been
rolled back because of a deadlock or timeout.
Reason code "2". SQLSTATE=40001
```

At this point the event monitor logs a deadlock event to its target. Application 1 can now continue:

Application 1

```
DEPTNAME
------------------------------
PLANNING
INFORMATION CENTER
. . .
SOFTWARE  SUPPORT
SYSTEM MONITOR

  9 record(s) selected
```

Because an event monitor buffers its output and this scenario did not generate enough event records to fill a buffer, the event monitor is turned off to force it to flush its buffers:

Monitor Session

```
db2 "set event monitor dlockmon state 0"
DB20000I  The SQL command completed
successfully.
```

The event trace is written as a binary file. It that can now be formatted using the db2evmon tool:

Monitor Session

```
db2evmon -path /tmp/dlocks

Reading /tmp/dlocks/00000000.evt . . .
```

This will format and print to *stdout*, a trace similar to the following:

```
------------------------------------------------------------------------
                            EVENT LOG HEADER
  Event Monitor name: DLOCKMON
  Server Product ID: SQL05000
  Version of event monitor data: 5
  Byte order: BIG ENDIAN
  Number of nodes in db2 instance: 1
  Codepage of database: 850
  Country code of database: 1
  Server instance name: bourbon
------------------------------------------------------------------------

------------------------------------------------------------------------
  Database Name: SAMPLE
  Database Path: /home/bourbon/bourbon/NODE0000/SQL00002/
  First connection timestamp: 06-03-1997 13:31:13.607548
  Event Monitor Start time:   06-03-1997 13:32:11.676071
------------------------------------------------------------------------

3) Connection Header Event ...
  Appl Handle: 0
  Appl Id: *LOCAL.bourbon.970603173114                      - Monitor session
  Appl Seq number: 0001
  DRDA AS Correlation Token: *LOCAL.bourbon.970603173113
  Program Name    : db2bp_32
  Authorization Id: BOURBON
  Execution Id    : bourbon
  Codepage Id: 850
  Country code: 1
  Client Process Id: 63590
  Client Database Alias: sample
  Client Product Id: SQL05000
  Client Platform: AIX
  Client Communication Protocol: Local
  Client Network Name:
  Connect timestamp: 06-03-1997 13:31:13.607548

4) Connection Header Event ...
  Appl Handle: 1                                          - Application 1
  Appl Id: *LOCAL.bourbon.970603173330
  Appl Seq number: 0001
  DRDA AS Correlation Token: *LOCAL.bourbon.970603173329
  Program Name    : db2bp_32
  Authorization Id: BOURBON
  Execution Id    : bourbon
  Codepage Id: 850
  Country code: 1
  Client Process Id: 119710
  Client Database Alias: sample
  Client Product Id: SQL05000
  Client Platform: AIX
  Client Communication Protocol: Local
  Client Network Name:
  Connect timestamp: 06-03-1997 13:33:29.518568

5) Connection Header Event ...
  Appl Handle: 2
  Appl Id: *LOCAL.bourbon.970603173409                    - Application 2
  Appl Seq number: 0001
  DRDA AS Correlation Token: *LOCAL.bourbon.970603173408
  Program Name    : db2bp_32
  Authorization Id: BOURBON
  Execution Id    : bourbon
  Codepage Id: 850
  Country code: 1
  Client Process Id: 33984
  Client Database Alias: sample
```

```
  Client Product Id: SQL05000
  Client Platform: AIX
  Client Communication Protocol: Local
  Client Network Name:
  Connect timestamp: 06-03-1997 13:34:08.972643

6) Deadlock Event ...
  Number of applications deadlocked: 2                  - Deadlock
  Deadlock detection time: 06-03-1997 13:36:48.817786
  Rolled back Appl Id: : *LOCAL.bourbon.970603173409
  Rolled back Appl seq number: : 0001

7) Deadlocked Connection ...
  Appl Id: *LOCAL.bourbon.970603173409
  Appl Seq number: 0001
  Appl Id of connection holding the lock: *LOCAL.bourbon.970603173330
  Seq. no. of connection holding the lock:
  Lock wait start time: 06-03-1997 13:36:43.251687
  Deadlock detection time: 06-03-1997 13:36:48.817786
  Table of lock waited on      : STAFF
  Schema of lock waited on     : BOURBON
  Tablespace of lock waited on : USERSPACE1
  Type of lock: Row
  Mode of lock: X
  Lock object name: 39

8) Deadlocked Connection ...
  Appl Id: *LOCAL.bourbon.970603173330
  Appl Seq number: 0001
  Appl Id of connection holding the lock: *LOCAL.bourbon.970603173409
  Seq. no. of connection holding the lock:
  Lock wait start time: 06-03-1997 13:35:32.227521
  Deadlock detection time: 06-03-1997 13:36:48.817786
  Table of lock waited on      : DEPARTMENT
  Schema of lock waited on     : BOURBON
  Tablespace of lock waited on : USERSPACE1
  Type of lock: Row
  Mode of lock: X
  Lock object name: 15
```

This event monitor trace shows that there was 1 application connected to the database when the event monitor was activated. This is indicated by the first *Connection Header Event* record in the output (record number 3). A Connection Event Header is generated for each active connection when an event monitor is turned on, and for each subsequent connection, once it becomes active. The other two Connection Headers, (records 4 and 5) were generated when the two applications connected.

The trace also shows that a deadlock occurred (record number 6). It shows which locks on which tables caused this deadlock (record numbers 7 and 8), and which application the deadlock detector chose to roll back (record number 6).

The **db2eva** graphical tool can also be used for formatting a trace. It is particularly useful for handling file traces that are too large to be read with db2evmon. It displays collected information in a tabular format. It includes a number of different view options, which allows you to filter unwanted records and drill down to the periods of interest in the trace. For instance, you can decide to display only the transaction events for a given connection. It also allows you to view the statement text for static SQL that it automatically fetches from the DB catalog (the text is only available for dynamic SQL in the event monitor trace).

You can invoke this tool with the db2eva command (see “db2eva - Event Analyzer Command” on page 222), or by selecting the Event Analyzer from the GUI.

Note: The files must be available on the machine where you invoked db2eva.

The db2eva tool is available on OS/2 and Windows systems.

Authority Required for Event Monitoring

To define and use an event monitor on a database, you must have at least DBADM authority on that database.

Using Event Monitors

As illustrated in the sample scenario, collecting system monitor data with an event monitor is a three step process:

1. Create the event monitor
2. Activate the event monitor
3. Read the trace produced.

Create the event monitor.

Specify the events to be monitored. An event monitor is created and activated by using SQL statements. Unlike snapshot monitoring, where data can be collected at the database manager level, an event monitor only gathers data for a single database.

Creating an event monitor stores its definition in the event monitor database system catalogs:

SYSCAT.EVENTMONITORS	event monitors defined for the database
SYSCAT.EVENTS	events monitored for the database

It is necessary to connect to the database when defining an event monitor.

Activate the event monitor.

Activating an event monitor starts a process or thread, which records monitor data to either a named pipe or a file as events occur. You may want an event monitor to be activated as soon as the database is started, so that all activity is monitored from start-up. This can be done by creating an AUTOSTART event monitor:

```
db2 "create event monitor DLOCKMON
  for deadlocks write to file '/tmp/dlocks'
  AUTOSTART"
```

This event monitor will be automatically started every time the database is activated, either by using the ACTIVATE DATABASE command, or when the first application connects. Note that creating an AUTOSTART event monitor does not activate it. This event

monitor will be activated the next time the database is stopped and re-activated. An event monitor that has not been automatically started must be manually started:

```
db2 set event monitor dlockmon state 1
```

All event monitors for a database are stopped when the database is deactivated.

Read the trace produced.

Reading a trace can be done using the **db2evmon** applet, or by writing your own application (see Chapter 4, “Event Monitor Output” on page 195). The Control Center and Event Analyzer (parts of the DB2 GUI) can be used to create and activate event monitors, and to read the traces produced by FILE event monitors.

Figure 3 illustrates the process and interface for using event monitors.

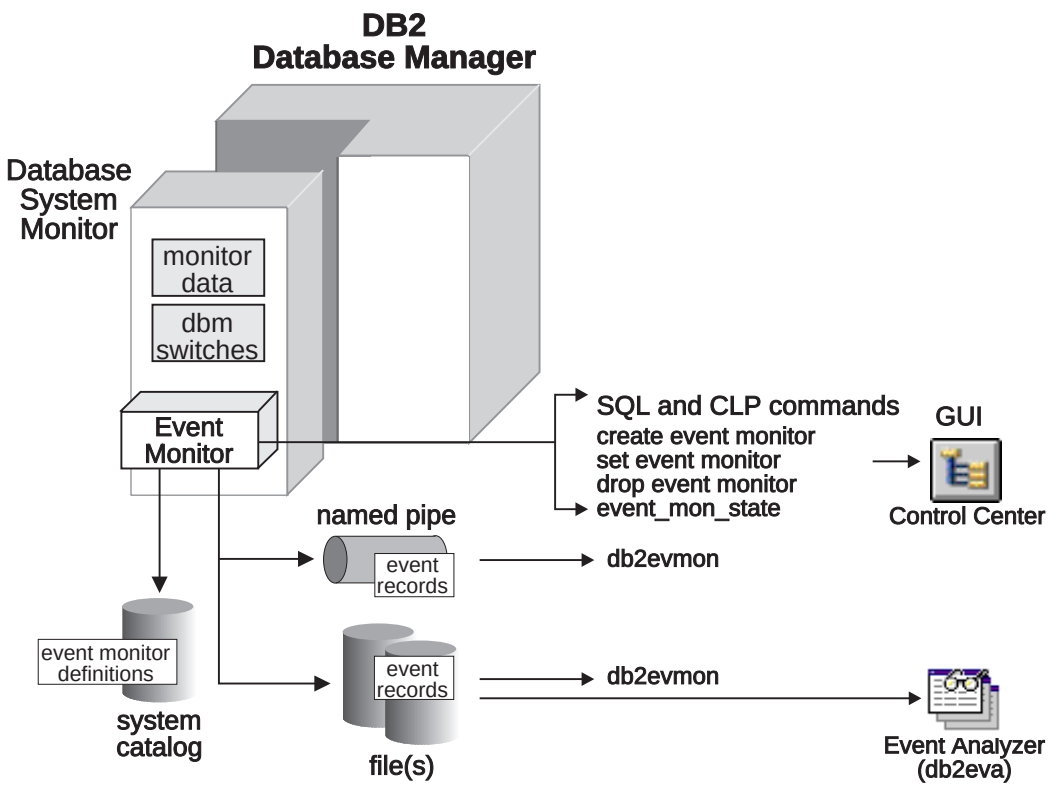


Figure 3. Event Monitoring Interfaces

As illustrated in Figure 3, event monitors are created and activated using the following SQL statements:

- CREATE EVENT MONITOR stores the event monitor definition in the database system catalogs for event monitors.
- SET EVENT MONITOR activates the event monitor, starting an **output thread** that will WRITE monitor data to either a file or named pipe. The trace produced can be formatted by the *db2evmon* or *db2eva* tools.
- DROP EVENT MONITOR deletes the event monitor definition from the database system catalogs for event monitors. An active event monitor cannot be dropped.

Querying the State of an Event Monitor

You can determine if an event monitor is active by using the SQL function EVENT_MON_STATE:

```
db2 connect to sample
db2 "select evmonname, EVENT_MON_STATE(evmonname)
from syscat.eventmonitors"

NAME            2
--------------  -------
DLOCKMON    0

  1 record(s) selected
```

A returned value of 0 indicates that the event monitor is inactive.

Information Available from Event Monitors

Event monitors return information that is similar to the information available using the snapshot API. In these cases, it is an event that controls when the snapshot is taken. For example, a connection event monitor basically takes an application snapshot just before the connection is terminated.

Event Types

When you define an event monitor you must declare the event types that will be monitored. The following table lists the event types supported and indicates the information returned. **Note:** an event monitor can be defined for more than one event type.

Event type	When data is collected	Information returned
Deadlock	Detection of a deadlock	The applications involved and locks in contention.
Statements	End of SQL statement	Statement start/stop time, CPU used, text of dynamic SQL, SQLCA (return code of SQL statement), and other metrics such as fetch count.
	End of subsection	For partitioned databases: CPU consumed, execution time, table and tablequeue information.
Transactions	End of unit of work	Unit of work start/stop time, previous UOW time, CPU consumed, locking and logging metrics.
Connections	End of connection	All application level counters.
Database	Database deactivation or last connect reset	All database level counters.
Buffer pools		Counters for buffer pool, prefetchers, page cleaners and direct I/O for each buffer pool.
Table spaces		Counters for buffer pool, prefetchers, page cleaners and direct I/O for each table space.
Tables		Rows read/written for each table.

Note: In addition to the above information, all event monitors trace the establishment of connections to the database, by generating a *connection header record* for each active connection when the event monitor is turned ON, and for each subsequent connection, thereafter.

See "Output Stream Format" on page 195 for a list of the records generated for each event type.

Using Pipe Event Monitors

A pipe event monitor allows you to process event records in real time. Another important advantage to using pipe event monitors is that your application can ignore unwanted data as it reads it off the pipe, giving the opportunity to considerably reduce storage requirements. It also allows an application to store event monitor data, in real-time, into an SQL database.

When you direct data to a pipe, I/O is always blocked and the only buffering is that performed by the pipe. It is the responsibility of the monitoring application to promptly read the data from the pipe as the event monitor writes the event data. If the event monitor is unable to write the data to the pipe (for example, because the pipe is full), monitor data will be lost.

The steps for using pipe event monitors are essentially the same on all operating systems. However, implementation can be different. The following section describes the basic steps, and highlights the differences between UNIX based systems, Windows NT, and OS/2.

1. Define the event monitor

   ```
     db2 connect to sample
   On AIX, and other UNIX platforms:
     db2 create event monitor STMT2 for statements
         write to PIPE '/tmp/evmpipe1'
   On Windows NT:
     db2 create event monitor STMT2 for statements
         write to PIPE '\\.\pipe\evmpipe1'
   On OS/2:
     db2 create event monitor STMT2 for statements
         write to PIPE '\pipe\evmpipe1'
   ```

2. Create the named pipe

   In UNIX (this includes AIX environments), use the mkfifo() function or mkfifo command. In OS/2, use the DosCreateNPipe() function. In Windows NT, use the CreateNamedPipe() function. The pipe name must be the same as the target path specified on the CREATE EVENT MONITOR statement.

3. Open the named pipe

   In UNIX, use the open() function. In OS/2, use the DosConnectNPipe() function. In Windows NT, use the ConnectNamedPipe() function.

   You can also use the db2evmon application, specifying the database and pipe name, for example:

   ```
   db2evmon -db sample -evm STMT2
   ```

   This will open the named pipe and wait for the event monitor to write to it.

4. Activate the event monitor

   If the event monitor is started automatically, you do not need to take any specific action to start it unless the database is already active (however, the pipe must already be opened).

   ```
   db2 set event monitor stmt2 state 1
   ```

5. Read data from the named pipe

   In UNIX, use the read() function. In OS/2, use the DosRead() function. In Windows NT, use the ReadFile() function. Your application may stop reading data from the pipe at any time. When it reads an EOF, there is no more monitor data. See Chapter 4, “Event Monitor Output” on page 195 for how to read the event monitor data.

6. Deactivate the event monitor

   ```
   db2 set event monitor stmt2 state 0
   ```

   This statement can be used to stop any event monitor, even one that was started automatically. If you do not explicitly stop an event monitor, it will be stopped when:

- The last application disconnects from the database
- It experiences an error while writing to the named pipe: for example, the monitoring application closes the pipe before deactivating the event monitor. In this case, the event monitor will turn itself off and log a system-error-level message in the diagnostic log, db2diag.log.

7. Close the named pipe.

   In UNIX, use the close() function. In OS/2, use the DosDisConnectNPipe() function. In Windows NT, use the DisconnectNamedPipe() function.

8. Delete the named pipe.

   In UNIX, use the unlink() function. In OS/2, use the DosClose() function. In Windows NT, use the CloseHandle() function.

For UNIX-based operating systems, named pipes are like files, so you do not have to delete them and create them again before each use.

Pipe Overflows

In addition, there must be enough space in the named pipe. If the application does not read the data fast enough from the named pipe, the pipe will fill up and overflow. Pipe overflows can also occur on platforms (such as OS/2) where the creator of the pipe can define the size of the named pipe buffer. The smaller the buffer, the greater the chance of an overflow occurring. When a pipe overflow occurs, the monitor creates overflow event records indicating that an overflow has occurred. The event monitor is not turned off, but monitor data is lost. If there are outstanding overflow event records when the monitor is deactivated, a diagnostic message will be logged. Otherwise, the overflow event records will be written to the pipe when possible.

If your operating system allows you to define the size of the pipe buffer, use a pipe buffer of at least 32K. For high-volume event monitors, you should set the monitoring application's process priority equal to or higher (lower nice value on AIX) than the agent process priority (see the section on Priority of Agents in the *Administration Guide*).

When Counters are Initialized

The data collected by the database manager includes several accumulating counters. These counters are incremented during the operation of the database, for example, every time an application commits a transaction.

Counters are initialized when their applicable object becomes active. For example, the number of buffer pool pages read for a database (a basic element) is set to zero when the database is activated.

Counters under switch control are reset to zero when their associated switch is turned on.

Counters returned by event monitors are reset to zero when the event monitor is activated.

Resetting Monitor Data

Each event monitor and any application using the snapshot monitor APIs has its own logical view of the DB2 monitor data and switches. This means that when counters are reset or initialized, it only affects the event monitor or application that reset or initialized them.

Event monitor data cannot be reset, except by turning the monitor off, and then on again.

An application taking snapshots can reset its view of the counters at any time by using the RESET MONITOR command.

When issuing its first snapshot API, an application inherits the default settings from the database manager configuration. For example, assuming that the statement switch was set in the database manager configuration file:

```
db2 update dbm cfg using DFT_MON_STMT on
db2start
```

Issuing a GET MONITOR SWITCHES command will show that the statement switch is ON.

```
db2 get monitor switches
```

```
           Monitor Recording Switches

 Buffer Pool Activity Information (BUFFERPOOL) = OFF
 Lock Information                        (LOCK) = OFF
 Sorting Information                     (SORT) = OFF
 SQL Statement Information          (STATEMENT) = ON   05-25-1997 10:44:34.820446
 Table Activity Information             (TABLE) = OFF
 Unit of Work Information                 (UOW) = OFF
```

Turning OFF the statement switch from the command line will only affect the application issuing the command. The statement switch will still be ON for other applications (unless they have also turned it OFF). For example:

```
db2 update monitor switches using STATEMENT OFF
DB20000I The UPDATE MONITOR SWITCHES command completed successfully
```

Then query your application's switches.

```
db2 get monitor switches
```

```
            Monitor Recording Switches

   Buffer Pool Activity Information (BUFFERPOOL) = OFF
   Lock Information                        (LOCK) = OFF
   Sorting Information                     (SORT) = OFF
   SQL Statement Information           (STATEMENT) = OFF
   Table Activity Information              (TABLE) = OFF
   Unit of Work Information                  (UOW) = OFF
```

Querying the database manager switches will show that the update did not affect its settings:

```
db2 get database manager monitor switches
```

```
            DBM System Monitor Information Collected

   Buffer Pool Activity Information (BUFFERPOOL) = OFF
   Lock Information                        (LOCK) = OFF
   Sorting Information                     (SORT) = OFF
   SQL Statement Information           (STATEMENT) = ON     05-25-1997 10:44:34
   Table Activity Information              (TABLE) = OFF
   Unit of Work Information                  (UOW) = OFF
```

When a monitoring application turns off a monitor switch or resets a data element counter, the DB2 server does not reset its own internal counters. Instead, it re-initialize the private **logical view** for that user. Other monitoring applications or event monitors are not affected.

You must have SYSADM, SYSCTRL, or SYSMAINT authority to use the UPDATE MONITOR SWITCHES command. See “UPDATE MONITOR SWITCHES Command” on page 266 for information on this command.

The database manager keeps track of all the applications using the snapshot monitor APIs and their switch settings. If a switch is set in its configuration, then the database manager always collects that monitor data. If a switch is OFF in the configuration, then the database manager will collect data as long as there is at least one application with this switch turned ON.

Chapter 3, “Database System Monitor Data Elements” on page 31 shows the data elements associated with each switch group.

Internally, event monitors also use switches to instruct the engine as to which data should be collected. However, this is an implementation issue, and the switch settings for a particular event monitor cannot be queried.

An actual DBMS monitor switch is set as long as at least one application or event monitor needs it, or if it is set in the configuration file.

System Monitor Memory Requirements - (mon_heap_sz)

The memory required for maintaining the private views of the database system monitor system monitor data is allocated from the monitor heap. Its size is controlled by the *mon_heap_sz* configuration parameter. The amount of memory required for monitoring activity varies widely depending on the number of monitoring applications and event monitors, the switches set, and the level of database activity. The following formula provides an approximation of the number of pages required for the monitor heap.

```
(number of monitoring applications + 1) *
(number of databases *
  (800 + (number of tables accessed * 20) +
    ((number of applications connected + 1) *
      (600 + (number of table spaces * 100)))))
/ 4096
```

You may need to experiment with this value, increasing it if monitor commands occasionally fail with an SQLCODE of -973, when the database manager switches are on.

Partitioned Database Considerations

The database system monitor interface is the same for all types of systems, whether they use single partition or multiple partition databases and whether intra-query parallelism is used. All the commands and APIs are exactly the same. The only difference is the output; more complex systems generally return more information.

Taking a Snapshot on Multi-node Systems

On systems that use inter-partition parallelism, taking a snapshot only returns monitor data from the instance where the application is attached. For example, assuming a table that is located in two database partitions, that is some of its rows are stored on one node (Node 100) and others are stored on another node (Node 200).

Node 100

```
db2 connect to sample
db2 list applications

Auth Id  Appl Name  Appl Handle  Application Id                 DB Name  # of Agents
-------------------------------------------------------------------------------
BOURBON  db2bp_32   6553638      *LOCAL.bourbon.970414221746   SAMPLE   1
```

Taking a snapshot on Node 200 initially returns no data:

Note: The LIST APPLICATION command uses the database system monitor. Invoking it actually calls the the snapshot API sqlmonss() with a request of type SQLMA_APPLINFO_ALL.

Node 200

```
db2 list applications
SQL1611W  No data was returned from Database System Monitor.
```

Now, issuing a query from Node 100 will result in a secondary connection to Node 200 to fetch the rows that reside in that partition:

Node 100

```
db2 +c select lastname from employee

Huras
Ofer
Bourbonnais
Musker
Cartwright
```

Now there is a subagent for the application running on Node 200:

Node 200

```
db2 list applications

Auth Id  Appl Name  Appl Handle  Application Id                  DB Name  # of Agents
-------------------------------------------------------------------------------------
BOURBON  db2bp_32   6553638      *LOCAL.bourbon.970414221746    SAMPLE   1
```

And there are now two agents running on Node 100; the coordinator agent and a subagent:

Node 100

```
db2 list applications

Auth Id  Appl Name  Appl Handle  Application Id                  DB Name  # of Agents
-------------------------------------------------------------------------------------
BOURBON  db2bp_32   6553638      *LOCAL.bourbon.970414221746    SAMPLE   2
```

On the non-coordinating node, you can determine where the coordinator resides, and check if the application originated on the node that issued the snapshot, using:

Node 200

```
db2 list application show detail

. . . Appl        Application Id                    . . . Coordinating   Coordinator
      Handle                                              Node Number  pid/thread
-------------------------------------------------------------------------------------
      6553638  *LOCAL.bourbon.970414221746    100              66204
```

The *Application Handle* returned, 6553638 is unique across all nodes. The *node number* corresponds to one of the nodes listed in the *db2nodes.cfg* configuration file (see the *Administration Guide*).

Using the application handle, you can request monitor information on any node by issuing a GET SNAPSHOT FOR APPLICATION, which will return data if the application is connected on that node. You can also FORCE the application, which will work from any node:

Node 200

```
db2 force application (6553638)
DB20000I The FORCE APPLICATION command completed successfully.
DB221024I This command is asynchronous and may not be effective
immediately.
```

Using Event Monitors on Multi-node Systems

An event monitor uses an operating system process or a thread to write its trace. The node where this process or thread runs is called the **monitor node**. An event monitor can be monitoring events as they occur locally on the monitor node, or globally as they occur on any node where the DB2 database manager is running. A global event monitor writes a single trace that contains activity from all nodes.

Whether an event monitor is local or global is referred to as its **monitoring scope**. Both the monitor node and monitor scope are part of an event monitor's definition. For example:

```
db2 connect to sample
db2 "create event monitor DLOCKS for
  deadlocks write to file '/tmp/dlocks'
  ON NODE 5 GLOBAL"
```

This global event monitor will report deadlocks that involve any nodes in the system. Its I/O component will physically run on Node 5, writing its records to files in the /tmp/dlocks directory on that node.

You can look at the definition for this monitor in the system catalog:

```
db2 "select evmonname,nodenum, monscope
from syscat.eventmonitors"

EVMONNAME    NODENUM    MONSCOPE
------------------    ---------------    ----------------
DLOCKS              5                  G

  1 record(s) selected
```

The returned information shows event monitor DLOCKS is defined as global and its monitor node is 5.

Note: In DB2 Version 5, only deadlock event monitors can be defined as global, all other event monitors must be defined as local.

Monitoring Subsections

On systems that use inter-partition parallelism, the SQL compiler partitions the access plan for an SQL statement into **subsections**. Each subsection is executed by a different DB2 agent.

The access plan for an SQL statement generated by the DB2 code generator during compilation can be obtained using the **db2expln** or **dynexpln** commands (see the *Command Reference*). As an example, selecting all the rows from a table that is partitioned across several nodes might result in an access plan having two subsections:

1. Subsection 0, the coordinator subsection, whose role is to collect rows fetched by the other DB2 agents (subagents) and return them to the application.
2. Subsection 1, whose role is to perform a table scan and return the rows to the coordinating agent.

In this simple example, subsection 1 would be distributed across all the database partitions. There would be a subagent executing this subsection on each physical node of the **nodegroup** to which this table belongs. See *Administration Guide* for more information on these concepts.

The database system monitor allows you to correlate run-time information with the access plan, which is compile-time information. With inter-partition parallelism, it breaks information down to the subsection level. For example, when the statement monitor switch is ON, a GET SNAPSHOT FOR APPLICATION will return information for each subsection executing on this node, as well as totals for the statement.

The subsection information returned for an application snapshot includes:

- the number of table rows read/written
- CPU consumption
- elapsed time

- the number of tablequeue rows sent and received from other agents working on this statement. This allows you to track the execution of a long running query by taking a series of snapshots.
- subsection status. If the subsection is in a WAIT state, because it is waiting for another agent to send or receive data, then the information also identifies the node or nodes preventing the subsection from progressing in its execution. You may then take a snapshot on these nodes to investigate the situation.

The information logged by a statement event monitor for each subsection after it has finished executing includes: CPU consumption, total execution, time, and several other counters.

DB2 Productivity Tools

The database system monitor is a very powerful function of the DB2 database manager. It can be exploited to develop productivity tools for the database administrator (DBA) and database developer. The following are a few examples of productivity tools that use the function of the database system monitor, and are included with the DB2 product:

- db2batch

  An application that uses snapshot monitoring to collect metrics for tuning SQL queries. It can be found in sqllib/misc/db2batch. See the *Administration Guide* for more information.

- db2gov

  The DB2 governor is an application that uses snapshot monitoring to supervise the load and usage of the database manager. It provides the functions to FORCE or change the run-time priority of applications exceeding certain limits. These limits are specified by the DBA in the db2gov configuration file. Application limits and privileges can be expressed using several different parameters, for example maximum amount of CPU. It can be found in sqllib/adm/db2gov. See *Administration Guide* for more information.

- db2evmon

  An application that formats the data stream created by an event monitor. It can be found in sqllib/misc/db2evmon.

- Control Center

  A GUI for snapshot and event monitoring. For snapshots, it allows you to define performance variables in terms of the metrics returned by the database system monitor and graph them over time. For example, you can request that it take a snapshot and graph the progression of a performance variable over the last eight hours. **Alerts** can be set to notify the DBA when certain threshold are reached. For event monitors, it allows you to create, activate, start, stop, and delete event monitors. See the online help for the Control Center for more information.

- Event Analyzer

  A GUI for viewing file event monitor traces. Information collected on connections, deadlocks, overflows, transactions, statements, and subsections is organized and displayed in a tabular format. See the online help for the Event Analyzer for more information.

System Monitor Definitions

- Data Element

  A piece of information collected by the database system monitor. Snapshot and Event Monitors are the two different interfaces for accessing data elements. Some elements are only accessible with the event monitor (for example, connections involved in a deadlock), and others are only available with snapshot monitoring (for example, data maintained at the database manager level - total number of database agents).

- Basic Data Element

  Performance data that is always collected or maintained by the DB2 server, even when all monitor switches are off.

- Database System Monitor

  The function of the DB2 database manager that maintains information about its operation and performance.

- Event Analyzer

  A function of the DB2 administration GUI for viewing event monitor traces.

- Event Monitoring

  Using an event monitor to monitor database activity and performance by obtaining traces of the data collected when specific events occur (for example, the end of a transaction). It is requested from the DB2 server using an SQL interface.

- Monitor Switch

  Instructs the DB2 database manager to collect those data elements that involve a non-negligible processing or memory overhead for a group of monitor data.

- Monitored Level or Object

  An object about which the database system monitor can return performance, informational, or status data: for example, the database manager, a database, a connection, or a transaction.

- Monitoring Application

  An application using the snapshot API or an event monitor. Each monitoring application has its own logical view of the data collected by the DB2 server. This means that if a monitoring application resets its counters, it does not affect the counters collected by other monitoring applications. The reset API resets counters for snapshot monitoring. Turning off/on an event monitor resets its counters.

Counters and switches can only be reset for an event monitor by deactivating and reactivating it.

- Snapshot Monitoring

  Using the system monitor API to monitor database activity and performance by sampling the monitor data maintained by the DB2 server. This API consists of:

sqlmon()	set or query monitor switch settings
sqlmonrset()	reset system monitor counters
sqlmonss()	take a snapshot
sqlmonsz()	estimate the size of the data that would be returned for a particular invocation of sqlmonss()

- Snapshot Monitor

  A function of the Control Center GUI for snapshot monitoring. It provides the ability to follow trends, define performance variables in terms of data elements, and set alerts when thresholds for these variables are reached.

Chapter 3. Database System Monitor Data Elements

This chapter describes the information that is available from the database system monitor. The information returned by database system monitor falls into the following categories:

- identification for the database manager, an application, or a database connection being monitored.
- data primarily intended to help you to configure the system.
- database activity at various levels including application, table, or statement. This information can be used for activity monitoring, problem determination, and performance analysis. But it can also be used for configuration.

In this chapter, data elements are organized by their primary use category. When applicable, elements that have multiple uses may be referred to by associated elements in other categories. Multi-use data element information only appears in its main category, it is not duplicated in other categories. Refer to *data elements* in the Index, if you have trouble finding a data element.

The information is grouped as follows:

Note: For Extended Enterprise Edition users, snapshot elements only apply to the partition where the snapshot was issued.

How to Read the Data Element Tables

The section for each data element begins with a table that lists standard information. An example is shown in Figure 4, followed by an explanation of each part of the table.

(1) Snapshot Information Level	(2) API Structure(s)	(3) Monitor Switch
Database	sqlm_dbase	Sort
Application	sqlm_appl	Sort

(4) Resettable	Yes

(5) Event Type	(6) Event Record(s)
Database	sqlm_db_event
Connection	sqlm_conn_event
Statement	sqlm_stmt_event

(7) API Element Name	total_sorts
Element Type	counter

(8) Related Information	See Resettable See Switches See Sort Overflows

Figure 4. Sample Element Table

1. The level of information that can be captured by the snapshot monitor.
2. The API structures where captured snapshot information is returned.
3. The snapshot monitor switch that must be set to obtain this information.
4. Whether or not the counter can be reset (snapshot monitor only).
5. The event monitor must be created with this event type to collect this information.
6. The data structure where captured event information is returned.
7. The name and type of element, as returned in the API structure or event record.
8. References to related data elements.

This table is followed by a description of the element and information on how you can use it when monitoring your database.

Element Types

Data elements are classified by the following categories:

- Counter

  A counter counts the number of times an activity occurs. Counter values increase during monitoring. Most are resettable.

- Gauge

  A gauge indicates the current value for an item. This value can go up and down depending on database activity (for example, the number of locks held).

- Water mark

  A water mark indicates the highest (maximum) or lowest (minimum) value an element reached since monitoring was started. These are not resettable.

- Information

  An information element provides reference-type details of your monitoring activities. This can include items such as node names, aliases, and path details.

- Timestamp

  A timestamp indicates the date and time that an activity took place, by providing the number of seconds and microseconds that have elapsed since January 1, 1970. In the C language, for example, this can be converted to calendar date and time using the `ctime()` function.

- Time

  Time returns the number of seconds and microseconds spent on an activity.

Server Identification and Status

The following elements provide identification and status information about the server:

Start Database Manager Timestamp

Snapshot Information Level	API Structure(s)	Monitor Switch
Database Manager	sqlm_db2	Basic
Resettable	No	
API Element Name	db2start_time	
Element Type	timestamp	
Related Information	• "Snapshot Time" on page 193	

Description: The date and time that the database manager was started using the `db2start` command.

Usage: This element may be used with the *Snapshot Time* monitor element to calculate the elapsed time since the database manager was started up until the snapshot was taken.

Configuration NNAME at Monitoring (Server) Node

Snapshot Information Level	API Structure(s)	Monitor Switch
Database Manager	sqlm_collected	Basic
Resettable	No	
API Element Name	server_nname	
Element Type	information	
Related Information	• "Configuration NNAME of Client" on page 53	

Description: The name of the node being monitored by the database system monitor.

Usage: This element can be used to identify the database server node you are monitoring. This information can be useful if you are saving your monitor output in a file or database for later analysis and you need to differentiate the data from different database server nodes. This node name is determined based on the *nname* configuration parameter.

If you are using the database system monitor APIs, note that the API constant SQLM_IDENT_SZ is used to define the length of this element. Only the first 8 characters are currently used.

Server Instance Name

Snapshot Information Level	API Structure(s)	Monitor Switch
Database Manager	sqlm_collected	Basic
Resettable	No	
Event Type	**Event Record(s)**	
Event Log Header	sqlm_event_log_header	
API Element Name	server_instance_name	
Element Type	information	
Related Information	• "Configuration NNAME at Monitoring (Server) Node" on page 34	

Description: The name of the database manager instance for which the snapshot was taken.

Usage: If more than one instance of the database manager is present on the same system, this data item is used to uniquely identify the instance for which the snapshot call was issued. Along with *Configuration NNAME at Monitoring (Server) Node,* this information can be useful if you are saving your monitor output in a file or database for later analysis, and you need to differentiate the data from different instances of the database manager.

If you are using the database system monitor APIs, note that the API constant SQLM_IDENT_SZ is used to define the length of this element. Only the first 8 characters are currently used.

Database Manager Type at Monitored (Server) Node

Snapshot Information Level	API Structure(s)	Monitor Switch
Database Manager	sqlm_collected	Basic
Resettable	No	
API Element Name	server_db2_type	
Element Type	information	
Related Information	• "Configuration NNAME at Monitoring (Server) Node" on page 34	

Description: Identifies the type of database manager being monitored.

Usage: It contains one of the following types of configurations for the database manager:

API Symbolic Constant	Command Line Processor Output
sqlf_nt_server	Database Server with local and remote clients
sqlf_nt_stand_req	Database Server with local clients

The API symbolic constants are defined in the include file *sqlutil.h*.

Server Product/Version ID

Snapshot Information Level	API Structure(s)	Monitor Switch
Database Manager	sqlm_collected	Basic
Resettable	No	
Event Type	**Event Record(s)**	
Database Manager	sqlm_event_log_header	
API Element Name	server_prdid	
Element Type	information	
Related Information	• "Client Product/Version ID" on page 53	

Description: The product and version that is running on the server.

Usage: It is in the form PPPVVRRM, where:

PPP is SQL
VV identifies a 2-digit version number (with high-order 0 in the case of a 1-digit version)
RR identifies a 2-digit release number (with high-order 0 in the case of a 1-digit release)
M identifies a 1-digit modification level

If you are using the database system monitor APIs, note that the API constant SQLM_IDENT_SZ is used to define the length of this element. Only the first 8 characters are currently used.

Server Version

Snapshot Information Level	API Structure(s)	Monitor Switch
Database Manager	sqlm_collected	Basic
Resettable	No	
API Element Name	server_version	
Element Type	information	
Related Information	• "Server Product/Version ID" on page 36	

Description: The version of the server returning the information.

Usage: This field identifies the level of the database server collecting database system monitor information. This allows applications to interpret the data based on the level of the server returning the data. Valid values are:

SQLM_VERSION1 Data was returned by DB2 Version 1

SQLM_VERSION2 Data was returned by DB2 Version 2

SQLM_VERSION5 Data was returned by DB2 Universal Database Version 5

Service Level

Snapshot Information Level	API Structure(s)	Monitor Switch
Database Manager	sqlm_db2	Basic
Resettable	No	
API Element Name	service_level	
Element Type	information	
Related Information	• "Product Identification" on page 38	

Description: This is the current corrective service level of the server.

Usage: Used to provide information when requesting service or reporting a problem with DB2 on OS/2. This element will be blank for non-OS/2 systems.

Note: This element is similar to the *corr_serv_lvl* field in the *sqlestat* output. See Appendix C, "DB2 Version 1 sqlestat Users" on page 271 for more information on sqlestat equivalent data elements.

Server Operating System

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Basic
Resettable	No	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
API Element Name	server_platform	
Element Type	information	
Related Information	• "Client Operating Platform" on page 58 • "Database Location" on page 43	

Description: The operating system running the database server.

Usage: This element can be used for problem determination for remote applications. Values for this field can be found in the header file *sqlmon.h*.

Note: This element is similar to the *db_type* field in the *sqlestat* output. See Appendix C, "DB2 Version 1 sqlestat Users" on page 271 for more information on sqlestat equivalent data elements.

Product Name

Snapshot Information Level	API Structure(s)	Monitor Switch
Database Manager	sqlm_db2	Basic
Resettable	No	
API Element Name	product_name	
Element Type	information	
Related Information	• "Product Identification" on page 38 • "Service Level" on page 37	

Description: Details of the version of the server that is running.

Usage: Used to provide information when requesting service or reporting a problem with DB2 on OS/2. This element will be blank for non-OS/2 systems.

Note: This element is similar to the *product_name* field in the *sqlestat* output. See Appendix C, "DB2 Version 1 sqlestat Users" on page 271 for more information on sqlestat equivalent data elements.

Product Identification

Snapshot Information Level	API Structure(s)	Monitor Switch
Database Manager	sqlm_db2	Basic
Resettable	No	
API Element Name	component_id	
Element Type	information	
Related Information	• "Product Name" on page 38 • "Service Level" on page 37	

Description: Details of the type of the server that is running.

Usage: Used to provide information when requesting service or reporting a problem with DB2 on OS/2. This element will be blank for non-OS/2 systems.

Note: This element is similar to the *component_id* field in the *sqlestat* output. See Appendix C, "DB2 Version 1 sqlestat Users" on page 271 for more information on sqlestat equivalent data elements.

Status of DB2 Instance

Snapshot Information Level	API Structure(s)	Monitor Switch
Database Manager	sqlm_db2	Basic
Resettable	No	
API Element Name	db2_status	
Element Type	information	
Related Information	• "Status of Database" on page 42	

Description: The current status of the instance of the database manager.

Usage: You can use this element to determine the state of your database manager instance.

The value returned is always SQLM_DB2_ACTIVE.

Database Identification and Status

The following elements provide identification and status information about the database:

- "Database Name"
- "Database Path" on page 40
- "Database Activation Timestamp" on page 41
- "Database Deactivation Timestamp" on page 41
- "Status of Database" on page 42
- "Catalog Node Network Name" on page 42
- "Database Location" on page 43
- "Catalog Node Number" on page 43
- "Last Backup Timestamp" on page 44

Database Name

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Basic
Application	sqlm_appl_id_info	Basic
Table Space	sqlm_tablespace_header	Buffer Pool
	sqlm_bufferpool	Buffer Pool
Table	sqlm_table_header	Table
Lock	sqlm_dbase_lock	Basic
DCS Application	sqlm_dcs_applinfo	Basic
Resettable	No	
Event Type	**Event Record(s)**	
Database	sqlm_dbheader_event	
API Element Name	db_name	
Element Type	information	

Related Information	• "Resetting Monitor Data" on page 21 • "Last Reset Timestamp" on page 192 • "Input Database Alias" on page 193 • "Database Alias Used by Application" on page 54 • "Database Path" on page 40

Description: The real name of the database for which information is collected or to which the application is connected. This is the name the database was given when created.

Usage: You may use this element to identify the specific database to which the data applies.

For applications that are not using DDCS to connect to a DRDA host database, you can use this element in conjunction with the Database Path monitor element to uniquely identify the database and help relate the different levels of information provided by the monitor.

If you are using the database system monitor APIs, note that the API constant SQLM_IDENT_SZ is used to define the length of this element. Only the first 8 characters are currently used for DB2 databases, and the first 18 characters are used for DRDA host databases.

Database Path

Snapshot Information Level	**API Structure(s)**	**Monitor Switch**
Database	sqlm_dbase	Basic
Application	sqlm_appl_id_info	Basic
Table Space	sqlm_tablespace_header	Buffer Pool
	sqlm_bufferpool	Buffer Pool
Table	sqlm_table_header	Table
Lock	sqlm_dbase_lock	Basic
Resettable	No	
Event Type	**Event Record(s)**	
Database	sqlm_dbheader_event	
API Element Name	db_path	
Element Type	information	
Related Information	• "Resetting Monitor Data" on page 21 • "Input Database Alias" on page 193 • "Database Name" on page 39	

Description: The full path of the location where the database is stored on the monitored system.

Usage: This element can be used with the *Database Name* monitor element to identify the specific database to which the data applies.

Database Activation Timestamp

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Basic
Table Space	sqlm_tablespace_header	Buffer Pool
Table	sqlm_table_header	Basic
Resettable	No	
Event Type	**Event Record(s)**	
Database	sqlm_db_event sqlm_dbheader_event	
Connection	sqlm_connheader_event	
API Element Name	db_conn_time conn_time	
Element Type	timestamp	
Related Information	• "Resetting Monitor Data" on page 21 • "Connection Request Start Timestamp" on page 62 • "Snapshot Time" on page 193	

Description: The date and time of the connection to the database (at the database level, this is the first connection to the database), or when the activate database was issued.

Usage: Use this element with the Database Deactivation Timestamp monitor element to calculate the total connection time.

Database Deactivation Timestamp

Event Type	Event Record(s)
Database	sqlm_db_event
Connection	sqlm_conn_event
API Element Name	disconn_time
Element Type	timestamp
Related Information	• None

Description: The date and time that the application disconnected from the database (at the database level, this is the time the last application disconnected).

Usage: Use this element to calculate the elapsed time since:

- The database was active (for information at the database level)
- The connection was active (for information at the connection level).

Status of Database

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Basic
Resettable	No	
API Element Name	db_status	
Element Type	information	
Related Information	• "Status of DB2 Instance" on page 39	

Description: The current status of the database.

Usage: You can use this element to determine the state of your database.

Values for this field are:

API Constant	Description
SQLM_DB_ACTIVE	The database is active.
SQLM_DB_QUIESCE_PEND	The database is in quiesce-pending state. New connections to the database are **not** permitted and new units of work **cannot** be started. Depending on the quiesce request, active units of work will be allowed to complete or will be rolled back immediately.
SQLM_DB_QUIESCED	The database has been quiesced. New connections to the database are **not** permitted and new units of work **cannot** be started.
SQLM_DB_ROLLFWD	A rollforward is in progress on the database.

Catalog Node Network Name

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Basic
Resettable	No	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
API Element Name	catalog_node_name	
Element Type	information	
Related Information	• None	

Description: The network name of the catalog node. On OS/2, the netbios name of the server where the database is located.

Usage: Use this element to determine the location of a database.

Note: This element is similar to the *node* field in the *sqlestat* output. See Appendix C, "DB2 Version 1 sqlestat Users" on page 271 for more information on sqlestat equivalent data elements.

Database Location

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Basic
Resettable	No	
API Element Name	db_location	
Element Type	information	
Related Information	• "Server Operating System" on page 37	

Description: The location of the database in relation to the application.

Usage: Determine the relative location of the database server with respect to the application taking the snapshot. Values are:

- SQLM_LOCAL
- SQLM_REMOTE

Note: This element is similar to the *location* field in the *sqlestat* output. See Appendix C, "DB2 Version 1 sqlestat Users" on page 271 for more information on sqlestat equivalent data elements.

Catalog Node Number

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Basic
Resettable	No	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
API Element Name	catalog_node	
Element Type	information	
Related Information	• None	

Description: The node number of the node where the database catalog tables are stored.

Usage: The catalog node is the node where all system catalog tables are stored. All access to system catalog tables must go through this node. See the *Administration Guide* for information on system catalog tables.

Last Backup Timestamp

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Basic
Resettable	No	
API Element Name	last_backup	
Element Type	timestamp	
Related Information	• None	

Description: The date and time that the latest database backup was completed.

Usage: You may use this element to help you identify a database that has not been backed up recently, or to identify which database backup file is the most recent. If the database has never been backed up, this timestamp is initialized to zero.

Application Identification and Status

The following elements provide information about databases and their related applications.

- "Unit of Work Stop Timestamp" on page 64
- "Unit of Work Completion Status" on page 65
- "Previous Transaction Stop Time" on page 65
- "Application Idle Time" on page 66

Application Handle (agent ID)

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_appl_id_info	Basic
Lock	sqlm_appl_lock	Basic
DCS Application	sqlm_dcs_applinfo	Basic
Resettable	No	
Event Type	**Event Record(s)**	
Connection	sqlm_connheader_event	
Statement	sqlm_stmt_event	
	sqlm_subsection_event	
API Element Name	agent_id	
Element Type	information	
Related Information	• None	

Description: A system-wide **unique** ID for the application. On multi-node systems, where a database is partitioned, this ID will be the same on every node where the application may make a secondary connection.

Usage: The application handle can be used to uniquely identify an active application (application handle is synonymous with agent Id).

Note: The name agent_id is still used in the APIs, so that old applications can still be compiled with the Version 5 header files.

It can be used as input to GET SNAPSHOT commands that require an agent Id.

When reading event traces, it can be used to match event records with a given application.

It can be used as input to the FORCE APPLICATION command or API. On multi-node systems this command can be issued from any node where the application has a connection. Its effect is global

Application Status

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_appl_id_info	Basic
Lock	sqlm_appl_lock	Basic
Resettable	No	
API Element Name	appl_status	
Element Type	information	
Related Information	• "Application Status Change Time" on page 48 • "Statement Operation" on page 176	

Description: The current status of the application.

Usage: This element can help you diagnose potential application problems. Values for this field are:

API Constant	Description
SQLM_CONNECTPEND	**Database Connect Pending:** The application has initiated a database connection but the request has not yet completed.
SQLM_CONNECTED	**Database Connect Completed:** The application has initiated a database connection and the request has completed.
SQLM_UOWEXEC	**Unit of Work Executing:** The database manager is executing requests on behalf of the unit of work.
SQLM_UOWWAIT	**Unit of Work waiting:** The database manager is waiting on behalf of the unit of work in the application. This status typically means that the system is executing in the application's code.
SQLM_LOCKWAIT	**Lock Wait:** The unit of work is waiting for a lock. After the lock is granted, the status is restored to its previous value.
SQLM_COMMIT_ACT	**Commit Active:** The unit of work is committing its database changes.
SQLM_ROLLBACK_ACT	**Rollback Active:** The unit of work is rolling back its database changes.
SQLM_RECOMP	**Recompiling:** The database manager is recompiling (that is, rebinding) a plan on behalf of the application.
SQLM_COMP	**Compiling:** The database manager is compiling an SQL statement or precompiling a plan on behalf of the application.
SQLM_INTR	**Request Interrupted:** An interrupt of a request is in progress.

API Constant	Description
SQLM_DISCONNECTPEND	**Database Disconnect Pending:** The application has initiated a database disconnect but the command has not yet completed executing. The application may not have explicitly executed the database disconnect command. The database manager will disconnect from a database if the application ends without disconnecting.
SQLM_TPREP	**Transaction Prepared:** The unit of work is part of a global transaction that has entered the prepared phase of the two-phase commit protocol.
SQLM_THCOMT	**Transaction Heuristically Committed:** The unit of work is part of a global transaction that has been heuristically committed.
SQLM_THABRT	**Transaction Heuristically Rolled Back:** The unit of work is part of a global transaction that has been heuristically rolled-back.
SQLM_TEND	**Transaction Ended:** The unit of work is part of a global transaction that has ended but has not yet entered the prepared phase of the two-phase commit protocol.
SQLM_CREATE_DB	**Creating Database:** The agent has initiated a request to create a database and that request has not yet completed.
SQLM_RESTART	**Restarting Database:** The application is restarting a database in order to perform crash recovery.
SQLM_RESTORE	**Restoring Database:** The application is restoring a backup image to the database.
SQLM_BACKUP	**Backing Up Database:** The application is performing a backup of the database.
SQLM_LOAD	**Data Fast Load:** The application is performing a “fast load” of data into the database.
SQLM_UNLOAD	**Data Fast Unload:** The application is performing a “fast unload” of data from the database.
SQLM_IOERROR_WAIT	**Wait to Disable Table space:** The application has detected an I/O error and is attempting to disable a particular table space. The application has to wait for all other active transactions on the table space to complete before it can disable the table space.
SQLM_QUIESCE_TABLESPACE	**Quiescing a Table space:** The application is performing a quiesce table space request.

ID of Code Page Used by Application

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_appl_id_info	Basic
Lock	sqlm_appl_lock	Basic
DCS Application	sqlm_dcs_applinfo	Basic
Resettable	No	
Event Type	**Event Record(s)**	
Event Log Header	sqlm_event_log_header	
Connection	sqlm_connheader_event	
API Element Name	codepage_id	
Element Type	information	
Related Information	• None	

Description: The code page identifier.

Usage: For snapshot monitor data, this is the code page at the node where the monitored application started. This identifier may be used for problem determination for remote applications. You may use this information to ensure that data conversion is supported between the application code page and the database code page (or for DRDA host databases, the host CCSID). For information about supported code pages, see the *Administration Guide*.

For event monitor data, this is the code page of the database for which event data is collected. You can use this element to determine whether your event monitor application is running under a different code page from that used by the database. Data written by the event monitor uses the database code page. If your event monitor application uses a different code page, you may need to perform some character conversion to make the data readable.

Application Status Change Time

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_appl_id_info	Unit of Work
Lock	sqlm_appl_lock	Unit of Work
Resettable	No	
API Element Name	status_change_time	
Element Type	timestamp	
Related Information	• "Resetting Monitor Data" on page 21 • "Application Status" on page 46	

Description: The date and time the application entered its current status.

Usage: This element allows you to determine how long an application has been in its current status. If it has been in the same status for a long period of time, this may indicate that it has a problem.

Application Name

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_appl_id_info	Basic
Lock	sqlm_appl_lock	Basic
DCS Application	sqlm_dcs_applinfo	Basic
Resettable	No	
Event Type	**Event Record(s)**	
Connection	sqlm_connheader_event	
API Element Name	appl_name	
Element Type	information	
Related Information	• "Application ID" on page 50 • "ID of Code Page Used by Application" on page 48	

Description: The name of the application running at the client as known to the database manager or DDCS.

Usage: This element may be used with *Application ID* to relate data items with your application.

In a client/server environment, this name is passed from the client to the server to establish the database connection. For DRDA-AS connections, this name is the DRDA external name.

Note: Although the name of the application that is executing could be more than 20 bytes, only the first 20 bytes after the last path separator are used to set this element.

The application name is not available for applications running on the following down-level database client products:

- DB2 Version 1
- IBM Extended Services for OS/2

In situations where the client application code page is different from the code page under which the database system monitor is running, you can use*ID of Code Page Used by Application* to help translate *Application Name*.

Application ID

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_appl_id_info	Basic
DCS Application	sqlm_dcs_applinfo	Basic
Lock	sqlm_appl_lock	Basic
Resettable	No	
Event Type	**Event Record(s)**	
Connection	sqlm_conn_event sqlm_connheader_event	
Statement	sqlm_stmt_event	
Transaction	sqlm_xaction_event	
Deadlock	sqlm_dlconn_event	
API Element Name	appl_id	
Element Type	information	
Related Information	• "Outbound Application ID" on page 55 • "Client Communication Protocol" on page 58	

Description: This identifier is generated when the application connects to the database at the database manager or when DDCS receives a request to connect to a DRDA database.

Usage: This ID is known on both the client and server, so you can use it to correlate the client and server parts of the application. For DDCS applications, you will also need to use Outbound Application ID to correlate the client and server parts of the application.

This identifier is unique across the network. There are different formats for the application ID, which are dependent on the communication protocol between the client and the server machine on which the database manager and/or DDCS are running. Each of the formats consists of three parts separated by periods.

1. APPC

   Format `Network.LU Name.Application instance`

   Example `CAIBMTOR.OSFDBX0.930131194520`

   Details This application ID is the displayable format of an actual SNA LUWID (Logical Unit-of-Work ID) that flows on the network when an APPC conversation is allocated. APPC-generated application IDs are made up by concatenating the network name, the LU name, and the LUWID instance number, which create a unique label for the client/server application. The network name and LU name can each be a maximum of 8 characters. The application instance corresponds to the 12-decimal-character LUWID instance number.

2. TCP/IP

   Format `*TCPIP.IPAddr.Application instance`

   Example `*TCPIP.A12CF9E8.930131214645`

Details A TCP/IP-generated application ID is made up by concatenating the string "*TCPIP," the IP address in hexadecimal characters, and a unique identifier for the instance of this application. The IP address is a 32-bit number displayed as a maximum of 8 hexadecimal characters.

3. IPX/SPX

   Format `Netid.nodeid.Application instance`

   Example `C11A8E5C.400011528250.0131214645`

   Details An IPX/SPX-generated application ID is made up by concatenating a character network ID (8 hexadecimal characters), a node id (12 hexadecimal characters), and a unique identifier for the instance of the application. The application instance corresponds to a 10-decimal-character time stamp of the form `MMDDHHMMSS`.

4. NetBIOS

   Format `*NETBIOS.nname.Application instance`

   Example `*NETBIOS.SBOIVIN.930131214645`

   Details A NetBIOS application ID is made up by concatenating the string "*NETBIOS," the nname defined in the client's database configuration file, and a unique identifier for the instance of this application.

5. Local Applications

   Format `*LOCAL.DB2 instance.Application instance`

   Example `*LOCAL.DB2INST1.930131235945`

   Details The application ID generated for a local application is made up by concatenating the string `*LOCAL`, the name of the DB2 instance, and a unique identifier for the instance of this application.

Use *Client Communication Protocol* to determine which communications protocol the connection is using and, as a result, the format of the *application ID*.

Sequence Number

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_appl_id_info	Basic
DCS Application	sqlm_dcs_applinfo	Basic
Resettable	No	
Event Type	**Event Record(s)**	
Connection	sqlm_conn_event sqlm_connheader_event	
Statement	sqlm_stmt_event	
Transaction	sqlm_xaction_event	
Deadlock	sqlm_dlconn_event	
API Element Name	sequence_no	
Element Type	information	
Related Information	• None	

Description: This element is reserved for future use. In this release, its value always be “0001.” It may contain different values in future releases of the product.

Authorization ID

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_appl_id_info	Basic
Lock	sqlm_appl_lock	Basic
DCS Application	sqlm_dcs_applinfo	Basic
Resettable	No	
Event Type	**Event Record(s)**	
Connection	sqlm_connheader_event	
API Element Name	auth_id	
Element Type	information	
Related Information	• “Application Name” on page 49	

Description: The authorization ID of the user who invoked the application that is being monitored. On a DDCS gateway node, this is the user's authorization ID on the host.

Usage: You can use this element to determine who is performing the monitoring, and to uniquely identify the application that is running.

If you are using the database system monitor APIs, note that the API constant SQLM_IDENT_SZ is used to define the length of this element. Only the first 8 characters are currently used.

Configuration NNAME of Client

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_appl_id_info	Basic
DCS Application	sqlm_dcs_applinfo	Basic
Resettable	No	
Event Type	**Event Record(s)**	
Connection	sqlm_connheader_event	
API Element Name	client_nname	
Element Type	information	
Related Information	• "Configuration NNAME at Monitoring (Server) Node" on page 34	

Description: The *nname* in the database manager configuration file at the client node.

Usage: You can use this element to identify the client node that is running the application.

If you are using the database system monitor APIs, note that the API constant SQLM_IDENT_SZ is used to define the length of this element. Only the first 8 characters are currently used.

Client Product/Version ID

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_appl_id_info	Basic
DCS Application	sqlm_dcs_applinfo	Basic
Resettable	No	
Event Type	**Event Record(s)**	
Connection	sqlm_connheader_event	
API Element Name	client_prdid	
Element Type	information	
Related Information	• "Server Product/Version ID" on page 36	

Description: The product and version that is running on the client.

Usage: You can use this element to identify the product and code version of the database client. It is in the form PPPVVRRM, where:

- PPP identifies the product, which is "SQL" for the DB2 products
- VV identifies a 2-digit version number (with high-order 0 in the case of a 1-digit version)
- RR identifies a 2-digit release number (with high-order 0 in the case of a 1-digit release)
- M identifies a 1-digit modification level.

If you are using the database system monitor APIs, note that the API constant SQLM_IDENT_SZ is used to define the length of this element. Only the first 8 characters are currently used.

Database Alias Used by Application

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_appl_id_info	Basic
Lock	sqlm_appl_lock	Basic
DCS Application	sqlm_dcs_applinfo	Basic
Resettable	No	
Event Type	**Event Record(s)**	
Connection	sqlm_connheader_event	
API Element Name	client_db_alias	
Element Type	information	
Related Information	• All other database-level information • All other application-level information • “Last Reset Timestamp” on page 192 • “Input Database Alias” on page 193 • “Database Name” on page 39	

Description: The alias of the database provided by the application to connect to the database.

Usage: This element can be used to identify the actual database that the application is accessing. The mapping between this name and *Database Name* could be done by using the database directories at the client node and the database manager server node.

This is the alias defined within the database manager where the database connection request originated.

If you are using the database system monitor APIs, note that the API constant SQLM_IDENT_SZ is used to define the length of this element. Only the first 8 characters are currently used.

This element can also be used to help you determine the authentication type, since different database aliases can have different authentication types.

Host Product/Version ID

Snapshot Information Level	API Structure(s)	Monitor Switch
DCS Application	sqlm_dcs_applinfo	Basic
Resettable	No	
API Element Name	host_prdid	
Element Type	information	
Related Information	• None	

Description: The product and version that is running on the server.

Usage: Used to identify the product and code version of the DRDA host database product. It is in the form PPPVVRRM, where:

- PPP identifies the host DRDA product
  - ARI for DB2 for VSE and VM
  - DSN for DB2 for MVS/ESA
  - QSQ for DB2 for AS/400
  - SQL for other DB2 products.
- VV identifies a 2-digit version number (with high-order 0 in the case of a 1-digit version)
- RR identifies a 2-digit release number (with high-order 0 in the case of a 1-digit release)
- M identifies a 1-digit modification level

This field is defined as SQLM_IDENT_SZ characters long to allow for future expansion, but only the first 8 characters are presently used.

Outbound Application ID

Snapshot Information Level	API Structure(s)	Monitor Switch
DCS Application	sqlm_dcs_applinfo	Basic
Resettable	No	
API Element Name	outbound_appl_id	
Element Type	information	
Related Information	• "Application ID" on page 50	

Description: This identifier is generated when the application connects to the DRDA host database. It is used to connect the DDCS gateway to the host, while the *Application ID* is used to connect a client to the DDCS gateway.

Usage: You may use this element in conjunction with*Application ID* to correlate the client and server parts of the application information.

This identifier is unique across the network.

Format `Network.LU Name.Application instance`

Example `CAIBMTOR.OSFDBM0.930131194520`

Details This application ID is the displayable format of an actual SNA LUWID (Logical Unit-of-Work ID) that flows on the network when an APPC conversation is allocated. APPC-generated application IDs are made up by concatenating the network name, the LU name, and the LUWID instance number, which creates a unique label for the client/server application. The network name and LU name can each be a maximum of 8 characters. The application instance corresponds to the 12-decimal-character LUWID instance number.

Outbound Sequence Number

Snapshot Information Level	**API Structure(s)**	**Monitor Switch**
DCS Application	sqlm_dcs_applinfo	Basic
Resettable	No	
API Element Name	outbound_sequence_no	
Element Type	information	
Related Information	• None	

Description: This element is reserved for future use. In this release, its value will always be "0001." It may contain different values in future releases of the product.

User Login ID

Snapshot Information Level	**API Structure(s)**	**Monitor Switch**
Application	sqlm_applinfo sqlm_appl	Basic Basic
Resettable	No	
Event Type	**Event Record(s)**	
Connection	sqlm_connheader_event	
API Element Name	execution_id	
Element Type	information	
Related Information	• "Authorization ID" on page 52	

Description: The ID that the user specified when logging in to the operating system. This ID is distinct from Authorization ID, which the user specifies when connecting to the database.

Usage: You can use this element to determine who is running the application that you are monitoring.

For operating systems such as OS/2 that do not support the concept of "logging in," this field will be the same as Authorization ID .

If you are using the database system monitor APIs, note that the API constant SQLM_IDENT_SZ is used to define the length of this element. Only the first 8 characters are currently used.

DRDA Correlation Token

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_applinfo sqlm_appl	Basic Basic
Resettable	No	
Event Type	**Event Record(s)**	
Connection	sqlm_connheader_event	
API Element Name	corr_token	
Element Type	information	
Related Information	• None	

Description: The DRDA AS correlation token.

Usage: The DRDA correlation token is used for correlating the processing between the application server and the application requester. It is an identifier dumped into logs when errors arise, that you can use to identify the conversation that is in error. In some cases, it will be the LUWID of the conversation.

If communications are not using DRDA, this element is blank.

If you are using the database system monitor APIs, note that the API constant SQLM_APPLID_SZ is used to define the length of this element.

Client Process ID

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_applinfo sqlm_appl	Basic Basic
Resettable	No	
Event Type	**Event Record(s)**	
Connection	sqlm_connheader_event	
API Element Name	client_pid	
Element Type	information	
Related Information	• None	

Description: The process ID of the client application that made the connection to the database.

Usage: You can use this element to correlate monitor information such as CPU and I/O time to your client application.

In the case of a DRDA AS connection, this element will be set to 0.

Client Operating Platform

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_applinfo	Basic
	sqlm_appl	Basic
Resettable	No	
Event Type	**Event Record(s)**	
Connection	sqlm_connheader_event	
API Element Name	client_platform	
Element Type	information	
Related Information	• "Server Operating System" on page 37	

Description: The operating system on which the client application is running.

Usage: This element can be used for problem determination for remote applications. Values for this field can be found in the header file *sqlmon.h*.

Client Communication Protocol

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_applinfo	Basic
	sqlm_appl	Basic
Resettable	No	
Event Type	**Event Record(s)**	
Connection	sqlm_connheader_event	
API Element Name	client_protocol	
Element Type	information	
Related Information	• None	

Description: The communication protocol that the client application is using to communicate with the server.

Usage: This element can be used for problem determination for remote applications. Values for this field are:

API Constant	Communication Protocol
SQLM_PROT_UNKNOWN	(note 1)
SQLM_PROT_LOCAL	none (note 2)
SQLM_PROT_APPC	APPC
SQLM_PROT_TCPIP	TCP/IP
SQLM_PROT_IPXSPX	IPX/SPX
SQLM_PROT_NETBIOS	NETBIOS

Notes:

1. The client is communicating using an unknown protocol. This value will only be returned if future clients connect with a down-level server.
2. The client is running on the same node as the server and no communications protocol is in use.

Database Country Code

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_applinfo sqlm_appl	Basic Basic
Resettable	No	
Event Type	**Event Record(s)**	
Event Log Header	sqlm_event_log_header	
Connection	sqlm_connheader_event	
API Element Name	country_code	
Element Type	information	
Related Information	• None	

Description: The country code of the database for which the monitor data is collected. Country code information is recorded in the database configuration file (see the *Administration Guide*).

For DRDA AS connections, this element will be set to 0.

Application Agent Priority

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_appl	Basic
Resettable	No	
Event Type	**Event Record(s)**	
Connection	sqlm_conn_event	
API Element Name	appl_priority	
Element Type	information	
Related Information	• "Application Priority Type" on page 60	

Description: The priority of the agents working for this application.

Usage: You can use this element to check if applications are running with the expected priorities. Application priorities can be set by an administrator. They can be changed by the governor utility (**db2gov**).

The governor is used by DB2 to monitor and change the behavior of applications running against a database. This information is used to schedule applications and balance system resources.

A governor daemon collects statistics about the applications by taking snapshots. It checks these statistics against the rules governing applications running on that database. If the governor detects a rule violation, it takes the appropriate action. These rules and actions were specified by you in the governor configuration file.

If the action associated with a rule is to change an application's priority, the governor changes the priority of the agents in the partition where the violation was detected.

See the *Administration Guide* for more information on the governor.

Application Priority Type

Snapshot Information Level	**API Structure(s)**	**Monitor Switch**
Application	sqlm_appl	Basic
Resettable	No	
Event Type	**Event Record(s)**	
Connection	sqlm_conn_event	
API Element Name	appl_priority_type	
Element Type	information	
Related Information	• "Query Cost Estimate" on page 183 • "Application Agent Priority" on page 59	

Description: Operating system priority for the application that is running.

Usage: Dynamic priority is recalculated by the operating system based on usage. Static priority does not change.

User Authorization Level

Snapshot Information Level	**API Structure(s)**	**Monitor Switch**
Application	sqlm_appl sqlm_applinfo	Basic Basic
Resettable	No	
Event Type	**Event Record(s)**	
Connection	sqlm_conn_event	
API Element Name	authority_lvl	
Element Type	information	
Related Information	• None	

Description: The highest authority level granted to an application.

Usage: The operations allowed by an application are granted either directly or indirectly in the *sql.h*.

These are the authorizations granted explicitly to a user:

- SQL_SYSADMIN
- SQL_DBADM

- SQL_CREATETAB
- SQL_BINDADD
- SQL_CONNECT
- SQL_CREATE_NOT_FENC
- SQL_SYSCTRL
- SQL_SYSMAINT

The following are indirect authorizations inherited from group or public:

- SQL_SYSADM_GRP
- SQL_DBADM_GRP
- SQL_CREATETAB_GRP
- SQL_BINDADD_GRP
- SQL_CONNECT_GRP
- SQL_CREATE_NOT_FENC_GRP
- SQL_SYSCTRL_GRP
- SQL_SYSMAINT_GRP

See the *Administration Guide* for detailed information on authority levels.

Coordinating Node

Snapshot Information Level Application	**API Structure(s)** sqlm_appl	**Monitor Switch** Basic
Resettable	No	
Event Type Connection	**Event Record(s)** sqlm_conn_event	
API Element Name **Element Type**	coord_node information	
Related Information	• None	

Description: In a multi-node system, the node number of the node where the application connected or attached to the instance.

Usage: Each connected application is served by one coordinator node.

Connection Request Start Timestamp

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_appl	Basic
Resettable	No	
API Element Name **Element Type**	appl_con_time timestamp	
Related Information	• "Connection Request Completion Timestamp" on page 62 • All information related to the application	

Description: The date and time that an application started a connection request.

Usage: Use this element to determine when the application started its connection request to the database.

Connection Request Completion Timestamp

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_appl	Basic
Resettable	No	
API Element Name **Element Type**	conn_complete_time timestamp	
Related Information	• "Connection Request Start Timestamp" on page 62 • All information related to the application	

Description: The date and time that a connection request was granted.

Usage: Use this element to determine when a connection request to the database was granted.

Previous Unit of Work Completion Timestamp

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_appl	Unit of Work
Resettable	No	
API Element Name **Element Type**	prev_uow_stop_time timestamp	
Related Information	• "Resetting Monitor Data" on page 21 • "Unit of Work Start Timestamp" on page 63 • "Unit of Work Stop Timestamp" on page 64 • "Connection Request Completion Timestamp" on page 62	

Description: This is the time the unit of work completed.

Usage: You may use this element with *Unit of Work Stop Timestamp* to calculate the total elapsed time between COMMIT/ROLLBACK points, and with *Unit of Work Start*

Timestamp to calculate the time spent in the application between units of work. The time of one of the following:

- For applications currently within a unit of work, this is the time that the latest unit of work completed.
- For applications not currently within a unit of work (the application has completed a unit of work, but not yet started a new one), this is the stop time of the last unit of work that completed prior to the one that just completed. The stop time of the one just completed is indicated "Unit of Work Stop Timestamp" on page 64.
- For applications within their first unit of work, this is the database connection request completion time.

Unit of Work Start Timestamp

Snapshot Information Level Application	**API Structure(s)** sqlm_appl	**Monitor Switch** Unit of Work
Resettable	No	
API Element Name **Element Type**	uow_start_time timestamp	
Related Information	• "Resetting Monitor Data" on page 21 • "Unit of Work Stop Timestamp" on page 64 • "Previous Unit of Work Completion Timestamp" on page 62 • "Connection Request Completion Timestamp" on page 62	

Description: The date and time that the unit of work first required database resources.

Usage: This resource requirement occurs at the first SQL statement execution of that unit of work:

- For the first unit of work, it is the time of the first database request (SQL statement execution) after *Connection Request Completion Timestamp* .
- For subsequent units of work, it is the time of the first database request (SQL statement execution) after the previous COMMIT or ROLLBACK.

Note: The *SQL Reference* defines the boundaries of a unit of work as the COMMIT or ROLLBACK points.

The database system monitor excludes the time spent between the COMMIT/ROLLBACK and the next SQL statement from its definition of a unit of work. This measurement method reflects the time spent by the database manager in processing database requests, separate from time spent in application logic before the first SQL statement of that unit of work. The unit of work elapsed time does include the time spent running application logic between SQL statements within the unit of work.

You may use this element with *Unit of Work Stop Timestamp* to calculate the total elapsed time of the unit of work and with *Previous Unit of Work Completion Timestamp* to calculate the time spent in the application between units of work.

You can use the *Unit of Work Stop Timestamp* and the *Previous Unit of Work Completion Timestamp* to calculate the elapsed time for the SQL Reference's definition of a unit of work.

Unit of Work Stop Timestamp

Snapshot Information Level	**API Structure(s)**	**Monitor Switch**
Application	sqlm_appl	Unit of Work
Resettable	No	
API Element Name	uow_stop_time	
Element Type	timestamp	
Related Information	• "Resetting Monitor Data" on page 21 • "Unit of Work Start Timestamp" on page 63 • "Previous Unit of Work Completion Timestamp" on page 62 • "Connection Request Completion Timestamp" on page 62	

Description: The date and time that the most recent unit of work completed, which occurs when database changes are committed or rolled back.

Usage: You may use this element with *Previous Unit of Work Completion Timestamp* to calculate the total elapsed time between COMMIT/ROLLBACK points, and with *Unit of Work Start Timestamp* to calculate the elapsed time of the latest unit of work.

The timestamp contents will be set as follows:

- When the application has completed a unit of work and has not yet started a new one (as defined in *Unit of Work Start Timestamp).* this element will be a valid, non-zero timestamp
- When the application is currently executing a unit of work, this element will contain zeros
- When the application first connects to the database, this element is set to *Connection Request Completion Timestamp. .*

As a new unit of work is started, the contents of this element are moved to*Previous Unit of Work Completion Timestamp .*

Unit of Work Completion Status

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_appl	Unit of Work
Resettable	No	
Event Type	**Event Record(s)**	
Transaction	sqlm_xaction_event	
API Element Name	uow_comp_status (Snapshot) status (Event)	
Element Type	information	
Related Information	• "Resetting Monitor Data" on page 21	

Description: The status of the unit of work and how it stopped.

Usage: You may use this element to determine if the unit of work ended due to a deadlock or abnormal termination. It may have been:

- Committed due to a commit statement
- Rolled back due to a rollback statement
- Rolled back due to a deadlock
- Rolled back due to an abnormal termination
- Committed at normal application termination.

Note: API users should refer to the header file (*sqlmon.h*) containing definitions of database system monitor constants.

Previous Transaction Stop Time

Event Type	Event Record(s)
Transaction	sqlm_xaction_event
API Element Name	prev_stop_time
Element Type	timestamp
Related Information	• None

Description: The completion time of the last unit of work.

Usage: You may use this element to calculate the time spent in the application between units of work.

This is the unit of work that completed prior to the unit of work for which this transaction event is generated.

For applications within their first unit of work, this is the database connection request completion time.

Application Idle Time

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_appl	Statement
Resettable	No	
API Element Name	appl_idle_time	
Element Type	information	
Related Information	• "Query Cost Estimate" on page 183 • "Application Agent Priority" on page 59 • "Application Priority Type" on page 60	

Description: Number of seconds since an application has issued any requests to the server. This includes applications that have not terminated a transaction, for example not issued a commit or rollback.

Usage: This information can be used to implement applications that force users that have been idle for a specified number of seconds.

DB2 Agent Information

The following database system monitor elements provide information about agents:

- "Process or Thread ID"

Process or Thread ID

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_appl sqlm_agent	Basic Statement
Resettable	No	
API Element Name	coord_agent_pid	
Element Type	agent_pid information	
Related Information	• None	

Description: The process Id (UNIX systems) or thread Id (OS/2 or Windows systems) of a DB2 agent.

Usage: You can use this element to link database system monitor information to other sources of diagnostic information, such as system traces. You can also use it to monitor how agents working for a database application use system resources.

Database Manager Configuration

The following elements provide database manager configuration information.

Agents and Connections

Agents are how work gets done using the database manager. An agent is a process or thread that carries out the requests made by a client application. Each connected application is served by exactly 1 **coordinator agent** and possibly, a set of subordinator agents or **subagents**. Subagents are used for parallel SQL processing in partitioned databases and on SMP machines. Agents are classified as follows:

- **Coordinator agent**

  This is the initial agent to which a local or remote application connects. There is one coordinator agent dedicated to each database connection or instance attachment. The maximum number of coordinating agents per partition is controlled by the *max_coordagents* configuration parameter.

- **Subagent**

  In partitioned databases, additional agents can be enlisted by the coordinator agent to speed up SQL processing. Subagents are selected from the agent pool and are returned there when their work is done. The size of the agent pool is controlled by the *num_poolagents* configuration parameter.

- **Associated agent**

  A coordinator or subagent that is doing work for an application is associated with that application. After it is finished an application's work, it goes into the agent pool as an idle agent, but it remains associated with that application. If the application attempts to do more work, DB2 will search the agent pool for an agent already associated with the application and assign the work to it. If none is found, DB2 will attempt to get an agent to satisfy the request by:

  1. Choosing an idle agent that is not associated with an application.
  2. Creating an agent, if an idle agent is not available.
  3. Finding an agent that is associated with another application. For example, if an agent cannot be created because *maxagents* has been reached, DB2 will try to take an idle agent associated with another application. This is referred to as a **stolen agent**.

The *maxagents* configuration parameter defines the maximum number of agents, regardless of type, that can exist for an instance. The *maxagents* value does not create any agents. The initial number of agents that are created in the agent pool at DB2START is determined by the *num_initagents* configuration parameter.

Assuming no idle agents, each connection creates a new agent, unless *max_coordagents* has been reached. If subagents are not used, *max_coordagents* equals *maxagents*. If subagents are used, some combination of coordinator agents and subagents could reach *maxagents*.

When an agent is assigned work, it attempts to obtain a token or permission to process the transaction. The database manager controls the number of tokens available using the *maxcagents* configuration parameter. If a token is not available, the agent will sleep until one becomes available, at which time the requested work will be processed. This

allows you to use *maxcagents* to control the load, or number of concurrently executing transactions, the server handles.

The following elements provide agent and connection information:

Remote Connections To Database Manager

Snapshot Information Level	**API Structure(s)**	**Monitor Switch**
Database Manager	sqlm_db2	Basic
Resettable	No	
API Element Name	rem_cons_in	
Element Type	gauge	
Related Information	• "Remote Connections Executing in the Database Manager" on page 69 • "Local Connections" on page 69 • "Local Connections Executing in the Database Manager" on page 70	

Description: The total number of current connections initiated from remote clients to the instance of the database manager that is being monitored.

Usage: Shows the number of connections from remote clients to databases in this instance. This value will change frequently, so you may need to sample it at specific intervals over an extended period of time to get a realistic view of system usage. This number does not include applications that were initiated from the same instance as the database manager.

When used in conjunction with the Local Connections monitor element, these elements can help you adjust the setting of the *max_coordagents* configuration parameter, described in the *Administration Guide*.

Remote Connections Executing in the Database Manager

Snapshot Information Level	**API Structure(s)**	**Monitor Switch**
Database Manager	sqlm_db2	Basic
Resettable	No	
API Element Name **Element Type**	rem_cons_in_exec gauge	
Related Information	• "Remote Connections To Database Manager" on page 68 • "Local Connections" on page 69 • "Local Connections Executing in the Database Manager" on page 70	

Description: The number of remote applications that are currently connected to a database and are currently processing a unit of work within the database manager instance being monitored.

Usage: This number can help you determine the level of concurrent processing occurring on the database manager. This value will change frequently, so you may need to sample it at specific intervals over an extended period of time to get a realistic view of system usage. This number does not include applications that were initiated from the same instance as the database manager.

When used in conjunction with the Local Connections Executing in the Database Manager monitor element, this element can help you adjust the setting of the *maxcagents* configuration parameter, described in the *Administration Guide*.

Local Connections

Snapshot Information Level	**API Structure(s)**	**Monitor Switch**
Database Manager	sqlm_db2	Basic
Resettable	No	
API Element Name **Element Type**	local_cons gauge	
Related Information	• "Remote Connections To Database Manager" on page 68 • "Remote Connections Executing in the Database Manager" on page 69 • "Local Connections Executing in the Database Manager" on page 70	

Description: The number of local applications that are currently connected to a local database within the database manager instance being monitored.

Usage: This number can help you determine the level of concurrent processing occurring in the database manager. This value will change frequently, so you may need to sample it at specific intervals over an extended period of time to get a realistic view of system usage.

This number only includes applications that were initiated from the same instance as the database manager. The applications are connected, but may or may not be executing a unit of work in the database.

When used in conjunction with the Remote Connections To Database Manager monitor element, this element can help you adjust the setting of the *maxagents* configuration parameter, described in the *Administration Guide*.

Local Connections Executing in the Database Manager

Snapshot Information Level Database Manager	**API Structure(s)** sqlm_db2	**Monitor Switch** Basic
Resettable	No	
API Element Name **Element Type**	local_cons_in_exec gauge	
Related Information	• "Remote Connections To Database Manager" on page 68 • "Remote Connections Executing in the Database Manager" on page 69 • "Local Connections" on page 69	

Description: The number of local applications that are currently connected to a local database within the database manager instance being monitored and are currently processing a unit of work.

Usage: This number can help you determine the level of concurrent processing occurring in the database manager. This value will change frequently, so you may need to sample it at specific intervals over an extended period of time to get a realistic view of system usage. This number only includes applications that were initiated from the same instance as the database manager.

When used in conjunction with the Remote Connections Executing in the Database Manager monitor element, this element can help you adjust the setting of the *maxcagents* configuration parameter, described in the *Administration Guide*.

Local Databases with Current Connects

Snapshot Information Level	API Structure(s)	Monitor Switch
Database Manager	sqlm_db2	Basic
Resettable	No	
API Element Name	con_local_dbases	
Element Type	gauge	
Related Information	• None	

Description: The number of local databases that have applications connected.

Usage: This value gives an indication of how many database information records you can expect when gathering data at the database level.

The applications can be running locally or remotely, and may or may not be executing a unit of work within the database manager

Connects Since First Database Connect

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Basic
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
API Element Name	total_cons	
Element Type	counter	
Related Information	• "When Counters are Initialized" on page 20 • "Database Activation Timestamp" on page 41 • "Applications Connected Currently" on page 72 • "Applications Executing in the Database Currently" on page 72 • "Secondary Connections" on page 78	

Description: Indicates the number of connections to the database (coordinator agents).

Usage: You can use this element in conjunction with the Database Activation Timestamp and the Start Database Manager Timestamp monitor elements to calculate the frequency at which applications have connected to the database.

If the frequency of connects is low, you may want to do a dummy connect initially before connecting any other application, because of the extra overhead that sometimes accompanies the first connect to a database (for example, initial buffer pool allocation or autorestart). This will result in subsequent connects being processed at a higher rate. You could also use the ACTIVATE DATABASE command.

Note: When you reset this element, its value is set to the number of applications that are currently connected, not to zero.

Applications Connected Currently

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Basic
Lock	sqlm_dbase_lock	Basic
Resettable	No	
API Element Name	appls_cur_cons	
Element Type	gauge	
Related Information	• "Applications Executing in the Database Currently" on page 72 • "Connects Since First Database Connect" on page 71 • "Remote Connections To Database Manager" on page 68 • "Local Connections" on page 69	

Description: Indicates the number of applications that are currently connected to the database.

Usage: You may use this element to help you understand the level of activity within a database and the amount of system resource being used.

It can help you adjust the setting of the *maxappls* and *max_coordagents* configuration parameters, which are described in the *Administration Guide*. For example, its value is always the same as *maxappls*, you may want to increase the value of *maxappls*. See the *Remote Connections To Database Manager* and the *Local Connections* monitor elements for more information.

Applications Executing in the Database Currently

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Basic
Resettable	No	
API Element Name	appls_in_db2	
Element Type	gauge	
Related Information	• "Applications Connected Currently" on page 72 • "Connects Since First Database Connect" on page 71 • "Remote Connections Executing in the Database Manager" on page 69 • "Local Connections Executing in the Database Manager" on page 70 • "Current Agents Waiting On Locks" on page 144	

Description: Indicates the number of applications that are currently connected to the database, and for which the database manager is currently processing a request.

Usage: You can use this element to understand how many of the database manager agent tokens are being used by applications connected to this database. If the sum of

Remote Connections Executing in the Database Manager and *Local Connections Executing in the Database Manager* is equal to the value of the *maxcagents* configuration parameter, you may want to increase the value of that parameter, as described in the *Administration Guide*.

Agents Registered

Snapshot Information Level	API Structure(s)	Monitor Switch
Database Manager	sqlm_db2	Basic
Resettable	No	
API Element Name	agents_registered	
Element Type	gauge	
Related Information	• None	

Description: The number of agents registered in the database manager instance that is being monitored (coordinator agents and subagents).

Usage: You can use this element to help evaluate your setting for the *maxagents* configuration parameter.

Agents Waiting for a Token

Snapshot Information Level	API Structure(s)	Monitor Switch
Database Manager	sqlm_db2	Basic
Resettable	No	
API Element Name	agents_waiting_on_token	
Element Type	gauge	
Related Information	• "Agents Registered" on page 73	

Description: The number of agents waiting for a token so they can execute a transaction in the database manager.

Usage: You can use this element to help evaluate your setting for the *maxcagents* configuration parameter.

Each application has a dedicated coordinator agent to process database requests within the database manager. Each agent has to get a token before it can execute a transaction. The maximum number of agents that can execute database manager transactions is limited by the configuration parameter *maxcagents*. For more information about this parameter, see the *Administration Guide*.

Maximum Number of Agents Registered

Snapshot Information Level	API Structure(s)	Monitor Switch
Database Manager	sqlm_db2	Basic
Resettable	No	
API Element Name **Element Type**	agents_registered_top water mark	
Related Information	• "Agents Registered" on page 73 • "Maximum Number of Agents Waiting" on page 74	

Description: The maximum number of agents that the database manager has ever registered, at the same time, since it was started (coordinator agents and subagents).

Usage: You may use this element to help you evaluate your setting of the *maxagents* configuration parameter, described in the *Administration Guide*.

The number of agents registered at the time the snapshot was taken is recorded by Agents Registered.

Maximum Number of Agents Waiting

Snapshot Information Level	API Structure(s)	Monitor Switch
Database Manager	sqlm_db2	Basic
Resettable	No	
API Element Name **Element Type**	agents_waiting_top water mark	
Related Information	• "Agents Waiting for a Token" on page 73 • "Maximum Number of Agents Registered" on page 74	

Description: The maximum number of agents that have ever been waiting for a token, at the same time, since the database manager was started.

Usage: You may use this element to help you evaluate your setting of the *maxcagents* configuration parameter, described in the *Administration Guide*.

The number of agents waiting for a token at the time the snapshot was taken is recorded by *Agents Waiting for a Token.*

If the *maxcagents* parameter is set to its default value (-1), no agents should wait for a token and the value of this monitor element should be zero.

Number of Idle Agents

Snapshot Information Level	API Structure(s)	Monitor Switch
Database Manager	sqlm_db2	Basic
Resettable	No	
API Element Name	idle_agents	
Element Type	gauge	
Related Information	• "Maximum Number of Agents Registered" on page 74 • "Maximum Number of Agents Waiting" on page 74 • "Agents Registered" on page 73	

Description: The number of agents in the agent pool that are currently unassigned to an application and are, therefore, "idle."

Usage: You can use this element to help set the *num_poolagents* configuration parameter. Having idle agents available to service requests for agents can improve performance. See the *Administration Guide* for more information.

Agents Assigned From Pool

Snapshot Information Level	API Structure(s)	Monitor Switch
Database Manager	sqlm_db2	Basic
Resettable	No	
API Element Name	agents_from_pool	
Element Type	counter	
Related Information	• "Agents Created Due to Empty Agent Pool" on page 76 • "Maximum Number of Coordinating Agents" on page 76	

Description: The number of agents assigned from the agent pool.

Usage: This element can be used with *"Agents Created Due to Empty Agent Pool" on page 76* to determine how often an agent must be created because the pool is empty.

If the ratio of

```
Agents Created Due to Empty Agent Pool / Agents Assigned From Pool
```

is high, it may indicate that the *num_poolagents* configuration parameter should be increased. A low ratio suggests that *num_poolagents* is set too high, and that some of the agents in the pool are rarely used and are wasting system resources.

A high ratio can indicate that the overall workload for this node is too high. You can adjust the workload by lowering the maximum number of coordinating agents specified by the *maxcagents* configuration parameter, or by redistributing data among the nodes.

See the *Administration Guide* for more information on the Agent Pool Size (*num_poolagents*) and Maximum Number of Concurrent Coordinating Agents (*maxcagents*) configuration parameters.

Agents Created Due to Empty Agent Pool

Snapshot Information Level	**API Structure(s)**	**Monitor Switch**
Database Manager	sqlm_db2	Basic
Resettable	No	
API Element Name	agents_created_empty_pool	
Element Type	counter	
Related Information	• "Agents Assigned From Pool" on page 75 • "Maximum Number of Coordinating Agents" on page 76	

Description: The number of agents created because the agent pool was empty.

Usage: In conjunction with Agents Assigned From Pool, you can calculate the ratio of

```
Agents Created Due to Empty Agent Pool / Agents Assigned From Pool
```

See "Agents Assigned From Pool" on page 75 for information on using this element.

Maximum Number of Coordinating Agents

Snapshot Information Level	**API Structure(s)**	Basic
Database Manager	sqlm_db2	Basic
Database	sqlm_dbase	
Resettable	No	
API Element Name	coord_agents_top	
Element Type	water mark	
Related Information	• "Agents Assigned From Pool" on page 75 • "Agents Created Due to Empty Agent Pool" on page 76	

Description: The maximum number of coordinating agents working at one time.

Usage: If the peak number of coordinating agents represents too high a workload for this node, you can reduce the number that can be concurrently executing a transaction by changing the *maxcagents* configuration parameter.

See the *Administration Guide* for more information on the Maximum Number of Concurrent Coordinating Agents (*maxcagents*) configuration parameter.

Stolen Agents

Snapshot Information Level	API Structure(s)	Monitor Switch
Database Manager	sqlm_db2	Basic
Application	sqlm_appl	Basic
Resettable	Yes	
API Element Name	agents_stolen	
Element Type	counter	
Related Information	• "Number of Agents Working on a Statement" on page 190	

Description: The number of times that agents are stolen from an application. Agents are stolen when an idle agent associated with an application is reassigned to work on a different application.

Usage: This element can be used in conjunction with *"Maximum Number of Associated Agents"* to evaluate the load that this application places on the system.

Maximum Number of Associated Agents

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_appl	Basic
Resettable	No	
API Element Name	associated_agents_top	
Element Type	water mark	
Related Information	• "Agents Assigned From Pool" on page 75 • "Agents Created Due to Empty Agent Pool" on page 76	

Description: The maximum number of subagents associated with this application.

Usage: If the peak number of subagents is close to the *num_poolagents* configuration parameter, this might indicate too high a workload for this node.

See the *Administration Guide* for more information on the Agent Pool Size (*num_poolagents*) configuration parameter.

Committed Private Memory

Snapshot Information Level	API Structure(s)	Monitor Switch
Database Manager	sqlm_db2	Basic
Resettable	No	
API Element Name	comm_private_mem	
Element Type	gauge	
Related Information	• none	

Description: The amount of private memory that the instance of the database manager has currently committed at the time of the snapshot.

Usage: You can use this element to help set the *min_priv_mem* configuration parameter (see the *Administration Guide*) to ensure you have enough private memory available.

This element is only applicable to platforms where DB2 uses threads (such as OS/2).

Secondary Connections

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Basic
Resettable	No	
API Element Name	total_sec_cons	
Element Type	counter	
Related Information	• "Connects Since First Database Connect" on page 71 • "Start Database Manager Timestamp" on page 34 • "Database Activation Timestamp" on page 41	

Description: The number of connections made by a subagent to the database at the node.

Usage: You can use this element in conjunction with the Connects Since First Database Connect, Database Activation Timestamp, and the Start Database Manager Timestamp monitor elements to calculate the frequency at which applications have connected to the database.

Sort

The following elements provide information about the database manager sort work performed:

- "Total Sort Heap Allocated" on page 79
- "Post Threshold Sorts" on page 80
- "Piped Sorts Requested" on page 80
- "Piped Sorts Accepted" on page 81
- "Total Sorts" on page 82
- "Total Sort Time" on page 82

- “Sort Overflows” on page 83
- “Active Sorts” on page 84

Total Sort Heap Allocated

Snapshot Information Level	API Structure(s)	Monitor Switch
Database Manager	sqlm_db2	Basic
Database	sqlm_dbase	Basic
Resettable	No	
API Element Name	sort_heap_allocated	
Element Type	gauge	
Related Information	• “Total Sorts” on page 82	

Description: The total number of allocated pages of sort heap space for all sorts at the level chosen and at the time the snapshot was taken.

Usage: The amount of memory allocated for each sort may be some or all of the available sort heap size. Sort heap size is the amount of memory available for each sort as defined in the database configuration parameter *sortheap*.

It is possible for a single application to have concurrent sorts active. For example, in some cases a SELECT statement with a subquery can cause concurrent sorts.

Information may be collected at two levels:

- At the database manager level, it represents the sum of sort heap space allocated for all sorts in all active databases in the database manager
- At the database level, it represents the sum of the sort heap space allocated for all sorts in a database.

Normal memory estimates do not include sort heap space. If excessive sorting is occurring, the extra memory used for the sort heap should be added to the base memory requirements for running the database manager. Generally, the larger the sort heap, the more efficient the sort. Appropriate use of indexes can reduce the amount of sorting required.

You may use the information returned at the database manager level to help you tune the *sheapthres* configuration parameter. If the element value is greater than or equal to *sheapthres*, it means that the sorts are not getting the full sort heap as defined by the *sortheap* parameter.

Post Threshold Sorts

Snapshot Information Level	API Structure(s)	Monitor Switch
Database Manager	sqlm_db2	Sort
Resettable	Yes	
API Element Name **Element Type**	post_threshold_sorts counter	
Related Information	• "Resetting Monitor Data" on page 21 • "When Counters are Initialized" on page 20 • "Statement Sorts" on page 181 • "Active Sorts" on page 84	

Description: The number of sorts that have requested heaps after the sort heap threshold has been reached.

Usage: Under normal conditions, the database manager will allocate sort heap using the value specified by the *sortheap* configuration parameter. If the amount of memory allocated to sort heaps exceeds the sort heap threshold (*sheapthres* configuration parameter), the database manager will allocate sort heap using a value less than that specified by the *sortheap* configuration parameter.

Each active sort on the system allocates memory, which may result in sorting taking up too much of the system memory available. Sorts that start after the sort heap threshold has been reached may not receive an optimum amount of memory to execute, but, as a result, the entire system may benefit. By modifying the sort heap threshold and sort heap size configuration parameters, the performance of sort operations and/or the overall system can be improved. If this element's value is high, you can:

- Increase the sort heap threshold (*sheapthres*) or,
- Adjust applications to use fewer or smaller sorts via SQL query changes.

Piped Sorts Requested

Snapshot Information Level	API Structure(s)	Monitor Switch
Database Manager	sqlm_db2	Basic
Resettable	Yes	
API Element Name **Element Type**	piped_sorts_requested counter	
Related Information	• "When Counters are Initialized" on page 20 • "Piped Sorts Accepted" on page 81 • "Post Threshold Sorts" on page 80	

Description: The number of piped sorts that have been requested.

Usage: Each active sort on the system allocates memory, which may result in sorting taking up too much of the available system memory.

The sort list heap (*sortheap*) and sort heap threshold (*sheapthres*) configuration parameters help to control the amount of memory used for sort operations. These parameters are also used to determine whether a sort will be piped.

Since piped sorts may reduce disk I/O, allowing more piped sorts can improve the performance of sort operations and possibly the performance of the overall system. A piped sort is not be accepted if the sort heap threshold will be exceeded when the sort heap is allocated for the sort. See *Piped Sorts Accepted* for more information if you are experiencing piped sort rejections.

The SQL EXPLAIN output will show whether the optimizer requests a piped sort. For more information on piped and non-piped sorts see the *Administration Guide*.

Piped Sorts Accepted

Snapshot Information Level	**API Structure(s)**	**Monitor Switch**
Database Manager	sqlm_db2	Basic
Resettable	Yes	
API Element Name	piped_sorts_accepted	
Element Type	counter	
Related Information	• "When Counters are Initialized" on page 20 • "Piped Sorts Requested" on page 80 • "Post Threshold Sorts" on page 80	

Description: The number of piped sorts that have been accepted.

Usage: Each active sort on the system allocates memory, which may result in sorting taking up too much of the available system memory.

When the number of accepted piped sorts is low compared to the number requested, you can improve sort performance by adjusting one or both of the following configuration parameters:

- sortheap
- sheapthres

If piped sorts are being rejected, you might consider decreasing your sort heap or increasing your sort heap threshold. You should be aware of the possible implications of either of these options. If you increase the sort heap threshold, then there is the possibility that more memory will remain allocated for sorting. This could cause the paging of memory to disk. If you decrease the sort heap, you might require an extra merge phase that could slow down the sort.

See the *Administration Guide* for more information on sorts.

Total Sorts

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Sort
Application	sqlm_appl	Sort
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Connection	sqlm_conn_event	
Statement	sqlm_stmt_event	
API Element Name	total_sorts	
Element Type	counter	
Related Information	• "Resetting Monitor Data" on page 21 • "When Counters are Initialized" on page 20 • "Sort Overflows" on page 83 • "Total Sort Time" on page 82	

Description: The total number of sorts that have been executed.

Usage: At a database or application level, use this value with *Sort Overflows* to calculate the percentage of sorts that need more heap space. You can also use it with *Total Sort Time* to calculate the average sort time.

If the number of sort overflows is small with respect to the total sorts, then increasing the sort heap size may have little impact on performance, unless this buffer size is increased substantially.

At a statement level, use this element to identify statements which are performing large numbers of sorts. These statements may benefit from additional tuning to reduce the number of sorts. You can also use the SQL EXPLAIN statement to identify the number of sorts a statement performs. See the *Administration Guide* for more information.

Total Sort Time

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Sort
Application	sqlm_appl	Sort
	sqlm_stmt	Sort
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Connection	sqlm_conn_event	
Statement	sqlm_stmt_event	
API Element Name	total_sort_time	
Element Type	counter	
Related Information	• "Resetting Monitor Data" on page 21 • "When Counters are Initialized" on page 20 • "Total Sorts" on page 82 • "Total Sorts" on page 82	

Description: The total elapsed time (in milliseconds) for all sorts that have been executed.

Usage: At a database or application level, use this element with *Total Sorts* to calculate the average sort time, which can indicate whether or not sorting is an issue as far as performance is concerned.

At a statement level, use this element to identify statements that spend a lot of time sorting. These statements may benefit from additional tuning to reduce the sort time.

This count also includes sort time of temporary tables created during related operations. It provides information for one statement, one application, or all applications accessing one database.

When using data elements providing elapsed times, you should consider:

1. Elapsed times are affected by system load, so the more processes you have running, the higher this elapsed time value.
2. To calculate this data element at a database level, the database system monitor sums the application-level times. This can result in double counting elapsed times at a database level, since more than one application process can be running at the same time.

   To provide meaningful data from the database level, you should normalize the data to a lower level. For example:

   ```
   total sort time / total sorts
   ```

   provides information about the average elapsed time for each sort.

Sort Overflows

Snapshot Information Level	**API Structure(s)**	**Monitor Switch**
Database	sqlm_dbase	Sort
Application	sqlm_appl	Sort
	sqlm_stmt	Sort
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Connection	sqlm_conn_event	
Statement	sqlm_stmt_event	
API Element Name	sort_overflows	
Element Type	counter	
Related Information	• "Resetting Monitor Data" on page 21 • "When Counters are Initialized" on page 20 • "Total Sorts" on page 82	

Description: The total number of sorts that ran out of sort heap and may have required disk space for temporary storage.

Usage: At a database or application level, use this element in conjunction with *Total Sorts* to calculate the percentage of sorts that had to overflow to disk. If this percentage is high, you may want adjust the database configuration by increasing the value of *sortheap*.

At a statement level, use this element to identify statements that require large sorts. These statements may benefit from additional tuning to reduce the amount of sorting required.

When a sort overflows, additional overhead will be incurred because the sort will require a merge phase and can potentially require more I/O, if data needs to be written to disk.

This element provides information for one statement, one application, or all applications accessing one database.

Active Sorts

Snapshot Information Level	**API Structure(s)**	**Monitor Switch**
Database	sqlm_dbase	Basic
Resettable	No	
API Element Name	active_sorts	
Element Type	counter	
Related Information	• "Total Sort Heap Allocated" on page 79 • "Total Sorts" on page 82	

Description: The number of sorts in the database that currently have a sort heap allocated.

Usage: Use this value in conjunction with *Total Sort Heap Allocated* to determine the average sort heap space used by each sort. If the *sortheap* configuration parameter is substantially larger than the average sort heap used, you may be able to lower the value of this parameter. (See the *Administration Guide* for more details.)

This value includes heaps for sorts of temporary tables that were created during relational operations.

Fast Communication Manager

The following database system monitor elements provide information about the Fast Communication Manager (FCM):

- "FCM Buffers Currently Free" on page 85
- "Minimum FCM Buffers Free" on page 85
- "Message Anchors Currently Free" on page 86
- "Minimum Message Anchors" on page 86
- "Connection Entries Currently Free" on page 87
- "Minimum Connection Entries" on page 87
- "Request Blocks Currently Free" on page 88
- "Minimum Request Blocks" on page 88

- “Number of Nodes” on page 89
- “Connection Status” on page 89
- “Total FCM Buffers Sent” on page 90
- “Total FCM Buffers Received” on page 90

FCM Buffers Currently Free

Snapshot Information Level	API Structure(s)	Monitor Switch
Database Manager	sqlm_fcm	Basic
Resettable	No	
API Element Name	buff_free	
Element Type	gauge	
Related Information	• “Minimum FCM Buffers Free” on page 85	

Description: This element indicates the number of FCM buffers currently free.

Usage: Use the number of FCM buffers currently free in conjunction with the *fcm_num_buffers* configuration parameter to determine the current FCM buffer pool utilization. You can use this information to tune *fcm_num_buffers*.

Minimum FCM Buffers Free

Snapshot Information Level	API Structure(s)	Monitor Switch
Database Manager	sqlm_fcm	Basic
Resettable	No	
API Element Name	buff_free_bottom	
Element Type	water mark	
Related Information	• “FCM Buffers Currently Free” on page 85	

Description: The lowest number of free FCM buffers reached during processing.

Usage: Use this element in conjunction with the *fcm_num_buffers* configuration parameter to determine the maximum FCM buffer pool utilization. If buff_free_bottom is low, you should increase *fcm_num_buffers* to ensure that operations do not run out of FCM buffers. If buff_free_bottom is high, you can decrease *fcm_num_buffers* to conserve system resources.

Message Anchors Currently Free

Snapshot Information Level	API Structure(s)	Monitor Switch
Database Manager	sqlm_fcm	Basic
Resettable	No	
API Element Name	MA_free	
Element Type	gauge	
Related Information	• "Minimum Message Anchors" on page 86	

Description: This element indicates the number of message anchors currently free.

Usage: Use the number of message anchors currently free in conjunction with the *fcm_num_anchors* configuration parameter to determine the current message anchor utilization. You can use this information to tune *fcm_num_anchors*.

Minimum Message Anchors

Snapshot Information Level	API Structure(s)	Monitor Switch
Database Manager	sqlm_fcm	Basic
Resettable	No	
API Element Name	MA_free_bottom	
Element Type	water mark	
Related Information	• "Message Anchors Currently Free" on page 86	

Description: The lowest number of free message anchors reached during processing.

Usage: Use this element in conjunction with the *fcm_num_anchors* configuration parameter to determine the maximum message anchors utilization. If MA_free_bottom is low, you should increase *fcm_num_anchors* to ensure that operations do not run out of message anchors. If MA_free_bottom is high, you can decrease *fcm_num_anchors* to conserve system resources.

Connection Entries Currently Free

Snapshot Information Level	API Structure(s)	Monitor Switch
Database Manager	sqlm_fcm	Basic
Resettable	No	
API Element Name	CE_free	
Element Type	gauge	
Related Information	• "Minimum Connection Entries" on page 87	

Description: This element indicates the number of connection entries currently free.

Usage: Use the number of connection entries currently free in conjunction with the *fcm_num_connect* configuration parameter to determine the current connection entry utilization. You can use this information to tune *fcm_num_connect*.

Minimum Connection Entries

Snapshot Information Level	API Structure(s)	Monitor Switch
Database Manager	sqlm_fcm	Basic
Resettable	No	
API Element Name	CE_free_bottom	
Element Type	water mark	
Related Information	• "Connection Entries Currently Free" on page 87	

Description: The lowest number of free connection entries reached during processing.

Usage: Use this element in conjunction with the *fcm_num_connect* configuration parameter to determine the maximum connection entry utilization. If CE_free_bottom is low, you should increase *fcm_num_connect* to ensure that operations do not run out of connection entries. If CE_free_bottom is high, you can decrease *fcm_num_connect* to conserve system resources.

Request Blocks Currently Free

Snapshot Information Level	API Structure(s)	Monitor Switch
Database Manager	sqlm_fcm	Basic
Resettable	No	
API Element Name	RB_free	
Element Type	gauge	
Related Information	• "Request Blocks Currently Free" on page 88	

Description: This element indicates the number of request blocks currently free.

Usage: Use the number of request blocks currently free in conjunction with the *fcm_num_rqb* configuration parameter to determine the current request block utilization. You can use this information to tune *fcm_num_rqb*.

Minimum Request Blocks

Snapshot Information Level	API Structure(s)	Monitor Switch
Database Manager	sqlm_fcm	Basic
Resettable	No	
API Element Name	RB_free_bottom	
Element Type	water mark	
Related Information	• "Request Blocks Currently Free" on page 88	

Description: The lowest number of free request blocks reached during processing.

Usage: Use this element in conjunction with the *fcm_num_rqb* configuration parameter to determine the maximum request block utilization. If RB_free_bottom is low, you should increase *fcm_num_rqb* to ensure that operations do not run out of request blocks. If RB_free_bottom is high, you can decrease *fcm_num_rqb* to conserve system resources.

Number of Nodes

Snapshot Information Level	API Structure(s)	Monitor Switch
Database Manager	sqlm_fcm	Basic
Resettable	No	
API Element Name	number_nodes	
Element Type	information	
Related Information	• None	

Description: The number of nodes in the current configuration.

Usage: Use this element to determine the number of *sqlm_fcm_node* structures that will be returned.

Connection Status

Snapshot Information Level	API Structure(s)	Monitor Switch
Database Manager	sqlm_fcm_node	Basic
Resettable	No	
API Element Name	connection_status	
Element Type	information	
Related Information	• "Total FCM Buffers Sent" on page 90 • "Total FCM Buffers Received" on page 90	

Description: This element indicates the status of the communication connection status between the node issuing the GET SNAPSHOT command and other nodes listed in the *db2nodes.cfg* file.

Usage: The connection values are :

SQLM_FCM_CONNECT_INACTIVE	No current connection
SQLM_FCM_CONNECT_ACTIVE	Connection is active
SQLM_FCM_CONNECT_CONGESTED	Connection is congested

Two nodes can be active, but the communication connection between them will remain inactive, unless there is some communication between those nodes.

Total FCM Buffers Sent

Snapshot Information Level	API Structure(s)	Monitor Switch
Database Manager	sqlm_fcm_node	Basic
Resettable	No	
API Element Name **Element Type**	total_buffers_sent counter	
Related Information	• "Connection Status" on page 89 • "Total FCM Buffers Received" on page 90	

Description: The total number of FCM buffers that have been sent from the node issuing the GET SNAPSHOT command to the node identified by the *node_number* (see the *db2nodes.cfg* file).

Usage: You can use this element to measure the level of traffic between the current node and the remote node. If the total number of FCM buffers sent to this node is high, you may want to redistribute the database, or move tables to reduce the inter-node traffic.

Total FCM Buffers Received

Snapshot Information Level	API Structure(s)	Monitor Switch
Database Manager	sqlm_fcm_node	Basic
Resettable	No	
API Element Name **Element Type**	total_buffers_rcvd counter	
Related Information	• "Connection Status" on page 89 • "Total FCM Buffers Sent" on page 90	

Description: The total number of FCM buffers received by the node issuing the GET SNAPSHOT command from the node identified by the *node_number* (see the *db2nodes.cfg* file).

Usage: You can use this element to measure the level of traffic between the current node and the remote node. If the total number of FCM buffers received from this node is high, you may want to redistribute the database, or move tables to reduce the inter-node traffic.

Database Configuration

The following elements provide information particularly helpful for database configuration.

Buffer Pool Activity

The database server reads and updates all data from a buffer pool. Data is copied from disk to a buffer pool as it is required by applications.

Pages are placed in a buffer pool:

- by the agent. This is synchronous I/O.
- by the I/O servers (prefetchers). This is asynchronous I/O.

Pages are written to disk from a buffer pool:

- by the agent, synchronously
- by page cleaners, asynchronously

If the server needs to read a page of data, and that page is already in the buffer pool, then the ability to access that page is much faster than if the page had to be read from disk. It is desirable to **hit** as many pages as possible in the buffer pool. Avoiding disk I/O is the main issue when trying to improve the performance of your server And so, proper configuration of the buffer pools are probably the most important consideration for performance tuning.

Buffer Pool Hit Ratio

The buffer pool hit ratio indicates the percentage of time that the database manager did not need to load a page from disk in order to service a page request. That is, the page was already in the buffer pool. The greater the buffer pool hit ratio, the lower the frequency of disk I/O.

The buffer pool hit ratio can be calculated as follows:

```
(1 - ((pool_data_p_reads + pool_index_p_reads) /
  (pool_data_l_reads + pool_index_l_reads))) * 100%
```

This calculation takes into account all of the pages (index and data) that are cached by the buffer pool.

For a large database, increasing the buffer pool size may have minimal effect on the buffer pool hit ratio. Its number of data pages may be so large, that the statistical chances of a hit are not improved increasing its size. But you might find that tuning the index buffer pool hit ratio achieves the desired result. This can be achieved using two methods:

1. Split the data and indices into two different buffer pools and tune them separately.
2. Use one buffer pool, but increase its size until the index hit ratio stops increasing. The index buffer pool hit ratio can be calculated as follows:

   ```
   (1 - ((pool_index_p_reads) / (pool_index_l_reads))) * 100%
   ```

The first method is often more effective, but because it requires indices and data to reside in different tablespaces, it may not be an option for existing databases. It also requires tuning two buffer pools instead of one, which can be a more difficult task, particularly when memory is constrained.

Prefetchers

You should also consider the impact that prefetchers may be having on the hit ratio. Prefetchers read data pages into the buffer pool anticipating their need by an application (asynchronously). In most situations, these pages are read just before they are needed (the desired case). However, prefetchers can cause unnecessary I/O by reading pages into the buffer pool that will not be used. For example, an application starts reading through a table. This is detected and prefetching starts, but the application fills an application buffer and stops reading. Meanwhile, prefetching has been done for a number of additional pages. I/O has occurred for pages that will not be used and the buffer pool is partially taken up with those pages.

Page Cleaners

Page cleaners monitor the buffer pool and asynchronously write pages to disk. Their goals are:

- Ensure that agents will always find free pages in the buffer pool. If an agent does not find free pages in the buffer pool, it must clean them itself, and the associated application will have a poorer response.
- Speed database recovery, if a system crash occurs. The more pages that have been written to disk, the smaller the number of log file records that must be processed to recover the database.

Although dirty pages are written out to disk, the pages are not removed from the buffer pool right away, unless the space is needed to read in new pages.

Note: Buffer pool information is typically gathered at a table space level, but the facilities of the database system monitor can roll this information up to the buffer pool and database levels. Depending on your type of analysis, you may need to examine this data at any or all of these levels.

The following elements provide information about buffer pool activity. For an overview how the database manager uses buffer pools, see the *Administration Guide*.

- "Buffer Pool Data Logical Reads" on page 93
- "Buffer Pool Data Physical Reads" on page 94
- "Buffer Pool Data Writes" on page 95
- "Buffer Pool Index Logical Reads" on page 96
- "Buffer Pool Index Physical Reads" on page 97
- "Buffer Pool Index Writes" on page 98
- "Total Buffer Pool Physical Read Time" on page 99
- "Total Buffer Pool Physical Write Time" on page 100
- "Database Files Closed" on page 100
- "Buffer Pool Asynchronous Data Reads" on page 101
- "Buffer Pool Asynchronous Data Writes" on page 102
- "Buffer Pool Asynchronous Index Writes" on page 103
- "Buffer Pool Asynchronous Index Reads" on page 104
- "Buffer Pool Asynchronous Read Time" on page 105
- "Buffer Pool Asynchronous Write Time" on page 106
- "Buffer Pool Asynchronous Read Requests" on page 107
- "Buffer Pool Log Space Cleaners Triggered" on page 107

- “Buffer Pool Victim Page Cleaners Triggered” on page 108
- “Buffer Pool Threshold Cleaners Triggered” on page 109
- “Buffer Pool Information” on page 109
- “Time Waited for Prefetch” on page 110

Buffer Pool Data Logical Reads

Snapshot Information Level	**API Structure(s)**	**Monitor Switch**
Database	sqlm_dbase	Buffer Pool
Table Space	sqlm_tablespace	Buffer Pool
	sqlm_bp_info	Buffer Pool
Application	sqlm_appl	Buffer Pool
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Table Space	sqlm_tablespace_event	
Connection	sqlm_conn_event	
API Element Name	pool_data_l_reads	
Element Type	counter	
Related Information	• “Resetting Monitor Data” on page 21 • “When Counters are Initialized” on page 20 • “Database Activation Timestamp” on page 41 • “Connection Request Start Timestamp” on page 62 • “Buffer Pool Data Physical Reads” on page 94 • “Buffer Pool Data Writes” on page 95 • “Buffer Pool Index Logical Reads” on page 96 • “Buffer Pool Index Physical Reads” on page 97	

Description: Indicates the number of logical read requests for data pages that have gone through the buffer pool.

Usage: This count includes accesses to data that is:

- Already in the buffer pool when the database manager needs to process the page
- Read into the buffer pool before the database manager can process the page.

In conjunction with *Buffer Pool Data Physical Reads,* you can calculate the data page hit ratio for the buffer pool using the following formula:

```
1 - (buffer pool data physical reads / buffer pool data logical reads)
```

In conjunction with Buffer Pool Data Physical Reads, Buffer Pool Index Physical Reads, and Buffer Pool Index Logical Reads, you can calculate the overall buffer pool hit ratio using the following formula:

```
1 - ((buffer pool data physical reads + buffer pool index physical reads)
    / (buffer pool data logical reads + buffer pool index logical reads))
```

Increasing buffer pool size will generally improve the hit ratio, but you will reach a point of diminishing return. Ideally, if you could a buffer pool large enough to store your entire database, then once the system is up and running you would get a hit ratio of 100%.

However, this is realistic in most cases. the significance of the hit ratio really depends on the size of you data, and the way it is accessed. A very large database where data is accessed evenly would have a poor hit ratio. There is little you can do with very large tables. In such case, you would focus your attention on smaller, frequently accessed tables, and on the indices. Perhaps, assigning them to an individual buffer pools, for which you can aim for higher hit ratios.

Buffer Pool Data Physical Reads

Snapshot Information Level	**API Structure(s)**	**Monitor Switch**
Database	sqlm_dbase	Buffer Pool
Table Space	sqlm_tablespace	Buffer Pool
	sqlm_bp_info	Buffer Pool
Application	sqlm_appl	Buffer Pool
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Table Space	sqlm_tablespace_event	
Connection	sqlm_conn_event	
API Element Name	pool_data_p_reads	
Element Type	counter	
Related Information	• "Resetting Monitor Data" on page 21 • "When Counters are Initialized" on page 20 • "Database Activation Timestamp" on page 41 • "Connection Request Start Timestamp" on page 62 • "Buffer Pool Data Logical Reads" on page 93 • "Buffer Pool Data Writes" on page 95 • "Buffer Pool Index Logical Reads" on page 96 • "Buffer Pool Index Physical Reads" on page 97 • "Buffer Pool Asynchronous Data Reads" on page 101	

Description: The number of read requests that required I/O to get data pages into the buffer pool.

Usage: See Buffer Pool Data Logical Reads and Buffer Pool Asynchronous Data Reads for information about how to use this element.

Buffer Pool Data Writes

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Buffer Pool
Table Space	sqlm_tablespace	Buffer Pool
	sqlm_bp_info	Buffer Pool
Application	sqlm_appl	Buffer Pool
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Table Space	sqlm_tablespace_event	
Connection	sqlm_conn_event	
API Element Name	pool_data_writes	
Element Type	counter	
Related Information	• "Resetting Monitor Data" on page 21 • "When Counters are Initialized" on page 20 • "Database Activation Timestamp" on page 41 • "Connection Request Start Timestamp" on page 62 • "Buffer Pool Data Logical Reads" on page 93 • "Buffer Pool Data Physical Reads" on page 94 • "Total Buffer Pool Physical Write Time" on page 100 • "Buffer Pool Asynchronous Data Writes" on page 102	

Description: Indicates the number of times a buffer pool data page was physically written to disk.

Usage: If a buffer pool data page is written to disk for a high percentage of the Buffer Pool Data Physical Reads, you may be able to improve performance by increasing the number of buffer pool pages available for the database.

A buffer pool data page is written to disk for the following reasons:

- To free a page in the buffer pool so another page can be read
- To flush the buffer pool.

The system does not always write a page to make room for a new one. If the page has not been updated, it can simply be replaced. This replacement is not counted for this element.

The data page can be written by an asynchronous page-cleaner agent before the buffer pool space is required. These asynchronous page writes are included in the value of this element in addition to synchronous page writes (see Buffer Pool Asynchronous Data Writes).

When calculating this percentage, disregard the number of physical reads required to initially fill the buffer pool. To determine the number of pages written:

1. Run your application (to load the buffer)
2. Note the value of this element
3. Run your application again

4. Subtract the value recorded in step 2 from the new value of this element.

In order to prevent the buffer pool from being deallocated between the runnings of your application, you should either;

- activate the database with the ACTIVATE DATABASE command
- have an idle application connected to the database.

If all applications are updating the database, increasing the size of the buffer pool may not have much impact on performance since most of the buffer pool pages contain updated data, which must be written to disk. However, if the updated pages can be used by other units of work before being written out, the buffer pool can save a write and a read, which will improve your performance.

See the *Administration Guide* for more information about buffer pool size.

Buffer Pool Index Logical Reads

Snapshot Information Level	**API Structure(s)**	**Monitor Switch**
Database	sqlm_dbase	Buffer Pool
Table Space	sqlm_tablespace	Buffer Pool
	sqlm_bp_info	Buffer Pool
Application	sqlm_appl	Buffer Pool
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Table Space	sqlm_tablespace_event	
Connection	sqlm_conn_event	
API Element Name	pool_index_l_reads	
Element Type	counter	
Related Information	• "Resetting Monitor Data" on page 21 • "When Counters are Initialized" on page 20 • "Database Activation Timestamp" on page 41 • "Connection Request Start Timestamp" on page 62 • "Buffer Pool Index Physical Reads" on page 97 • "Buffer Pool Index Writes" on page 98 • "Buffer Pool Data Physical Reads" on page 94 • "Buffer Pool Data Writes" on page 95 • "Buffer Pool Data Logical Reads" on page 93	

Description: Indicates the number of logical read requests for index pages that have gone through the buffer pool.

Usage: This count includes accesses to index pages that are:

- Already in the buffer pool when the database manager needs to process the page
- Read into the buffer pool before the database manager can process the page.

In conjunction with Buffer Pool Index Physical Reads, you can calculate the index page hit ratio for the buffer pool using one of the following:

```
1 - (buffer pool index physical reads / buffer pool index logical reads)
```

To calculate the overall buffer pool hit ratio, see Buffer Pool Data Logical Reads.

If the hit ratio is low, increasing the number of buffer pool pages may improve performance. See the *Administration Guide* for more information about buffer pool size.

Buffer Pool Index Physical Reads

Snapshot Information Level	**API Structure(s)**	**Monitor Switch**
Database	sqlm_dbase	Buffer Pool
Table Space	sqlm_tablespace	Buffer Pool
	sqlm_bp_info	Buffer Pool
Application	sqlm_appl	Buffer Pool
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Table Space	sqlm_tablespace_event	
Connection	sqlm_conn_event	
API Element Name	pool_index_p_reads	
Element Type	counter	
Related Information	• "Resetting Monitor Data" on page 21 • "When Counters are Initialized" on page 20 • "Database Activation Timestamp" on page 41 • "Connection Request Start Timestamp" on page 62 • "Buffer Pool Index Logical Reads" on page 96 • "Buffer Pool Index Writes" on page 98 • "Buffer Pool Data Logical Reads" on page 93 • "Buffer Pool Data Physical Reads" on page 94	

Description: Indicates the number of physical read requests to get index pages into the buffer pool.

Usage: See Buffer Pool Index Logical Reads for information about how to use this element.

Buffer Pool Index Writes

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Buffer Pool
Table Space	sqlm_tablespace	Buffer Pool
	sqlm_bp_info	Buffer Pool
Application	sqlm_appl	Buffer Pool
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Table Space	sqlm_tablespace_event	
Connection	sqlm_conn_event	
API Element Name	pool_index_writes	
Element Type	counter	
Related Information	• "Resetting Monitor Data" on page 21 • "When Counters are Initialized" on page 20 • "Database Activation Timestamp" on page 41 • "Connection Request Start Timestamp" on page 62 • "Buffer Pool Index Logical Reads" on page 96 • "Buffer Pool Index Physical Reads" on page 97 • "Buffer Pool Asynchronous Index Writes" on page 103	

Description: Indicates the number of times a buffer pool index page was physically written to disk.

Usage: Like a data page, a buffer pool index page is written to disk for the following reasons:

- To free a page in the buffer pool so another page can be read
- To flush the buffer pool.

The system does not always write a page to make room for a new one. If the page has not been updated, it can simply be replaced. This replacement is not counted for this element.

The index page can be written by an asynchronous page-cleaner agent before the buffer pool space is required. These asynchronous index page writes are included in the value of this element in addition to synchronous index page writes (see Buffer Pool Asynchronous Index Writes).

If a buffer pool index page is written to disk for a high percentage of the *Buffer Pool Index Physical Reads* , you may be able to improve performance by increasing the number of buffer pool pages available for the database.

When calculating this percentage, disregard the number of physical reads required to initially fill the buffer pool. To determine the number of pages written:

1. Run your application (to load the buffer)
2. Note the value of this element
3. Run your application again

4. Subtract the value recorded in step 2 from the new value of this element.

In order to prevent the buffer pool from being deallocated between the runnings of your application, you should either:

- activate the database with the ACTIVATE DATABASE command
- have an idle application connected to the database.

If all applications are updating the database, increasing the size of the buffer pool may not have much impact on performance, since most of the pages contain updated data which must be written to disk.

See the *Administration Guide* for more information about buffer pool size.

Total Buffer Pool Physical Read Time

Snapshot Information Level	**API Structure(s)**	**Monitor Switch**
Database	sqlm_dbase	Buffer Pool
Table Space	sqlm_tablespace	Buffer Pool
	sqlm_bp_info	Buffer Pool
Application	sqlm_appl	Buffer Pool
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Table Space	sqlm_tablespace_event	
Connection	sqlm_conn_event	
API Element Name	pool_read_time	
Element Type	counter	
Related Information	• "Resetting Monitor Data" on page 21 • "When Counters are Initialized" on page 20 • "Buffer Pool Data Physical Reads" on page 94 • "Buffer Pool Index Physical Reads" on page 97 • "Database Activation Timestamp" on page 41 • "Connection Request Start Timestamp" on page 62 • "Buffer Pool Asynchronous Read Time" on page 105	

Description: Provides the total amount of elapsed time spent processing read requests that caused data or index pages to be physically read from disk to buffer pool.

Usage: You can use this element with Buffer Pool Data Physical Reads and Buffer Pool Index Physical Reads to calculate the average page-read time. This average is important since it may indicate the presence of an I/O wait, which in turn may indicate that you should be moving data to a different device.

At the database and table space levels, this element includes the value of Buffer Pool Asynchronous Read Time.

Total Buffer Pool Physical Write Time

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Buffer Pool
Table Space	sqlm_tablespace	Buffer Poo
	sqlm_bp_info	Buffer Pooll
Application	sqlm_appl	Buffer Pool
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Table Space	sqlm_tablespace_event	
Connection	sqlm_conn_event	
API Element Name	pool_write_time	
Element Type	counter	
Related Information	• "Resetting Monitor Data" on page 21 • "When Counters are Initialized" on page 20 • "Buffer Pool Data Writes" on page 95 • "Buffer Pool Index Writes" on page 98 • "Database Activation Timestamp" on page 41 • "Connection Request Start Timestamp" on page 62	

Description: Provides the total amount of time spent physically writing data or index pages from the buffer pool to disk.

Usage: You can use this element with Buffer Pool Data Writes and Buffer Pool Index Writes to calculate the average page-write time. This average is important since it may indicate the presence of an I/O wait, which in turn may indicate that you should be moving data to a different device.

At the database and table space levels, this element includes the value of Buffer Pool Asynchronous Write Time.

Database Files Closed

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Buffer Pool
Table Space	sqlm_tablespace	Buffer Pool
	sqlm_bp_info	Buffer Pool
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Table Space	sqlm_tablespace_event	
API Element Name	files_closed	
Element Type	counter	
Related Information	• "Resetting Monitor Data" on page 21 • "When Counters are Initialized" on page 20	

Description: The total number of database files closed.

Usage: The database manager opens files for reading and writing into and out of the buffer pool. The maximum number of database files open by an application at any time is controlled by the *maxfilop* configuration parameter. If the maximum is reached, one file will be closed before the new file is opened. Note that the actual number of files opened may not equal the number of files closed.

You can use this element to help you determine the best value for the *maxfilop* configuration parameter (see the *Administration Guide* for more information).

Buffer Pool Asynchronous Data Reads

Snapshot Information Level	**API Structure(s)**	**Monitor Switch**
Database	sqlm_dbase	Buffer Pool
Table Space	sqlm_tablespace	Buffer Pool
	sqlm_bp_info	Buffer Pool
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Table Space	sqlm_tablespace_event	
API Element Name	pool_async_data_reads	
Element Type	counter	
Related Information	• "Resetting Monitor Data" on page 21 • "When Counters are Initialized" on page 20 • "Buffer Pool Asynchronous Read Time" on page 105 • "Buffer Pool Data Physical Reads" on page 94 • "Buffer Pool Asynchronous Read Requests" on page 107 • "Direct Reads From Database" on page 115	

Description: The number of pages read asynchronously into the buffer pool.

Usage: You can use this element with Buffer Pool Data Physical Reads to calculate the number of physical reads that were performed synchronously (that is, physical data page reads that were performed by database manager agents). Use the following formula:

```
buffer pool data physical reads - buffer pool asynchronous data reads
```

By comparing the ratio of asynchronous to synchronous reads, you can gain insight into how well the prefetchers are working. This element can be helpful when you are tuning the *num_ioservers* configuration parameter (see the *Administration Guide*).

Asynchronous reads are performed by database manager prefetchers. For information about these prefetchers, see the *Administration Guide*.

Buffer Pool Asynchronous Data Writes

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Buffer Pool
Table Space	sqlm_tablespace	Buffer Pool
	sqlm_bp_info	Buffer Pool
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Table Space	sqlm_tablespace_event	
API Element Name	pool_async_data_writes	
Element Type	counter	
Related Information	• "Resetting Monitor Data" on page 21 • "When Counters are Initialized" on page 20 • "Buffer Pool Asynchronous Index Writes" on page 103 • "Buffer Pool Data Writes" on page 95 • "Buffer Pool Asynchronous Write Time" on page 106 • "Buffer Pool Log Space Cleaners Triggered" on page 107 • "Buffer Pool Victim Page Cleaners Triggered" on page 108 • "Buffer Pool Threshold Cleaners Triggered" on page 109 • "Direct Writes to Database" on page 116	

Description: The number of times a buffer pool data page was physically written to disk by either an asynchronous page cleaner, or a prefetcher. A prefetcher may have written dirty pages to disk to make space for the pages being prefetched.

Usage: You can use this element with Buffer Pool Data Writes to calculate the number of physical write requests that were performed synchronously (that is, physical data page writes that were performed by database manager agents). Use the following formula:

```
buffer pool data writes - buffer pool asynchronous data writes
```

By comparing the ratio of asynchronous to synchronous writes, you can gain insight into how well the buffer pool page cleaners are performing. This ratio can be helpful when you are tuning the *num_iocleaners* configuration parameter.

For more information about asynchronous page cleaners, see the *Administration Guide*.

Buffer Pool Asynchronous Index Writes

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Buffer Pool
Table Space	sqlm_tablespace	Buffer Pool
	sqlm_bp_info	Buffer Pool
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Table Space	sqlm_tablespace_event	
API Element Name	pool_async_index_writes	
Element Type	counter	
Related Information	• "Resetting Monitor Data" on page 21 • "When Counters are Initialized" on page 20 • "Buffer Pool Asynchronous Data Writes" on page 102 • "Buffer Pool Asynchronous Index Reads" on page 104 • "Buffer Pool Index Writes" on page 98 • "Buffer Pool Asynchronous Write Time" on page 106 • "Buffer Pool Log Space Cleaners Triggered" on page 107 • "Buffer Pool Victim Page Cleaners Triggered" on page 108 • "Buffer Pool Threshold Cleaners Triggered" on page 109 • "Direct Writes to Database" on page 116	

Description: The number of times a buffer pool index page was physically written to disk by either an asynchronous page cleaner, or a prefetcher. A prefetcher may have written dirty pages to disk to make space for the pages being prefetched.

Usage: You can use this element with Buffer Pool Index Writes to calculate the number of physical index write requests that were performed synchronously. That is, physical index page writes that were performed by database manager agents. Use the following formula:

```
buffer pool index writes - buffer pool asynchronous index writes
```

By comparing the ratio of asynchronous to synchronous writes, you can gain insight into how well the buffer pool page cleaners are performing. This ratio can be helpful when you are tuning the *num_iocleaners* configuration parameter.

For more information about asynchronous page cleaners, see the *Administration Guide*.

Buffer Pool Asynchronous Index Reads

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Buffer Pool
Table Space	sqlm_tablespace	Buffer Pool
	sqlm_bp_info	Buffer Pool
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Table Space	sqlm_tablespace_event	
API Element Name	pool_async_index_reads	
Element Type	counter	
Related Information	• "Resetting Monitor Data" on page 21 • "When Counters are Initialized" on page 20 • "Buffer Pool Asynchronous Data Writes" on page 102 • "Buffer Pool Asynchronous Index Writes" on page 103 • "Buffer Pool Index Physical Reads" on page 97 • "Buffer Pool Asynchronous Read Time" on page 105 • "Buffer Pool Log Space Cleaners Triggered" on page 107 • "Buffer Pool Victim Page Cleaners Triggered" on page 108 • "Buffer Pool Threshold Cleaners Triggered" on page 109 • "Direct Reads From Database" on page 115	

Description: The number of index pages read asynchronously into the buffer pool by a prefetcher.

Usage: You can use this element with Buffer Pool Index Physical Reads to calculate the number of physical reads that were performed synchronously (that is, physical index page reads that were performed by database manager agents). Use the following formula:

```
buffer pool index physical reads - buffer pool asynchronous index reads
```

By comparing the ratio of asynchronous to synchronous reads, you can gain insight into how well the prefetchers are working. This element can be helpful when you are tuning the *num_ioservers* configuration parameter (see the *Administration Guide*).

Asynchronous reads are performed by database manager prefetchers. For information about these prefetchers, see the *Administration Guide*.

Buffer Pool Asynchronous Read Time

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Buffer Pool
Table Space	sqlm_tablespace	Buffer Pool
	sqlm_bp_info	Buffer Pool
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Table Space	sqlm_tablespace_event	
API Element Name	pool_async_read_time	
Element Type	counter	
Related Information	• "Resetting Monitor Data" on page 21 • "When Counters are Initialized" on page 20 • "Buffer Pool Asynchronous Data Reads" on page 101 • "Total Buffer Pool Physical Read Time" on page 99 • "Buffer Pool Asynchronous Read Requests" on page 107 • "Direct Read Time" on page 118	

Description: The total elapsed time spent reading by database manager prefetchers.

Usage: You can use this element to calculate the elapsed time for synchronous reading, using the following formula:

```
total buffer pool physical read time - buffer pool asynchronous read time
```

You can also use this element to calculate the average asynchronous read time using the following formula:

```
buffer pool asynchronous read time / buffer pool asynchronous data reads
```

These calculations can be used to understand the I/O work being performed.

Buffer Pool Asynchronous Write Time

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Buffer Pool
Table Space	sqlm_tablespace	Buffer Pool
	sqlm_bp_info	Buffer Pool
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Table Space	sqlm_tablespace_event	
API Element Name	pool_async_write_time	
Element Type	counter	
Related Information	• "Resetting Monitor Data" on page 21 • "When Counters are Initialized" on page 20 • "Buffer Pool Asynchronous Data Writes" on page 102 • "Buffer Pool Asynchronous Index Writes" on page 103 • "Total Buffer Pool Physical Write Time" on page 100 • "Buffer Pool Asynchronous Read Requests" on page 107 • "Direct Write Time" on page 118	

Description: The total elapsed time spent writing data or index pages from the buffer pool to disk by database manager page cleaners.

Usage: To calculate the elapsed time spent writing pages synchronously, use the following formula:

```
total buffer pool physical write time - buffer pool asynchronous write time
```

You can also use this element to calculate the average asynchronous read time using the following formula:

```
buffer pool asynchronous write time
 / (buffer pool asynchronous data writes + buffer pool asynchronous index writes)
```

These calculations can be used to understand the I/O work being performed.

Buffer Pool Asynchronous Read Requests

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Buffer Pool
Table Space	sqlm_tablespace	Buffer Pool
	sqlm_bp_info	Buffer Pool
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Table Space	sqlm_tablespace_event	
API Element Name	pool_async_data_read_reqs	
Element Type	counter	
Related Information	• "Resetting Monitor Data" on page 21 • "When Counters are Initialized" on page 20 • "Buffer Pool Asynchronous Data Reads" on page 101	

Description: The number of asynchronous read requests.

Usage: To calculate the average number of data pages read per asynchronous request, use the following formula:

```
buffer pool asynchronous data reads / buffer pool asynchronous read requests
```

This average can help you determine the amount of asynchronous I/O done in each interaction with the prefetcher.

Buffer Pool Log Space Cleaners Triggered

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Buffer Pool
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
API Element Name	pool_lsn_gap_clns	
Element Type	counter	
Related Information	• "Resetting Monitor Data" on page 21 • "When Counters are Initialized" on page 20 • "Buffer Pool Victim Page Cleaners Triggered" on page 108 • "Buffer Pool Threshold Cleaners Triggered" on page 109	

Description: The number of times a page cleaner was invoked because the logging space used had reached a predefined criterion for the database.

Usage: This element can be used to help evaluate whether you have enough space for logging, and whether you need more log files or larger log files.

The page cleaning criterion is determined by the setting for the *softmax* configuration parameter. Page cleaners are triggered if the oldest page in the buffer pool contains an update described by a log record that is older than the current log position by the criterion value. See the *Administration Guide* for more information.

Buffer Pool Victim Page Cleaners Triggered

Snapshot Information Level	**API Structure(s)**	**Monitor Switch**
Database	sqlm_dbase	Buffer Pool
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
API Element Name	pool_drty_pg_steal_clns	
Element Type	counter	
Related Information	• "Resetting Monitor Data" on page 21 • "When Counters are Initialized" on page 20 • "Buffer Pool Log Space Cleaners Triggered" on page 107 • "Buffer Pool Threshold Cleaners Triggered" on page 109	

Description: The number of times a page cleaner was invoked because a synchronous write was needed during the victim buffer replacement for the database.

Usage: Using the following formula, you may calculate what percentage of all cleaner invocations are represented by this element:

```
  buffer pool victim page cleaners triggered
/ ( buffer pool victim page cleaners triggered
  + buffer pool threshold cleaners triggered
  + buffer pool log space cleaners triggered)
```

If this ratio is low, it may indicate that you have defined too many page cleaners. If your *chngpgs_thresh* is set too low, you may be writing out pages that you will dirty later. Aggressive cleaning defeats one purpose of the buffer pool, that is to defer writing to the last possible moment.

If this ratio is high, it may indicate that you have too few page cleaners defined. Too few page cleaners will increase recovery time after failures (see the *Administration Guide*).

Note: Although dirty pages are written out to disk, the pages are not removed from the buffer pool right away, unless the space is needed to read in new pages.

Buffer Pool Threshold Cleaners Triggered

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Buffer Pool
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
API Element Name	pool_drty_pg_thrsh_clns	
Element Type	counter	
Related Information	• "Resetting Monitor Data" on page 21 • "When Counters are Initialized" on page 20 • "Buffer Pool Log Space Cleaners Triggered" on page 107	

Description: The number of times a page cleaner was invoked because a buffer pool had reached the dirty page threshold criterion for the database.

Usage: The threshold is set by the *chngpgs_thresh* configuration parameter. It is a percentage applied to the buffer pool size. When the number of dirty pages in the pool exceeds this value, the cleaners are triggered.

If this value is set too low, pages might be written out too early, requiring them to be read back in. If set too high, then too many pages may accumulate, requiring users to write out pages synchronously. See the *Administration Guide* for more information.

Buffer Pool Information

Snapshot Information Level	API Structure(s)	Monitor Switch
Table Space	sqlm_bufferpool	Buffer Pool
Resettable	No	
Event Type	**Event Record(s)**	
Table Space	sqlm_bufferpool_event	
API Element Name	bp_info	
Element Type	information	
Related Information	• "Resetting Monitor Data" on page 21	

Description: Data management counters for a buffer pool.

Usage: Activity performed for a bufferpool.

Time Waited for Prefetch

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Buffer Pool
Application	sqlm_appl	Buffer Pool
Resettable	No	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Connection	sqlm_conn_event	
API Element Name	prefetch_wait_time	
Element Type	counter	
Related Information	• None	

Description: The time an application spent waiting for an I/O server (prefetcher) to finish loading pages into the buffer pool.

Usage: This element can be used to experiment with changing the number of I/O servers, and I/O server sizes.

Extended Storage

Extended storage provides a secondary level of storage for buffer pools. This allows a user to access memory beyond the maximum allowed for each process. Extended storage consists of segments that will be allocated in addition to the buffer pools. The extended storage will assign pages to segments that are attached or detached, as needed. The number and size of segments are configurable. Attachment is allowed to only one segment at a given time.

There is one extended storage for all buffer pools, and each buffer pool can be configured to use it or not. See the *Administration Guide* for more information.

Extended storage should only be used on systems with very large amount of real memory. These are systems that have more memory than can be attached to by a single process.

Using Extended Storage Counters: If you have extended storage set on for a buffer pool, all pages removed from the buffer pool will be written to extended storage. Each of these writes has a cost associated with it. Some of these pages may never be required or they may be forced out of extended storage before they are ever read back into the buffer pool.

You can calculate the extended storage read/write ratio as follows:

```
(data + index copied from extended storage)
/ (data + index copied to extended storage)
```

Where the numerator in this equation is pages from extended storage to buffer pool and the denominator is pages from buffer pool to extended storage.

The top portion of this equation represents a performance saving. When a page is transferred from extended storage to buffer pool, you save a system I/O call. However, you still incur the cost of attaching to the extended memory segment, copying the page, and detaching from the segment. The bottom part represents the cost of transferring a page to extended storage, that is, attaching to the segment, copying the page, and detaching.

The higher the ratio, the more likely you are to benefit from extended storage. In general, extended storage is particularly useful if I/O activity is very high on your system.

There is a crossover point where the cost of copying pages to be removed from the buffer pool to extended storage equals the savings from reading pages from extended storage, instead of having to read them from disk. This crossover point is affected by:

- cost of an I/O on your system
- cost of copying data in memory and accessing shared memory segments

It is difficult to establish an exact crossover point. To establish a baseline, you must experiment by enabling extended storage for different buffer pools, and determine whether it improves your overall database performance. This can be measured by using application benchmarks. For instance, you may want to monitor transaction rates and execution time. See the *Administration Guide* for information on benchmarking.

Once you have established that extended storage is beneficial for some buffer pools. You want to measure the read/write ratio to obtain a baseline. This ratio is most important during database creation and initial setup. After that, you want to monitor this ratio to ensure that it is not deviating from the initial baseline.

The following elements provide information about buffer pools and extended storage. For more information on how the database manager uses extended storage, see the *Administration Guide*.

- "Buffer Pool Data Pages to Extended Storage" on page 112
- "Buffer Pool Index Pages to Extended Storage" on page 112
- "Buffer Pool Data Pages from Extended Storage" on page 113
- "Buffer Pool Index Pages from Extended Storage" on page 114

Buffer Pool Data Pages to Extended Storage

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Buffer Pool
Table Space	sqlm_tablespace	Buffer Pool
	sqlm_bp_info	Buffer Pool
Application	sqlm_appl	Buffer Pool

Resettable	Yes

Event Type	Event Record(s)
Database	sqlm_db_event
Connection	sqlm_conn_event
Table Space	sqlm_tablespace_event

API Element Name	pool_data_to_estore
Element Type	counter

Related Information

- "Resetting Monitor Data" on page 21
- "Buffer Pool Index Pages to Extended Storage" on page 112
- "Buffer Pool Data Pages from Extended Storage" on page 113
- "Buffer Pool Index Pages from Extended Storage" on page 114

Description: Number of buffer pool data pages copied to extended storage.

Usage: Pages are copied from the buffer pool to extended storage, when they are selected as victim pages. This copying is required to make space for new pages in the buffer pool.

Buffer Pool Index Pages to Extended Storage

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Buffer Pool
Table Space	sqlm_tablespace	Buffer Pool
	sqlm_bp_info	Buffer Pool
Application	sqlm_appl	Buffer Pool

Resettable	Yes

Event Type	Event Record(s)
Database	sqlm_db_event
Connection	sqlm_conn_event
Table Space	sqlm_tablespace_event

API Element Name	pool_index_to_estore
Element Type	counter

Related Information

- "Resetting Monitor Data" on page 21
- "Buffer Pool Data Pages to Extended Storage" on page 112
- "Buffer Pool Data Pages from Extended Storage" on page 113
- "Buffer Pool Index Pages from Extended Storage" on page 114

Description: Number of buffer pool index pages copied to extended storage.

Usage: Pages are copied from the buffer pool to extended storage, when they are selected as victim pages. This copying is required to make space for new pages in the buffer pool.

Buffer Pool Data Pages from Extended Storage

Snapshot Information Level	**API Structure(s)**	**Monitor Switch**
Database	sqlm_dbase	Buffer Pool
Table Space	sqlm_tablespace	Buffer Pool
	sqlm_bp_info	Buffer Pool
Application	sqlm_appl	Buffer Pool

Resettable	Yes

Event Type	**Event Record(s)**
Database	sqlm_db_event
Connection	sqlm_conn_event
Table Space	sqlm_tablespace_event

API Element Name	pool_data_from_estore
Element Type	counter

Related Information

- "Resetting Monitor Data" on page 21
- "Buffer Pool Data Pages to Extended Storage" on page 112
- "Buffer Pool Index Pages to Extended Storage" on page 112
- "Buffer Pool Index Pages from Extended Storage" on page 114

Description: Number of buffer pool data pages copied from extended storage.

Usage: Required pages are copied from extended storage to the buffer pool, if they are not in the buffer pool, but are in extended storage. This copying may incur the cost of connecting to the shared memory segment, but saves the cost of a disk read.

Buffer Pool Index Pages from Extended Storage

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Buffer Pool
Table Space	sqlm_tablespace	Buffer Pool
	sqlm_bp_info	Buffer Pool
Application	sqlm_appl	Buffer Pool
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Connection	sqlm_conn_event	
Table Space	sqlm_tablespace_event	
API Element Name	pool_index_from_estore	
Element Type	counter	
Related Information	• "Resetting Monitor Data" on page 21 • "Buffer Pool Data Pages to Extended Storage" on page 112 • "Buffer Pool Index Pages to Extended Storage" on page 112 • "Buffer Pool Data Pages from Extended Storage" on page 113	

Description: Number of buffer pool index pages copied from extended storage.

Usage: Required index pages are copied from extended storage to the buffer pool, if they are not in the buffer pool, but are in extended storage. This copying may incur the cost of connecting to the shared memory segment, but saves the cost of a disk read.

Non-buffered I/O Activity

The following elements provide information about I/O activity that does not use the buffer pool:

- "Direct Reads From Database" on page 115
- "Direct Writes to Database" on page 116
- "Direct Read Requests" on page 117
- "Direct Write Requests" on page 117
- "Direct Read Time" on page 118
- "Direct Write Time" on page 118

Direct Reads From Database

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Buffer Pool
Table Space	sqlm_tablespace	Buffer Pool
	sqlm_bp_info	Buffer Pool
Application	sqlm_appl	Buffer Pool
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Connection	sqlm_conn_event	
Table Space	sqlm_tablespace_event	
API Element Name	direct_reads	
Element Type	counter	
Related Information	• "Resetting Monitor Data" on page 21 • "When Counters are Initialized" on page 20 • "Direct Read Requests" on page 117 • "Direct Read Time" on page 118 • "Direct Writes to Database" on page 116	

Description: The number of read operations that do not use the buffer pool.

Usage: Use the following formula to calculate the average number of sectors that are read by a direct read:

```
direct reads from database / direct read requests
```

When using system monitors to track I/O, this data element helps you distinguish database I/O from non-database I/O on the device.

Direct reads are performed in units, the smallest being a 512-byte sector. They are used when:

- Reading LONG VARCHAR columns
- Reading LOB (large object) columns
- Performing a backup

Direct Writes to Database

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Buffer Pool
Table Space	sqlm_tablespace	Buffer Pool
	sqlm_bp_info	Buffer Pool
Application	sqlm_appl	Buffer Pool
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Connection	sqlm_conn_event	
Table Space	sqlm_tablespace_event	
API Element Name	direct_writes	
Element Type	counter	
Related Information	• "Resetting Monitor Data" on page 21 • "When Counters are Initialized" on page 20 • "Direct Write Requests" on page 117 • "Direct Write Time" on page 118 • "Direct Reads From Database" on page 115	

Description: The number of write operations that do not use the buffer pool.

Usage: Use the following formula to calculate the average number of sectors that are written by a direct write.

```
direct writes to database / direct write requests
```

When using system monitors to track I/O, this data element helps you distinguish database I/O from non-database I/O on the device.

Direct writes are performed in units, the smallest being a 512-byte sector. They are used when:

- Writing LONG VARCHAR columns
- Writing LOB (large object) columns
- Performing a restore
- Performing a load.

Direct Read Requests

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Buffer Pool
Table Space	sqlm_tablespace	Buffer Pool
	sqlm_bp_info	Buffer Pool
Application	sqlm_appl	Buffer Pool
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Connection	sqlm_conn_event	
Table Space	sqlm_tablespace_event	
API Element Name	direct_read_reqs	
Element Type	counter	
Related Information	• "Resetting Monitor Data" on page 21 • "When Counters are Initialized" on page 20 • "Direct Reads From Database" on page 115 • "Direct Read Time" on page 118 • "Direct Write Requests" on page 117	

Description: The number of requests to perform a direct read of one or more sectors of data.

Usage: Use the following formula to calculate the average number of sectors that are read by a direct read:

```
direct reads from database / direct read requests
```

Direct Write Requests

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Buffer Pool
Table Space	sqlm_tablespace	Buffer Pool
	sqlm_bp_info	Buffer Pool
Application	sqlm_appl	Buffer Pool
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Connection	sqlm_conn_event	
Table Space	sqlm_tablespace_event	
API Element Name	direct_write_reqs	
Element Type	counter	
Related Information	• "Resetting Monitor Data" on page 21 • "When Counters are Initialized" on page 20 • "Direct Writes to Database" on page 116 • "Direct Write Time" on page 118 • "Direct Read Requests" on page 117	

Description: The number of requests to perform a direct write of one or more sectors of data.

Usage: Use the following formula to calculate the average number of sectors that are written by a direct write:

```
direct writes to database / direct write requests
```

Direct Read Time

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Buffer Pool
Table Space	sqlm_tablespace	Buffer Pool
	sqlm_bp_info	Buffer Pool
Application	sqlm_appl	Buffer Pool
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Connection	sqlm_conn_event	
Table Space	sqlm_tablespace_event	
API Element Name	direct_read_time	
Element Type	counter	
Related Information	• "Resetting Monitor Data" on page 21 • "When Counters are Initialized" on page 20 • "Direct Reads From Database" on page 115 • "Direct Read Requests" on page 117 • "Direct Write Time" on page 118	

Description: The elapsed time (in milliseconds) required to perform the direct reads.

Usage: Use the following formula to calculate the average direct read time per sector:

```
direct read time / direct reads from database
```

A high average time may indicate an I/O conflict.

Direct Write Time

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Buffer Pool
Table Space	sqlm_tablespace	Buffer Pool
	sqlm_bp_info	Buffer Pool
Application	sqlm_appl	Buffer Pool
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Connection	sqlm_conn_event	
Table Space	sqlm_tablespace_event	
API Element Name	direct_write_time	
Element Type	counter	

Related Information	• "Resetting Monitor Data" on page 21 • "When Counters are Initialized" on page 20 • "Direct Writes to Database" on page 116 • "Direct Write Requests" on page 117 • "Direct Read Time" on page 118

Description: The elapsed time (in milliseconds) required to perform the direct writes.

Usage: Use the following formula to calculate the average direct write time per sector:

```
direct write time / direct writes to database
```

A high average time may indicate an I/O conflict.

Catalog Cache

The catalog cache stores table descriptors for tables, views, and aliases. A descriptor stores information about a table, view, or alias in a condensed internal format. When a transaction references a table, it causes an insert of a table descriptor into the cache, so that subsequent transactions referencing that same table can use that descriptor and avoid reading from disk. (Transactions reference a table descriptor when compiling an SQL statement.)

The following database system monitor elements are used for catalog caches:

- "Catalog Cache Lookups"
- "Catalog Cache Inserts" on page 120
- "Catalog Cache Overflows" on page 121
- "Catalog Cache Heap Full" on page 122

Catalog Cache Lookups

Snapshot Information Level	**API Structure(s)**	**Monitor Switch**
Database	sqlm_dbase	Basic
Application	sqlm_appl	Basic
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
API Element Name	cat_cache_lookups	
Element Type	counter	
Related Information	• "Catalog Cache Inserts" on page 120 • "Catalog Cache Overflows" on page 121 • "Catalog Cache Heap Full" on page 122	

Description: The number of times that the catalog cache was referenced to obtain table descriptor information.

Usage: This element includes both successful and unsuccessful accesses to the catalog cache. The catalog cache is referenced whenever a table, view, or alias name is processed during the compilation of an SQL statement.

To calculate the catalog cache hit ratio use the following formula:

```
(1 - (cat_cache_inserts / cat_cache_lookups))
```

indicates how well the catalog cache is avoiding catalog accesses. If the ratio is high (more than 0.8), then the cache is performing well. A smaller ratio might suggest that the *catalogcache_sz* should be increased. You should expect a large ratio immediately following the first connection to the database.

The execution of Data Definition Language (DDL) SQL statements involving a table, view, or alias will evict the table descriptor information for that object from the catalog cache causing it to be re-inserted on the next reference. Therefore, the heavy use of DDLs may also increase the ratio.

See the *Administration Guide* for more information on the Catalog Cache Size configuration parameter.

Catalog Cache Inserts

Snapshot Information Level	**API Structure(s)**	**Monitor Switch**
Database	sqlm_dbase	Basic
Application	sqlm_appl	Basic
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
API Element Name	cat_cache_inserts	
Element Type	counter	
Related Information	• "Catalog Cache Lookups" on page 119 • "Catalog Cache Overflows" on page 121 • "Catalog Cache Heap Full" on page 122 • "Data Definition Language (DDL) SQL Statements" on page 169	

Description: The number of times that the system tried to insert table descriptor information into the catalog cache.

Usage: Table descriptor information is usually inserted into the cache following a failed lookup to the catalog cache while processing a table, view, or alias reference in an SQL statement. The *catalog cache inserts* value includes attempts to insert table descriptor information that fail due to catalog cache overflow and heap full conditions.

See "Catalog Cache Lookups" on page 119 for more catalog cache information.

Catalog Cache Overflows

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Basic
Application	sqlm_appl	Basic
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Connection	sqlm_conn_event	
API Element Name	cat_cache_overflows	
Element Type	counter	
Related Information	• "Catalog Cache Lookups" on page 119 • "Catalog Cache Inserts" on page 120	

Description: The number of times that an insert into the catalog cache failed due the catalog cache being full.

Usage: The catalog cache space is filled with table descriptor information.

The cache entries for transactions that compile SQL statements, either by issuing dynamic SQL statements or by binding a package, will not be eligible to be removed from the cache until that transaction has either been committed or rolled back. Catalog cache space is reclaimed by evicting table descriptor information for tables, views, or aliases that are not currently in use by any transaction. Once a transaction has experienced a catalog cache overflow, all subsequent attempts by the same transaction to insert table descriptor information into the catalog cache will also result in an an overflow.

Note: A transaction involved in an overflow will proceed, but its descriptor information is not inserted into the cache.

If *catalog cache overflows* is large, the catalog cache may be too small for the workload. Enlarging the catalog cache may improve its performance. If the workload includes transactions which compile a large number of SQL statements referencing many tables, views, and aliases in a single unit of work, then compiling fewer SQL statements in a single transaction may improve the performance of the catalog cache. Or if it includes binding of packages containing many SQL statements referencing many tables, views or aliases, you can try splitting packages so that they include fewer SQL statements to improve performance.

Catalog Cache Heap Full

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Basic
Application	sqlm_appl	Basic
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Connection	sqlm_conn_event	
API Element Name	cat_cache_heap_full	
Element Type	counter	
Related Information	• "Package Cache Inserts" on page 124 • "Data Definition Language (DDL) SQL Statements" on page 169 • "Dynamic SQL Statements Attempted" on page 165 • "Static SQL Statements Attempted" on page 164	

Description: The number of times that an insert into the catalog cache failed due to a heap-full condition in the database heap.

Usage: The catalog cache draws its storage dynamically from the database heap and even if the cache storage has not reached its limit, inserts into the catalog cache may fail due to a lack of space in the database heap.

If the catalog cache heap full count is not zero, then this insert failure condition can be corrected by increasing the database heap size or reducing the catalog cache size.

Package Cache

The package and section information required for the execution of dynamic and static SQL statements are placed in the package cache as required. This information is required whenever a dynamic or static statement is being executed. The package cache exists at a database level. This means that agents with similar environments can share the benefits of another agent's work. For static SQL statements, this can mean avoiding catalog access. For dynamic SQL statements, this can mean avoiding the cost of compilation.

The following database system monitor elements are used for package caches:

- "Package Cache Lookups" on page 123
- "Package Cache Inserts" on page 124
- "Section Lookups" on page 125
- "Section Inserts" on page 126

Package Cache Lookups

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Basic
Application	sqlm_appl	Basic
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Connection	sqlm_conn_event	
API Element Name	pkg_cache_lookups	
Element Type	counter	
Related Information	• "Package Cache Inserts" on page 124 • "Section Lookups" on page 125 • "Section Inserts" on page 126 • "Static SQL Statements Attempted" on page 164 • "Dynamic SQL Statements Attempted" on page 165 • "Data Definition Language (DDL) SQL Statements" on page 169	

Description: The number of times that an application looked for a section or package in the package cache. At a database level, it indicates the overall number of references since the database was started, or monitor data was reset.

Note: This counter includes the cases where the section is already loaded in the cache and when the section has to be loaded into the cache.

Usage: To calculate the package cache hit ratio use the following formula:

```
1 - (Package Cache Inserts / Package Cache Lookups)
```

The package cache hit ratio tells you whether or not the package cache is being used effectively. If the hit ratio is high (more than 0.8), the cache is performing well. A smaller ratio may indicate that the package cache should be increased.

You will need to experiment with the size of the package cache to find the optimal number for the *pckcachesz* configuration parameter. For example, you might be able to use a smaller package cache size if there is no increase in the *pkg_cache_inserts* data element when you decrease the size of the cache. Decreasing the package cache size frees up system resources for other work. It is also possible that you could improve overall system performance by increasing the size of the package cache if by doing so, you decrease the number of package cache inserts. This experimentation is best done under full workload conditions.

You can use this data element with *ddl_sql_stmts* to determine whether or not the execution of DDL statements is impacting the performance of the package cache. Sections for dynamic SQL statements can become invalid when DDL statements are executed. Invalid sections are implicitly prepared by the system when next used. The execution of a DDL statement could invalidate a number of sections and the resulting extra overhead incurred when preparing those sections could significantly impact performance. In this case, the package cache hit ratio reflects the implicit recompilation of invalid

sections and not the insertion of new sections into the cache, so increasing the size of the package cache will not improve overall performance. You might find it less confusing to tune the cache for an application on its own before working in the full environment.

It is necessary to determine the role that DDL statements are playing in the value of the package cache hit ratio before deciding on what action to take. If DDL statements rarely occur, then cache performance may be improved by increasing its size. If DDL statements are frequent, then improvements may require that you limit the use of DDL statements (possibly to specific time periods).

The *static_sql_stmts* and *dynamic_sql_stmts* counts can be used to help provide information on the quantity and type of sections being cached.

See the *Administration Guide* for more information on the Package Cache Size (*pckcachesz*) configuration parameter.

Note: You may want to use this information at the database level to calculate the average package cache hit ratio all each applications. You should look at this information at an application level to find out the exact package cache hit ratio for a given application. It may not be worthwhile to increase the size of the package cache in order to satisfy the cache requirements of an application that only executes infrequently.

Package Cache Inserts

Snapshot Information Level	**API Structure(s)**	**Monitor Switch**
Database	sqlm_dbase	Basic
Application	sqlm_appl	Basic
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Connection	sqlm_conn_event	
API Element Name	pkg_cache_inserts	
Element Type	counter	
Related Information	• “Package Cache Lookups” on page 123 • “Section Lookups” on page 125 • “Section Inserts” on page 126	

Description: The total number of times that a requested section was not available for use and had to be loaded into the package cache. This count includes any implicit prepares performed by the system.

Usage: In conjunction with "Package Cache Lookups", you can calculate the package cache hit ratio using the following formula:

```
1 - (Package Cache Inserts / Package Cache Lookups)
```

See “Package Cache Lookups” on page 123 for information on using this element.

Section Lookups

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Basic
Application	sqlm_appl	Basic
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Connection	sqlm_conn_event	
API Element Name	appl_section_lookups	
Element Type	counter	
Related Information	• "Package Cache Lookups" on page 123 • "Package Cache Inserts" on page 124 • "Section Inserts" on page 126	

Description: Lookups of SQL sections by an application from its SQL work area.

Usage: Each agent has access to a unique SQL work area where the working copy of any executable section is kept. In partitioned databases, this work area is shared by all non-SMP agents. In other environments and with SMP agents, each agent has its own unique SQL work area.

This counter indicates how many times the SQL work area was accessed by agents for an application. It is a cumulative total of all lookups on all SQL work heaps for agents working for this application.

You can use this element in conjunction with *"Section Inserts" on page 126* to tune the size of the heap used for the SQL work area. In partitioned databases this size is controlled by the *app_ctl_heap_sz* configuration parameter. SQL work area size in other database environments use the the *applheapsz* configuration parameter. The size of the SQL work area for SMP agents is controlled by *applheapsz* in all environments.

Section Inserts

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Basic
Application	sqlm_appl	Basic
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Connection	sqlm_conn_event	
API Element Name	appl_section_inserts	
Element Type	counter	
Related Information	• "Package Cache Lookups" on page 123 • "Package Cache Inserts" on page 124 • "Section Lookups" on page 125	

Description: Inserts of SQL sections by an application from its SQL work area.

Usage: The working copy of any executable section is stored in a unique SQL work area. This is a count of when a copy was not available and had to be inserted. See "Section Lookups" on page 125 for more information on using sections.

Database Heap

The following database system monitor elements are used for database heaps:

- "Maximum Database Heap Allocated"

Maximum Database Heap Allocated

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Basic
Resettable	No	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
API Element Name	db_heap_top	
Element Type	water mark	
Related Information	• None	

Description: The largest amount of database heap allocated and used by the database, since the first application connected to the database (in bytes).

Usage: You may use this element to evaluate the setting of the *dbheap* configuration parameter, which is described in the *Administration Guide*. The *dbheap* parameter limits the amount of storage that can be allocated for database heap.

If the value of this element is the same as the *dbheap* parameter, it is quite likely that an application has received an error indicating that there was not enough storage available.

Logging

The following database system monitor elements are used only when circular logging is being used. That is, they are not used if either the *logretain* or *userexit* configuration parameter is enabled.

- "Maximum Secondary Log Space Used"
- "Maximum Total Log Space Used" on page 128
- "Secondary Logs Allocated Currently" on page 129

The following database system monitor elements are used for all types of logging:

- "Number of Log Pages Read" on page 129
- "Number of Log Pages Written" on page 130
- "Unit of Work Log Space Used" on page 130

For more information about logging and log configuration parameters, see the *Administration Guide*.

Maximum Secondary Log Space Used

Snapshot Information Level	**API Structure(s)**	**Monitor Switch**
Database	sqlm_dbase	Basic
Resettable	No	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
API Element Name	sec_log_used_top	
Element Type	water mark	
Related Information	• "Unit of Work Log Space Used" on page 130 • "Secondary Logs Allocated Currently" on page 129 • "Maximum Total Log Space Used" on page 128	

Description: The maximum amount of secondary log space used (in bytes).

Usage: You may use this element in conjunction with *Secondary Logs Allocated Currently* and *Maximum Total Log Space Used* to show your current dependency on secondary logs. If this value is high, you may need larger log files, or more primary log files, or more frequent COMMIT statements within your application.

As a result, you may need to adjust the following configuration parameters:

- logfilsz
- logprimary
- logsecond
- logretain

The value will be zero if the database does not have any secondary log files. This would be the case if there were none defined.

For more information, see the *Administration Guide*.

Note: While the database system monitor information is given in bytes, the configuration parameters are set in pages, which are each 4K bytes.

Maximum Total Log Space Used

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Basic
Resettable	No	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
API Element Name	tot_log_used_top	
Element Type	water mark	
Related Information	• "Unit of Work Log Space Used" on page 130 • "Secondary Logs Allocated Currently" on page 129 • "Maximum Secondary Log Space Used" on page 127	

Description: The maximum amount of total log space used (in bytes).

Usage: You can use this element to help you evaluate the amount of primary log space that you have allocated. Comparing the value of this element with the amount of primary log space you have allocated can help you to evaluate your configuration parameter settings. Your primary log space allocation can be calculated using the following formula:

```
logprimary x logfilsiz x 4096 (see note below)
```

You can use this element in conjunction with *Maximum Secondary Log Space Used* and *Secondary Logs Allocated Currently* to show your current dependency on secondary logs.

This value includes space used in both primary and secondary log files, and is only returned if circular logging is used. (That is, it is not returned if either the *logretain* or *userexit* configuration parameter is enabled.)

As a result, you may need to adjust the following configuration parameters:

- logfilsz
- logprimary
- logsecond
- logretain

For more information, see the *Administration Guide*.

Note: While the database system monitor information is given in bytes, the configuration parameters are set in pages, which are each 4K bytes.

Secondary Logs Allocated Currently

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Basic
Resettable	No	
API Element Name	sec_logs_allocated	
Element Type	gauge	
Related Information	• "Unit of Work Log Space Used" on page 130 • "Maximum Secondary Log Space Used" on page 127 • "Maximum Total Log Space Used" on page 128	

Description: The total number of secondary log files that are currently being used for the database.

Usage: You may use this element in conjunction with *Maximum Secondary Log Space Used* and *Maximum Total Log Space Used* to show your current dependency on secondary logs. If this value is consistently high, you may need larger log files, or more primary log files, or more frequent COMMIT statements within your application.

As a result, you may need to adjust the following configuration parameters:

- logfilsz
- logprimary
- logsecond
- logretain

For more information, see the *Administration Guide*.

Number of Log Pages Read

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Basic
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
API Element Name	log_reads	
Element Type	counter	
Related Information	• "When Counters are Initialized" on page 20 • "Number of Log Pages Written" on page 130	

Description: The number of log pages read from disk by the logger.

Usage: You can use this element with an operating system monitor to quantify the amount of I/O on a device that is attributable to database activity.

Number of Log Pages Written

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Basic
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
API Element Name	log_writes	
Element Type	counter	
Related Information	• "When Counters are Initialized" on page 20 • "Number of Log Pages Read" on page 129	

Description: The number of log pages written to disk by the logger.

Usage: You may use this element with an operating system monitor to quantify the amount of I/O on a device that is attributable to database activity.

Unit of Work Log Space Used

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_appl	Unit of Work
Resettable	No	
Event Type	**Event Record(s)**	
Transaction	sqlm_xaction_event	
API Element Name	uow_log_space_used (Snapshot) log_space_used (Event)	
Element Type	gauge	
Related Information	• "Resetting Monitor Data" on page 21 • "Secondary Logs Allocated Currently" on page 129 • "Maximum Secondary Log Space Used" on page 127 • "Maximum Total Log Space Used" on page 128	

Description: The amount of log space (in bytes) used in the current unit of work of the monitored application.

Usage: You may use this element to understand the logging requirements at the unit of work level.

Database and Application Activity

The following sections provide information on database and application activity.

Locks and Deadlocks

The following elements provide information about locks and deadlocks:

- "Locks Held" on page 131
- "Total Lock List Memory In Use" on page 132

- "Deadlocks Detected" on page 133
- "Lock Escalations" on page 134
- "Exclusive Lock Escalations" on page 135
- "Lock Mode" on page 136
- "Lock Status" on page 137
- "Lock Object Type Waited On" on page 137
- "Lock Object Name" on page 138
- "Number of Lock Timeouts" on page 139
- "Maximum Number of Locks Held" on page 139
- "Connections Involved in Deadlock" on page 140

Locks Held

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Basic
Application	sqlm_appl	Basic
Lock	sqlm_dbase_lock	Basic
	sqlm_appl_lock	Basic
Resettable	No	
API Element Name	locks_held	
Element Type	gauge	
Related Information	• "Lock Escalations" on page 134 • "Exclusive Lock Escalations" on page 135 • "Maximum Number of Locks Held" on page 139	

Description: The number of locks currently held.

Usage: If the monitor information is at the database level, this is the total number of locks currently held by all applications in the database.

If it is at the application level, this is the total number of locks currently held by all agents for the application. How you use this element depends on the level of information being returned from the database system monitor.

- At the database level, you can use it in one of two ways:
  - This element can provide summary information about locking. For example, you can calculate the average number of locks per application by dividing the value of this element by *Applications Connected Currently.* If the resulting number is high, it may indicate that you can tune one of your applications to improve performance.
  - You can also compare the value of this element against the results of the following formula to determine the number of additional locks that may be requested. This comparison can help you determine if the configuration parameters need adjusting or your applications need tuning.

    ```
    (locklist * 4096 / 36 ) - locks held = # remaining
    ```

    where:

- *locklist* is the configuration parameter as described in the *Administration Guide*
- 4096 is the number of bytes in one 4K page
- 36 is the number of bytes required for each lock.

Note: You may also use "Total Lock List Memory In Use" in a similar fashion.

- At the application level, you can use this counter to determine if the application is approaching the maximum number of locks available to it, as defined by the *maxlocks* configuration parameter. This parameter indicates the percentage of the lock list that each application can use before lock escalations occur. Lock escalations can result in a decrease in concurrency between applications connected to a database. (See the *Administration Guide* for more information about this parameter.)

  Since the *maxlocks* parameter is specified as a percentage and this element is a counter, you can compare the count provided by this element against the total number of locks that can be held by an application, as calculated using the following formula:

  ```
  (locklist * 4096 / 36 ) * (maxlocks / 100)
  ```

  If you have a large number of locks, you may need to perform more commits within your application so that some of the locks can be released.

Total Lock List Memory In Use

Snapshot Information Level	**API Structure(s)**	**Monitor Switch**
Database	sqlm_dbase	Basic
Resettable	No	
API Element Name	lock_list_in_use	
Element Type	gauge	
Related Information	• None	

Description: The total amount of lock list memory (in bytes) that is in use.

Usage: This element may be used in conjunction with the *locklist* configuration parameter to calculate the lock list utilization. If the lock list utilization is high, you may want to consider increasing the size of that parameter. See the *Administration Guide* for more information.

Note: When calculating utilization, it is important to note that the *locklist* configuration parameter is allocated in pages of 4K bytes each, while this monitor element provides results in bytes.

Deadlocks Detected

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Basic
Application	sqlm_appl	Lock
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Connection	sqlm_conn_event	
API Element Name	deadlocks	
Element Type	counter	
Related Information	• "Resetting Monitor Data" on page 21 • "When Counters are Initialized" on page 20 • "Database Activation Timestamp" on page 41 • "Connection Request Start Timestamp" on page 62 • "Lock Escalations" on page 134 • "Exclusive Lock Escalations" on page 135 • "Application ID Holding Lock" on page 146	

Description: The total number of deadlocks that have occurred.

Usage: This element can indicate that applications are experiencing contention problems. These problems could be caused by the following situations:

- Lock escalations are occurring for the database
- An application may be locking tables explicitly when system-generated row locks may be sufficient
- An application may be using an inappropriate isolation level when binding
- Catalog tables are locked for repeatable read
- Applications are getting the same locks in different orders, resulting in deadlock.

You may be able to resolve the problem by determining in which applications (or application processes) the deadlocks are occurring. You may then be able to modify the application to better enable it to execute concurrently. Some applications, however, may not be capable of running concurrently.

You can use the connection timestamp monitor elements (*"Last Reset Timestamp" on page 192, Database Activation Timestamp* , and *Connection Request Start Timestamp)* to determine the severity of the deadlocks. For example, 10 deadlocks in 5 minutes is much more severe than 10 deadlocks in 5 hours.

The descriptions for the related elements listed above may also provide additional tuning suggestions.

Lock Escalations

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Basic
Application	sqlm_appl	Basic
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Connection	sqlm_conn_event	
Transaction	sqlm_xaction_event	
API Element Name	lock_escals	
Element Type	counter	
Related Information	• "When Counters are Initialized" on page 20 • "Database Activation Timestamp" on page 41 • "Exclusive Lock Escalations" on page 135 • "Maximum Number of Locks Held" on page 139	

Description: The number of times that locks have been escalated from several row locks to a table lock.

Usage: A lock is escalated when the total number of locks held by an application reaches the maximum amount of lock list space available to the application, or the lock list space consumed by all applications is approaching the total lock list space. The amount of lock list space available is determined by the *maxlocks* and *locklist* configuration parameters.

When an application reaches the maximum number of locks allowed and there are no more locks to escalate, it will then use space in the lock list allocated for other applications. When the entire lock list is full, an error occurs.

This data item includes a count of all lock escalations, including exclusive lock escalations.

There are several possible causes for excessive lock escalations:

- The lock list size (*locklist*) may be too small for the number of concurrent applications
- The percent of the lock list usable by each application (*maxlocks*) may be too small
- One or more applications may be using an excessive number of locks.

To resolve these problems, you may be able to:

- Increase the *locklist* configuration parameter value. See the *Administration Guide* for a description of this configuration parameter.
- Increase the *maxlocks* configuration parameter value. See the *Administration Guide* for a description of this configuration parameter.
- Identify the applications with large numbers of locks (see *Maximum Number of Locks Held)*, or those that are holding too much of the lock list, using the following formula:

```
(((locks held * 36) / (locklist * 4096)) * 100)
```

and comparing the value to maxlocks. These applications can also cause lock escalations in other applications by using too large a portion of the lock list. These applications may need to resort to using table locks instead of row locks, although table locks may cause an increase in “Lock Waits” on page 141 and “Time Waited On Locks” on page 142.

- Identify applications holding too much of the lock list, using the following formula:

Exclusive Lock Escalations

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Basic
Application	sqlm_appl	Basic
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Connection	sqlm_conn_event	
Transaction	sqlm_xaction_event	
API Element Name	x_lock_escals	
Element Type	counter	
Related Information	• “When Counters are Initialized” on page 20 • “Database Activation Timestamp” on page 41 • “Lock Escalations” on page 134 • “Connection Request Start Timestamp” on page 62 • “Maximum Number of Locks Held” on page 139	

Description: The number of times that locks have been escalated from several row locks to one exclusive table lock, or the number of times an exclusive lock on a row caused the table lock to become an exclusive lock.

Usage: Other applications cannot access data held by an exclusive lock; therefore it is important to track exclusive locks since they can impact the concurrency of your data.

A lock is escalated when the total number of locks held by an application reaches the maximum amount of lock list space available to the application. The amount of lock list space available is determined by the *locklist* and *maxlocks* configuration parameters.

When an application reaches the maximum number of locks allowed and there are no more locks to escalate, it will then use space in the lock list allocated for other applications. When the entire lock list is full, an error occurs.

See *“Lock Escalations” on page 134* for possible causes and resolutions to excessive exclusive lock escalations.

An application may be using exclusive locks when share locks are sufficient. Although share locks may not reduce the total number of lock escalations share lock escalations may be preferable to exclusive lock escalations.

Lock Mode

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_appl	Lock
Lock	sqlm_lock	Lock
	sqlm_lock_wait	Lock
Resettable	No	
Event Type	**Event Record(s)**	
Deadlock	sqlm_dlconn_event	
API Element Name	lock_mode	
Element Type	information	
Related Information	• "Resetting Monitor Data" on page 21 • Other lock information	

Description: The type of lock being held.

Usage: This mode can help you determine the source of contention for resources.

This element indicates one of the following, depending on the type of monitor information being examined:

- The type of lock another application holds on the object that this application is waiting to lock (for application-monitoring and deadlock-monitoring levels)
- The type of lock held on the object by this application (for object-lock levels).

The values for this field are:

Mode	Type of Lock	API Constant
	No Lock	SQLM_LNON
IS	Intention Share Lock	SQLM_LOIS
IX	Intention Exclusive Lock	SQLM_LOIX
S	Share Lock	SQLM_LOOS
SIX	Share with Intention Exclusive Lock	SQLM_LSIX
X	Exclusive Lock	SQLM_LOOX
IN	Intent None	SQLM_LOIN
Z	Super Exclusive Lock	SQLM_LOOZ
U	Update Lock	SQLM_LOOU
NS	Next Key Share Lock	SQLM_LONS
NX	Next Key Exclusive Lock	SQLM_LONX
W	Weak Exclusive Lock	SQLM_LOOW
NW	Next Key Weak Exclusive Lock	SQLM_LONW

Lock Status

Snapshot Information Level	API Structure(s)	Monitor Switch
Lock	sqlm_lock	Basic
Resettable	No	
API Element Name	lock_status	
Element Type	information	
Related Information	• "Lock Mode" on page 136 • "Lock Object Type Waited On" on page 137 • "Table File ID" on page 159	

Description: Indicates the internal status of the lock.

Usage: This element can help explain what is happening when an application is waiting to obtain a lock on an object. While it may appear that the application already has a lock on the object it needs, it may have to wait to obtain a different type of lock on the same object.

The lock can be in one of the following statuses:

Granted state indicates that the application has the lock in the state specified by "Lock Mode" on page 136.

Converting state indicates that the application is trying to change the lock held to a different type; for example, changing from a share lock to an exclusive lock.

Note: API users should refer to the *sqlmon.h* header file containing definitions of database system monitor constants.

Lock Object Type Waited On

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_appl	Lock
	sqlm_appl_lock	Lock
Lock	sqlm_lock	Basic
	sqlm_lock_wait	Lock
Resettable	No	
Event Type	**Event Record(s)**	
Deadlock	sqlm_dlconn_event	
API Element Name	lock_object_type	
Element Type	information	
Related Information	• "Resetting Monitor Data" on page 21	

Description: The type of object against which the application holds a lock (for object-lock-level information), or the type of object for which the application is waiting to obtain a lock (for application-level and deadlock-level information).

Usage: This element can help you determine the source of contention for resources.

The objects may be one of the following types:

- Table space
- Table
- Record (or row)
- Internal (another type of lock held internally by the database manager).

Lock Object Name

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_appl	Lock
Lock	sqlm_appl_lock	Lock
	sqlm_lock	Basic
Resettable	No	
Event Type	**Event Record(s)**	
Deadlock	sqlm_dlconn_event	
API Element Name	lock_object_name	
Element Type	information	
Related Information	• "Resetting Monitor Data" on page 21 • "Lock Object Type Waited On" on page 137 • "Table Space Name" on page 143 • "Table Name" on page 151 • "Table Schema Name" on page 152	

Description: This element is provided for informational purposes only. It is the name of the object for which the application holds a lock (for object-lock-level information), or the name of the object for which the application is waiting to obtain a lock (for application-level and deadlock-level information).

Usage: It is the name of the object for table-level locks is the file ID (FID) for SMS and DMS table spaces. For row-level locks, the object name is the row ID (RID). For table space locks, the object name is blank.

To determine the table holding the lock, use *Table Name* and *Table Schema Name* instead of the file ID, since the file ID may not be unique.

To determine the table space holding the lock, use *Table Space Name* .

Number of Lock Timeouts

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Basic
Application	sqlm_appl	Basic

Resettable	Yes

Event Type	Event Record(s)
Database	sqlm_db_event
Connection	sqlm_conn_event

API Element Name	lock_timeouts
Element Type	counter

Related Information

- "When Counters are Initialized" on page 20
- Other elements in "Locks and Deadlocks" on page 130

Description: The number of times that a request to lock an object timed-out instead of being granted.

Usage: This element can help you adjust the setting for the *locktimeout* database configuration parameter. If the number of lock time-outs becomes excessive when compared to normal operating levels, you may have an application that is holding locks for long durations. In this case, this element may indicate that you should analyze some of the other elements related to "Locks and Deadlocks" on page 130 to determine if you have an application problem.

You could also have too few lock time-outs if your *locktimeout* database configuration parameter is set too high. In this case, your applications may wait excessively to obtain a lock. See the *Administration Guide* for more information.

Maximum Number of Locks Held

Event Type	Event Record(s)
Transaction	sqlm_xaction_event

API Element Name	locks_held_top
Element Type	counter

Related Information

- "Locks Held" on page 131
- "Lock Escalations" on page 134
- "Exclusive Lock Escalations" on page 135

Description: The maximum number of locks held during this transaction.

Usage: You can use this element to determine if your application is approaching the maximum number of locks available to it, as defined by the *maxlocks* configuration parameter. This parameter indicates the percentage of the lock list that each application can use before lock escalations occur. Lock escalations can result in a decrease in concurrency between applications connected to a database. (See the *Administration Guide* for more information about this parameter.)

Since the *maxlocks* parameter is specified as a percentage and this element is a counter, you can compare the count provided by this element against the total number of locks that can be held by an application, as calculated using the following formula:

```
(locklist * 4096 / 36 ) * (maxlocks / 100)
```

If you have a large number of locks, you may need to perform more commits within your application so that some of the locks can be released.

Connections Involved in Deadlock

Event Type	**Event Record(s)**
Deadlock	sqlm_deadlock_event
API Element Name	dl_conns
Element Type	gauge
Related Information	• None

Description: The number of connections that are involved in the deadlock.

Usage: Use this element in your monitoring application to identify how many deadlock connection event records will follow in the event monitor data stream.

Lock Wait Information

The following elements provide information is returned when a DB2 agent working on behalf of an application is waiting to obtain a lock:

Lock Waits

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Lock
Application	sqlm_appl	Lock

Resettable	Yes

Event Type	Event Record(s)
Database	sqlm_db_event
Connection	sqlm_conn_event

API Element Name	lock_waits
Element Type	counter

Related Information

- "Resetting Monitor Data" on page 21
- "When Counters are Initialized" on page 20
- "Connection Request Start Timestamp" on page 62
- "Time Waited On Locks" on page 142

Description: The total number of times that applications or connections waited for locks.

Usage: At the database level, this is the total number of times that applications have had to wait for locks within this database.

At the application-connection level, this is the total number of times that this connection requested a lock but had to wait because another connection was already holding a lock on the data.

This element may be used with *Time Waited On Locks* to calculate, at the database level, the average wait time for a lock. This calculation can be done at either the database or the application-connection level.

If the average lock wait time is high, you should look for applications that hold many locks, or have lock escalations, with a focus on tuning your applications to improve concurrency, if appropriate. If escalations are the reason for a high average lock wait time, then the values of one or both of the *locklist* and *maxlocks* configuration parameters may be too low. See the *Administration Guide* for more information.

Time Waited On Locks

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Lock
Application	sqlm_appl sqlm_appl_lock	Lock
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Connection	sqlm_conn_event	
Transaction	sqlm_xaction_event	
API Element Name **Element Type**	lock_wait_time counter	
Related Information	• "Resetting Monitor Data" on page 21 • "When Counters are Initialized" on page 20 • "Current Agents Waiting On Locks" on page 144 • "Lock Waits" on page 141	

Description: The total elapsed time waited for a lock.

Usage: At the database level, this is the total amount of elapsed time that all applications were waiting for a lock within this database.

At the application-connection and transaction levels, this is the total amount of elapsed time that this connection or transaction has waited for a lock to be granted to it.

This element may be used in conjunction with the Lock Waits monitor element to calculate the average wait time for a lock. This calculation can be performed at either the database or the application-connection level.

When using data elements providing elapsed times, you should consider:

- Elapsed times are affected by system load, so the more processes you have running, the higher this elapsed time value.
- To calculate this data element at the database level, the database system monitor sums the application-level times. This can result in double counting elapsed times at a database level, since more than one application process can be running at the same time.

  To provide meaningful data, you can calculate the average wait time for a lock, as described above.

Table Space Name

Snapshot Information Level	API Structure(s)	Monitor Switch
Table Space	sqlm_tablespace	Buffer Pool
Application	sqlm_appl_lock	Basic
Lock	sqlm_lock	Lock
	sqlm_lock_wait	Lock
Resettable	No	
Event Type	**Event Record(s)**	
Deadlock	sqlm_dlconn_event	
Table Space	sqlm_tablespace_header	
API Element Name	tablespace_name	
Element Type	information	
Related Information	• "Resetting Monitor Data" on page 21 • "Lock Object Type Waited On" on page 137	

Description: The name of a table space.

Usage: This element can help you determine the source of contention for resources.

It is equivalent to the TBSPACE column in the database catalog table SYSCAT.TABLESPACE. At the application level, application-lock level, and deadlock monitoring level, this is the name of the table space that the application is waiting to lock. Another application currently holds a lock on this table space.

At the lock level, this is the name of the table space against which the application currently holds a lock.

At the table space level (when the buffer pool monitor group is ON), this is the name of the table space for which information is returned.

If you are using the database system monitor APIs, note that the API constant SQLM_IDENT_SZ is used to define the length of this element. Only the first 18 characters are currently used.

Current Agents Waiting On Locks

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Basic
Lock	sqlm_dbase_lock	Basic
Resettable	No	
API Element Name	locks_waiting	
Element Type	gauge	
Related Information	• "Applications Connected Currently" on page 72	

Description: Indicates the number of agents waiting on a lock.

Usage: When used in conjunction with *Applications Connected Currently,* this element indicates the percentage of applications waiting on locks. If this number is high, the applications may have concurrency problems, and you should identify applications that are holding locks or exclusive locks for long periods of time.

Total Time Unit of Work Waited on Locks

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_appl	Unit of Work
Resettable	No	
API Element Name	uow_lock_wait_time	
Element Type	counter	
Related Information	• "Resetting Monitor Data" on page 21 • Application-level information on locks	

Description: The total amount of elapsed time this unit of work has spent waiting for locks.

Usage: This element can help you determine the severity of the resource contention problem.

Lock Wait Start Timestamp

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_appl	Lock
Lock	sqlm_lock_wait	Lock
Resettable	No	
Event Type	**Event Record(s)**	
Deadlock	sqlm_dlconn_event	
API Element Name	lock_wait_start_time	
Element Type	timestamp	
Related Information	• "Resetting Monitor Data" on page 21 • "Agent ID Holding Lock" on page 145	

Description: The date and time that this application started waiting to obtain a lock on the object that is currently locked by another application.

Usage: This element can help you determine the severity of resource contention.

Agent ID Holding Lock

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_appl	Lock
	sqlm_appl_lock	Lock
Lock	sqlm_lock_wait	Lock
Resettable	No	
API Element Name	agent_id_holding_lock	
Element Type	information	
Related Information	• "Resetting Monitor Data" on page 21 • "Lock Wait Start Timestamp" on page 145 • "Application ID Holding Lock" on page 146	

Description: The application handle of the agent holding a lock for which this application is waiting. The lock monitor group must be turned on to obtain this information.

Usage: This element can help you determine which applications are in contention for resources.

If this element is 0 (zero) and the application is waiting for a lock, this indicates that the lock is held by an indoubt transaction. You can use either "Application ID Holding Lock" on page 146 or the command line processor LIST INDOUBT TRANSACTIONS command (which displays the application ID of the CICS agent that was processing the transaction when it became indoubt) to determine the indoubt transaction, and then either commit it or roll it back.

Note that more than one application can hold a shared lock on an object for which this application is waiting. See "Lock Mode" on page 136 for information about the type of lock that the application holds. If you are taking an application snapshot, only one of the

agent IDs holding a lock on the object will be returned. If you are taking a lock snapshot, all of the agent IDs holding a lock on the object will be identified.

Application ID Holding Lock

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_appl	Lock
	sqlm_appl_lock	Lock
Lock	sqlm_lock_wait	Lock
Resettable	No	
Event Type	**Event Record(s)**	
Deadlock	sqlm_dlconn_event	
API Element Name	appl_id_holding_lk	
Element Type	information	
Related Information	• "Resetting Monitor Data" on page 21 • "Agent ID Holding Lock" on page 145 • "Deadlocks Detected" on page 133	

Description: The application ID of the application that is holding a lock on the object that this application is waiting to obtain.

Usage: This element can help you determine which applications are in contention for resources. Specifically, it can help you identify the application handle (agent ID) and table ID that are holding the lock. Note that you may use the LIST APPLICATIONS command to obtain information to relate the application ID with an agent ID. However, it is a good idea to collect this type of information when you take the snapshot, as it could be unavailable if the application ends before you run the LIST APPLICATIONS command.

If you are using the database system monitor APIs, note that the API constant SQLM_APPLID_SZ is used to define the length of this element. Only the first 30 characters are currently used.

Note that more than one application can hold a shared lock on an object for which this application is waiting to obtain a lock. See "Lock Mode" on page 136 for information about the type of lock that the application holds. If you are taking an application snapshot, only one of the application IDs holding a lock on the object will be returned. If you are taking a lock snapshot, all of the application IDs holding a lock on the object will be returned.

Sequence Number Holding Lock

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_appl sqlm_appl_lock	Basic Basic
Resettable	No	
Event Type	**Event Record(s)**	
Deadlock	sqlm_dlconn_event	
API Element Name	sequence_no_holding_lk	
Element Type	information	
Related Information	• None	

Description: This element is reserved for future use. In this release, its value will always be "0001." In future releases of the product, it may contain different values.

Rolled Back Application

Event Type	Event Record(s)
Deadlock	sqlm_deadlock_event
API Element Name	rolled_back_appl_id
Element Type	information
Related Information	• "Service Level" on page 37 • "Maximum Number of Coordinating Agents" on page 76

Description: Application id that was rolled back when a deadlock occurred.

Usage: A system administrator can use this information to determine which application did not complete its updates, and determine which applications should be restarted

Rollforward Monitoring

Recovering database changes can be a time consuming process. You can use the database system monitor to monitor the progression of a recovery. The following elements provide information about rollforward status:

- "Rollforward Timestamp" on page 148
- "Tablespace Being Rolled Forward" on page 148
- "Rollforward Type" on page 148
- "Log Being Rolled Forward" on page 149
- "Log Phase" on page 149
- "Number of Rollforward Table Spaces" on page 149

Rollforward Timestamp

Snapshot Information Level	API Structure(s)	Monitor Switch
Table Space	sqlm_rollfwd_info	Basic
Resettable	No	
API Element Name	rf_timestamp	
Element Type	timestamp	
Related Information	• "Tablespace Being Rolled Forward" on page 148	

Description: The timestamp of the log being processed.

Usage: If a rollforward is in progress, this is the timestamp of the log record being processed. This is an indicator of the data changes that will be recovered.

Tablespace Being Rolled Forward

Snapshot Information Level	API Structure(s)	Monitor Switch
Table Space	sqlm_rollfwd_ts_info	Basic
Resettable	No	
API Element Name	ts_name	
Element Type	information	
Related Information	• "Rollforward Timestamp" on page 148	

Description: The name of the table space currently rolled forward.

Usage: If a rollforward is in progress, this element identifies the table spaces involved.

Rollforward Type

Snapshot Information Level	API Structure(s)	Monitor Switch
Table Space	sqlm_rollfwd_info	Basic
Resettable	No	
API Element Name	rf_type	
Element Type	information	
Related Information	• None	

Description: The type of rollforward in progess.

Usage: An indicator of whether recovery is happening at a database or table space level. For more information on rollforward recovery at the database or table space level see the *Administration Guide*.

Log Being Rolled Forward

Snapshot Information Level	API Structure(s)	Monitor Switch
Table Space	sqlm_rollfwd_info	Basic
Resettable	No	
API Element Name **Element Type**	rf_log_num information	
Related Information	• None	

Description: The log being processed.

Usage: If a rollforward is in progress, this element identifies the log involved.

Log Phase

Snapshot Information Level	API Structure(s)	Monitor Switch
Table Space	sqlm_rollfwd_info	Basic
Resettable	No	
API Element Name **Element Type**	rf_status information	
Related Information	• None	

Description: The status of the recovery.

Usage: This element indicates the progression of a recovery. It indicates if the recovery is in an undo (rollback) or redo (rollforward) phase.

Number of Rollforward Table Spaces

Snapshot Information Level	API Structure(s)	Monitor Switch
Table Space	sqlm_rollfwd_info	Basic
Resettable	No	
API Element Name **Element Type**	rf_num_tspaces counter	
Related Information	• None	

Description: The number of table spaces involved in a rollforward.

Usage: This is a counter of the table spaces involved in recovery.

Table Activity

The following elements provide information about the tables:

- "Table Type" on page 150
- "Table Name" on page 151
- "Table Schema Name" on page 152
- "Rows Deleted" on page 153

- “Rows Inserted” on page 153
- “Rows Updated” on page 154
- “Rows Selected” on page 154
- “Rows Written” on page 155
- “Rows Read” on page 156
- “Accesses to Overflowed Records” on page 157
- “Internal Rows Deleted” on page 157
- “Internal Rows Updated” on page 158
- “Internal Rows Inserted” on page 159
- “Table File ID” on page 159

Table Type

Snapshot Information Level	**API Structure(s)**	**Monitor Switch**
Table	sqlm_table	Table
Resettable	No	
Event Type	**Event Record(s)**	
Table	sqlm_table_event	
API Element Name	table_type	
Element Type	information	
Related Information	• “Resetting Monitor Data” on page 21 • “Table File ID” on page 159	

Description: The type of table for which information is returned.

Usage: You can use this element to help identify the table for which information is returned. If the table is a user table or a system catalog table, you can use *Table Name* and *Table Schema Name* to identify the table.

The type of table may be one of the following:

- User table.
- User table that has been dropped. The table type will only be updated after the changes are committed (either explicitly or implicitly).
- Temporary table. Information regarding temporary tables is returned, even though the tables are not kept in the database after being used. You may still find information about this type of table useful.
- System catalog table.
- Reorganization table. A table created and used by the database manager while performing a reorganization of another table.

Table Name

Snapshot Information Level	API Structure(s)	Monitor Switch
Table	sqlm_table	Table
Application	sqlm_appl	Lock
	sqml_appl_lock	Lock
Lock	sqlm_lock	Lock
	sqlm_lock_wait	Lock
Resettable	No	
Event Type	**Event Record(s)**	
Table	sqlm_table_event	
Deadlock	sqlm_dlconn_event	
API Element Name	table_name	
Element Type	information	
Related Information	• "Resetting Monitor Data" on page 21 • "Table Schema Name" on page 152 • "Lock Object Type Waited On" on page 137	

Description: The name of the table.

Usage: Along with *Table Schema Name,* this element can help you determine the source of contention for resources.

At the application-level, application-lock level, and deadlock-monitoring-level, this is the table that the application is waiting to lock, because it is currently locked by another application. For snapshot monitoring, this item is only valid when the "lock" monitor group information is turned on, and when *Lock Object Type Waited On* indicates that the application is waiting to obtain a table lock.

For snapshot monitoring at the object-lock level, this item is returned for table-level and row-level locks. The table reported at this level is the table against which this application holds these locks.

For snapshot and event monitoring at the table level, this is the table for which information has been collected. This element is blank for temporary tables, reorganization tables, and tables that were dropped. Table names are only provided for catalog and user tables. For snapshot monitoring, this element is only valid when the "table" monitor group information is turned on.

If you are using the database system monitor APIs, note that the API constant SQLM_IDENT_SZ is used to define the length of this element. Only the first 18 characters are currently used.

Table Schema Name

Snapshot Information Level	API Structure(s)	Monitor Switch
Table	sqlm_table	Table
Application	sqlm_appl	Lock
	sqlm_appl_lock	Lock
Lock	sqlm_lock	Lock
	sqlm_lock_wait	Lock
Resettable	No	
Event Type	**Event Record(s)**	
Table	sqlm_table_event	
Deadlock	sqlm_dlconn_event	
API Element Name	table_schema	
Element Type	information	
Related Information	• "Resetting Monitor Data" on page 21 • "Table Name" on page 151 • "Lock Object Type Waited On" on page 137	

Description: The schema of the table.

Usage: Along with *Table Name,* this element can help you determine the source of contention for resources.

For application-level, application-lock-level, deadlock-monitoring-level, this is the schema of the table that the application is waiting to lock, because it is currently locked by another application. This element is only set if *Lock Object Type Waited On* indicates that the application is waiting to obtain a table lock. For snapshot monitoring at the application-level and application-lock levels, this item is only valid when the "lock" monitor group information is turned on.

For snapshot monitoring at the object-lock level, this item is returned for table and row level locks. The table reported at this level is the table against which this application holds these locks.

For snapshot and event monitoring at the table level, this element identifies the schema of the table for which information has been collected. This element is blank for temporary tables, reorganization tables, and tables that were dropped. Schema names are provided only for catalog and user tables. For snapshot monitoring, this element is valid only when the "table" monitor group information is turned on.

If you are using the database system monitor APIs, note that the API constant SQLM_IDENT_SZ is used to define the length of this element. Only the first 8 characters are currently used.

Rows Deleted

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Basic
Application	sqlm_appl	Basic
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Connection	sqlm_conn_event	
API Element Name	rows_deleted	
Element Type	counter	
Related Information	• "When Counters are Initialized" on page 20 • "Internal Rows Deleted" on page 157	

Description: This is the number of row deletions attempted.

Usage: You can use this element to gain insight into the current level of activity within the database manager.

This count does not include the attempts counted in *Internal Rows Deleted*.

Rows Inserted

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Basic
Application	sqlm_appl	Basic
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Connection	sqlm_conn_event	
API Element Name	rows_inserted	
Element Type	counter	
Related Information	• "When Counters are Initialized" on page 20	

Description: This is the number of row insertions attempted.

Usage: You can use this element to gain insight into the current level of activity within the database manager.

Rows Updated

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Basic
Application	sqlm_appl	Basic
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Connection	sqlm_conn_event	
API Element Name	rows_updated	
Element Type	counter	
Related Information	• "When Counters are Initialized" on page 20 • "Internal Rows Updated" on page 158	

Description: This is the number of row updates attempted.

Usage: You can use this element to gain insight into the current level of activity within the database manager.

This value does not include updates counted in *Internal Rows Updated.*. However, rows that are updated by more than one update statement are counted for each update.

Rows Selected

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Basic
Application	sqlm_appl	Basic
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Connection	sqlm_conn_event	
API Element Name	rows_selected	
Element Type	counter	
Related Information	• "When Counters are Initialized" on page 20 • "Select SQL Statements Executed" on page 168	

Description: This is the number of rows that have been selected and returned to the application.

Usage: You can use this element to gain insight into the current level of activity within the database manager.

This element does not include a count of rows read for actions such as COUNT(*) or joins.

Rows Written

Snapshot Information Level	API Structure(s)	Monitor Switch
Table	sqlm_table	Table
Application	sqlm_appl	Basic
	sqlm_stmt	Basic
	sqlm_subsection	Statement
Resettable	Yes	
Event Type	**Event Record(s)**	
Connection	sqlm_conn_event	
Table	sqlm_table_event	
Statement	sqlm_stmt_event	
Transaction	sqlm_xaction_event	
API Element Name	rows_written	
Element Type	counter	
Related Information	• "Resetting Monitor Data" on page 21 • "When Counters are Initialized" on page 20 • "Rows Read" on page 156 • "Internal Rows Inserted" on page 159 • "Internal Rows Deleted" on page 157 • "Internal Rows Updated" on page 158	

Description: This is the number of rows changed (inserted, deleted or updated) in the table.

Usage: A high value for table-level information indicates there is heavy usage of the table and you may want to use the Run Statistics (RUNSTATS) utility to maintain efficiency of the packages used for this table.

For application-connections and statements, this element includes the number of rows inserted, updated, and deleted in temporary tables.

At the application, transaction, and statement levels, this element can be useful for analyzing the relative activity levels, and for identifying candidates for tuning.

Rows Read

Snapshot Information Level	API Structure(s)	Monitor Switch
Table	sqlm_table	Table
Application	sqlm_appl	Basic
	sqlm_stmt	Basic
	sqlm_subsection	Statement
Resettable	Yes	
Event Type	**Event Record(s)**	
Connection	sqlm_conn_event	
Table	sqlm_table_event	
Statement	sqlm_stmt_event	
Transaction	sqlm_xaction_event	
API Element Name	rows_read	
Element Type	counter	
Related Information	• "Resetting Monitor Data" on page 21 • "When Counters are Initialized" on page 20 • "Rows Written" on page 155 • "Accesses to Overflowed Records" on page 157	

Description: This is the number of rows read from the table.

Usage: This element helps you identify tables with heavy usage for which you may want to create additional indexes. To avoid the maintenance of unnecessary indexes, you may use the SQL EXPLAIN statement, described in the *Administration Guide* to determine if the package uses an index.

This count is **not** the number of row that were returned to the calling application. Rather, it is the number of rows that had to be read in order to return the result set. For example, the following statement returns one row to the application, but many rows are read to determine the average salary:

```
SELECT AVG(SALARY) FROM USERID.EMPLOYEE
```

This count includes the value in *Accesses to Overflowed Records.*

Accesses to Overflowed Records

Snapshot Information Level	API Structure(s)	Monitor Switch
Table	sqlm_table	Table
Resettable	Yes	
Event Type	**Event Record(s)**	
Table	sqlm_table_event	
API Element Name	overflow_accesses	
Element Type	counter	
Related Information	• "Resetting Monitor Data" on page 21 • "When Counters are Initialized" on page 20 • "Rows Read" on page 156 • "Rows Written" on page 155	

Description: The number of accesses (reads and writes) to overflowed rows of this table.

Usage: Overflowed rows indicate that data fragmentation has occurred. If this number is high, you may be able to improve table performance by reorganizing the table using the REORG utility, which cleans up this fragmentation.

A row overflows if it is updated and no longer fits in the data page where it was originally written. This usually happens as a result of an update of a VARCHAR or an ALTER TABLE statement.

Internal Rows Deleted

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Basic
Application	sqlm_appl	Basic
	sqlm_stmt	Basic
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Connection	sqlm_conn_event	
Statement	sqlm_stmt_event	
API Element Name	int_rows_deleted	
Element Type	counter	
Related Information	• "When Counters are Initialized" on page 20 • "Rows Deleted" on page 153	

Description: This is the number of rows deleted from the database as a result of internal activity.

Usage: This element can help you gain insight into internal activity within the database manager of which you might not be aware. If this activity is high, you may want to evaluate your table design to determine if the referential constraints or triggers that you have defined on your database are necessary.

Internal delete activity can be a result of:

- A cascading delete enforcing an ON CASCADE DELETE referential constraint
- A trigger being fired.

Internal Rows Updated

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Basic
Application	sqlm_appl	Basic
	sqlm_stmt	Basic
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Connection	sqlm_conn_event	
Statement	sqlm_stmt_event	
API Element Name	int_rows_updated	
Element Type	counter	
Related Information	• "When Counters are Initialized" on page 20 • "Rows Updated" on page 154	

Description: This is the number of rows updated from the database as a result of internal activity.

Usage: This element can help you gain insight into internal activity within the database manager of which you might not be aware. If this activity is high, you may want to evaluate your table design to determine if the referential constraints that you have defined on your database are necessary.

Internal update activity can be a result of:

- A *set null* row update enforcing a referential constraint defined with the ON DELETE SET NULL rule
- A trigger being fired.

Internal Rows Inserted

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Basic
Application	sqlm_appl	Basic
	sqlm_stmt	Basic
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Connection	sqlm_conn_event	
Statement	sqlm_stmt_event	
API Element Name	int_rows_inserted	
Element Type	counter	
Related Information	• "When Counters are Initialized" on page 20 • "Rows Inserted" on page 153	

Description: The number of rows inserted into the database as a result of internal activity caused by triggers.

Usage: This element can help you gain insight into the internal activity within the database manager. If this activity is high, you may want to evaluate your design to determine if you can alter it to reduce this activity.

Table File ID

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_appl	Lock
Table	sqlm_table	Table
Lock	sqlm_appl_lock	Lock
	sqlm_lock	Lock
Resettable	No	
API Element Name	table_file_id	
Element Type	information	
Related Information	• "Resetting Monitor Data" on page 21 • "Table Name" on page 151 • "Table Schema Name" on page 152 • "Table Type" on page 150	

Description: This is the file ID (FID) for the table.

Usage: This element is provided for information purposes only. It is returned for compatibility with previous versions of the database system monitor, and it may **not** uniquely identify the table. Use *Table Name* and *Table Schema Name* to identify the table.

SQL Cursors

The following elements provide information about the SQL cursors:

Open Remote Cursors

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_appl	Basic
Resettable	No	
API Element Name	open_rem_curs	
Element Type	gauge	
Related Information	• "Open Remote Cursors with Blocking" on page 161 • "Open Local Cursors" on page 163	

Description: The number of remote cursors currently open for this application, including those cursors counted by *Open Remote Cursors with Blocking.*

Usage: You may use this element in conjunction with *Open Remote Cursors with Blocking* to calculate the percentage of remote cursors that are blocking cursors. If the percentage is low, you may be able to improve performance by improving the row blocking in the application. See *Open Remote Cursors with Blocking* for more information.

For the number of open cursors used by applications connected to a local database, see *Open Local Cursors.*

Open Remote Cursors with Blocking

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_appl	Basic
Resettable	No	
API Element Name	open_rem_curs_blk	
Element Type	gauge	
Related Information	• "Open Remote Cursors" on page 160 • "Rejected Block Cursor Requests" on page 162 • "Accepted Block Cursor Requests" on page 162 • "Open Local Cursors" on page 163 • "Open Local Cursors with Blocking" on page 163	

Description: The number of remote blocking cursors currently open for this application.

Usage: You can use this element in conjunction with *Open Remote Cursors* to calculate the percentage of remote cursors that are blocking cursors. If the percentage is low, you may be able to improve performance by improving the row blocking in the application:

- Check the pre-compile options for record blocking for treatment of ambiguous cursors
- Redefine cursors to allow for blocking (for example, if possible, specify FOR FETCH ONLY on your cursors).

Rejected Block Cursor Requests and *Accepted Block Cursor Requests* provide additional information that may help you tune your configuration parameters to improve row blocking in your application.

For the number of open blocking cursors used by applications connected to a local database see *Open Local Cursors with Blocking.*

Rejected Block Cursor Requests

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_appl	Basic
Resettable	No	
Event Type Connection	**Event Record(s)** sqlm_conn_event	
API Element Name **Element Type**	rej_curs_blk counter	
Related Information	• "Accepted Block Cursor Requests" on page 162 • "Open Local Cursors" on page 163 • "Open Local Cursors with Blocking" on page 163	

Description: The number of times that a request for an I/O block at server was rejected and the request was converted to non-blocked I/O.

Usage: If there are many cursors blocking data, the communication heap may become full. When this heap is full, an error is not returned. Instead, no more I/O blocks are allocated for blocking cursors. If cursors are unable to block data, performance can be affected.

If a large number of cursors were unable to perform data blocking, you may be able to improve performance by:

- Increasing the size of the *query_heap* database manager configuration parameter. For more information see the *Administration Guide*.

Accepted Block Cursor Requests

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_appl	Basic
Resettable	No	
Event Type Connection	**Event Record(s)** sqlm_conn_event	
API Element Name **Element Type**	acc_curs_blk counter	
Related Information	• "Rejected Block Cursor Requests" on page 162 • "Open Local Cursors" on page 163 • "Open Local Cursors with Blocking" on page 163	

Description: The number of times that a request for an I/O block was accepted.

Usage: You can use this element in conjunction with *Rejected Block Cursor Requests* to calculate the percentage of blocking requests that are accepted and/or rejected.

See *Rejected Block Cursor Requests* for suggestions on how to use this information to tune your configuration parameters.

Open Local Cursors

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_appl	Basic
Resettable	No	
API Element Name	open_loc_curs	
Element Type	gauge	
Related Information	• "Open Local Cursors with Blocking" on page 163 • "Open Remote Cursors" on page 160 • "Open Remote Cursors with Blocking" on page 161 • "Rejected Block Cursor Requests" on page 162 • "Accepted Block Cursor Requests" on page 162	

Description: The number of local cursors currently open for this application, including those cursors counted by *Open Local Cursors with Blocking.*

Usage: You may use this element in conjunction with*Open Local Cursors with Blocking* to calculate the percentage of local cursors that are blocking cursors. If the percentage is low, you may be able to improve performance by improving the row blocking in the application.

For cursors used by remote applications, see *Open Remote Cursors.*

Open Local Cursors with Blocking

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_appl	Basic
Resettable	No	
API Element Name	open_loc_curs_blk	
Element Type	gauge	
Related Information	• "Open Local Cursors" on page 163 • "Open Remote Cursors" on page 160 • "Open Remote Cursors with Blocking" on page 161 • "Rejected Block Cursor Requests" on page 162 • "Accepted Block Cursor Requests" on page 162	

Description: The number of local blocking cursors currently open for this application.

Usage: You may use this element in conjunction with*Open Local Cursors* to calculate the percentage of local cursors that are blocking cursors. If the percentage is low, you may be able to improve performance by improving the row blocking in the application:

- Check the pre-compile options for record blocking for treatment of ambiguous cursors
- Redefine cursors to allow for blocking (for example, if possible, specify FOR FETCH ONLY on your cursors).

Rejected Block Cursor Requests and Accepted Block Cursor Requests provide additional information that may help you tune your configuration parameters to improve row blocking in your application.

For blocking cursors used by remote applications, see *Open Remote Cursors with Blocking.*

SQL Statement Activity

The following elements provide information about SQL statement activity:

Static SQL Statements Attempted

Snapshot Information Level	**API Structure(s)**	**Monitor Switch**
Database	sqlm_dbase	Basic
Application	sqlm_appl	Basic
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Connection	sqlm_conn_event	
API Element Name	static_sql_stmts	
Element Type	counter	
Related Information	• "When Counters are Initialized" on page 20 • "Failed Statement Operations" on page 165	

Description: The number of static SQL statements that were attempted.

Usage: You can use this element to calculate the total number of successful SQL statements at the database or application level:

```
  Dynamic SQL Statements Attempted
+ Static SQL Statements Attempted
- Failed Statement Operations
= throughput during monitoring period
```

Dynamic SQL Statements Attempted

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Basic
Application	sqlm_appl	Basic
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Connection	sqlm_conn_event	
API Element Name	dynamic_sql_stmts	
Element Type	counter	
Related Information	• "When Counters are Initialized" on page 20 • "Failed Statement Operations" on page 165	

Description: The number of dynamic SQL statements that were attempted.

Usage: You can use this element to calculate the total number of successful SQL statements at the database or application level:

```
  Dynamic SQL Statements Attempted
+ Static SQL Statements Attempted
- Failed Statement Operations
= throughput during monitoring period
```

Failed Statement Operations

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Basic
Application	sqlm_appl	Basic
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Connection	sqlm_conn_event	
API Element Name	failed_sql_stmts	
Element Type	counter	
Related Information	• "When Counters are Initialized" on page 20 • "Dynamic SQL Statements Attempted" on page 165 • "Static SQL Statements Attempted" on page 164	

Description: The number of SQL statements that were attempted, but failed.

Usage: You can use this element to calculate the total number of successful SQL statements at the database or application level:

```
  Dynamic SQL Statements Attempted
+ Static SQL Statements Attempted
- Failed Statement Operations
= throughput during monitoring period
```

This count includes all SQL statements that received a negative SQLCODE.

This element may also help you in determining reasons for poor performance, since failed statements mean time wasted by the database manager and as a result, lower throughput for the database.

Commit Statements Attempted

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Basic
Application	sqlm_appl	Basic
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Connection	sqlm_conn_event	
API Element Name	commit_sql_stmts	
Element Type	counter	
Related Information	• "When Counters are Initialized" on page 20 • "Internal Commits" on page 171 • "Rollback Statements Attempted" on page 167 • "Internal Rollbacks" on page 172 • "Internal Rollbacks Due To Deadlock" on page 173	

Description: The total number of SQL COMMIT statements that have been attempted.

Usage: A small rate of change in this counter during the monitor period may indicate that applications are not doing frequent commits, which may lead to problems with logging and data concurrency.

You can also use this element to calculate the total number of units of work by calculating the sum of the following:

```
   commit statements attempted
 + internal commits
 + rollback statements attempted
 + internal rollbacks
```

Note: The units of work calculated will only include those since the later of:

- The connection to the database (for database-level information, this is the time of the first connection)
- The last reset of the database monitor counters.

This calculation can be done at a database or application level.

Rollback Statements Attempted

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Basic
Application	sqlm_appl	Basic
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Connection	sqlm_conn_event	
API Element Name	rollback_sql_stmts	
Element Type	counter	
Related Information	• "When Counters are Initialized" on page 20 • "Statement Type" on page 175 • "Commit Statements Attempted" on page 166 • "Internal Commits" on page 171 • "Internal Rollbacks" on page 172 • "Internal Rollbacks Due To Deadlock" on page 173	

Description: The total number of SQL ROLLBACK statements that have been attempted.

Usage: A rollback can result from an application request, a deadlock, or an error situation. This element **only** counts the number of rollback statements issued from applications.

At the application level, this element can help you determine the level of database activity for the application and the amount of conflict with other applications. At the database level, it can help you determine the amount of activity in the database and the amount of conflict between applications on the database.

Note: You should try to minimize the number of rollbacks, since higher rollback activity results in lower throughput for the database.

It may also be used to calculate the total number of units of work, by calculating the sum of the following:

```
   commit statements attempted
 + internal commits
 + rollback statements attempted
 + internal rollbacks
```

Select SQL Statements Executed

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Basic
Table Space	sqlm_tablespace	Basic
Application	sqlm_appl	Basic
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Connection	sqlm_conn_event	
API Element Name	select_sql_stmts	
Element Type	counter	
Related Information	• "When Counters are Initialized" on page 20 • "Static SQL Statements Attempted" on page 164 • "Dynamic SQL Statements Attempted" on page 165	

Description: The number of SQL SELECT statements that were executed.

Usage: You can use this element to determine the level of database activity at the application or database level.

You can also use the following formula to determine the ratio of SELECT statements to the total statements:

```
  select SQL statements executed
/ ( static SQL statements attempted
  + dynamic SQL statements attempted )
```

This information can be useful for analyzing application activity and throughput.

Update/Insert/Delete SQL Statements Executed

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Basic
Application	sqlm_appl	Basic
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Connection	sqlm_conn_event	
API Element Name	uid_sql_stmts	
Element Type	counter	
Related Information	• "When Counters are Initialized" on page 20 • "Static SQL Statements Attempted" on page 164 • "Dynamic SQL Statements Attempted" on page 165	

Description: The number of SQL UPDATE, INSERT, and DELETE statements that were executed.

Usage: You can use this element to determine the level of database activity at the application or database level.

You can also use the following formula to determine the ratio of UPDATE, INSERT and DELETE statements to the total number of statements:

```
   update/insert/delete SQL statements executed
 / (static SQL statements attempted + dynamic SQL statements attempted )
```

This information can be useful for analyzing application activity and throughput.

Data Definition Language (DDL) SQL Statements

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Basic
Application	sqlm_appl	Basic
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Connection	sqlm_conn_event	
API Element Name	ddl_sql_stmts	
Element Type	counter	
Related Information	• "When Counters are Initialized" on page 20	

Description: This element indicates the number of SQL Data Definition Language (DDL) statements that were executed.

Usage: You can use this element to determine the level of database activity at the application or database level. DDL statements are expensive to run due to their impact on the system catalog tables. As a result, if the value of this element is high, you should determine the cause, and possibly restrict this activity from being performed.

You can also use this element to determine the percentage of DDL activity using the following formula:

```
data definition language (DDL) SQL statements / total number of statements
```

This information can be useful for analyzing application activity and throughput. DDL statements can also impact the package cache, by invalidating sections that are stored there and causing additional system overhead due to section recompilation.

Examples of DDL statements are CREATE TABLE, CREATE VIEW, ALTER TABLE, and DROP INDEX.

Internal Automatic Rebinds

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Basic
Application	sqlm_appl	Basic
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Connection	sqlm_conn_event	
API Element Name	int_auto_rebinds	
Element Type	counter	
Related Information	• "When Counters are Initialized" on page 20 • "Binds/Precompiles Attempted" on page 174	

Description: The number of automatic rebinds (or recompiles) that have been attempted.

Usage: Automatic rebinds are the internal binds the system performs when an package has been invalidated. The rebind is performed the first time that the database manager needs to execute an SQL statement from the package. For example, packages are invalidated when you:

- Drop an object, such as a table, view, or index, on which the plan is dependent
- Add or drop a foreign key
- Revoke object privileges on which the plan is dependent.

You can use this element to determine the level of database activity at the application or database level. Since internal automatic rebinds can have a significant impact on performance, they should be minimized where possible.

You can also use this element to determine the percentage of rebind activity using the following formula:

```
internal automatic rebinds / total number of statements
```

This information can be useful for analyzing application activity and throughput.

Internal Commits

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Basic
Application	sqlm_appl	Basic

Resettable	Yes

Event Type	Event Record(s)
Database	sqlm_db_event
Connection	sqlm_conn_event

API Element Name	int_commits
Element Type	counter

Related Information

- "When Counters are Initialized" on page 20
- "Commit Statements Attempted" on page 166
- "Rollback Statements Attempted" on page 167
- "Internal Rollbacks" on page 172

Description: The total number of commits initiated internally by the database manager.

Usage: An internal commit may occur during any of the following:

- A reorganization
- An import
- A bind or pre-compile
- An application ends without executing an explicit SQL COMMIT statement (on UNIX).

This value, which does not include explicit SQL COMMIT statements, represents the number of these internal commits since the later of:

- The connection to the database (for database-level information, this is the time of the first connection)
- The last reset of the database monitor counters.

You can use this element to calculate the total number of units of work by calculating the sum of the following:

```
  commit statements attempted
+ internal commits
+ rollback statements attempted
+ internal rollbacks
```

Note: The units of work calculated will only include those since the later of:

- The connection to the database (for database-level information, this is the time of the first connection)
- The last reset of the database monitor counters.

This calculation can be done at the application or the database level.

Internal Rollbacks

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Basic
Application	sqlm_appl	Basic
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Connection	sqlm_conn_event	
API Element Name	int_rollbacks	
Element Type	counter	
Related Information	• "When Counters are Initialized" on page 20 • "Commit Statements Attempted" on page 166 • "Internal Commits" on page 171 • "Rollback Statements Attempted" on page 167 • "Internal Rollbacks Due To Deadlock" on page 173	

Description: The total number of rollbacks initiated internally by the database manager.

Usage: An internal rollback occurs when any of the following **cannot** complete successfully:

- A reorganization
- An import
- A bind or pre-compile
- An application ends as a result of a deadlock situation or lock timeout situation
- An application ends without executing an explicit commit or rollback statement (on Windows).

This value represents the number of these internal rollbacks since the later of:

- The connection to the database (for database-level information, this is the time of the first connection)
- The last reset of the database monitor counters.

While this value does not include explicit SQL ROLLBACK statements, the count from Internal Rollbacks Due To Deadlock is included.

You can use this element to calculate the total number of units of work by calculating the sum of the following:

```
   commit statements attempted
 + internal commits
 + rollback statements attempted
 + internal rollbacks
```

Note: The units of work calculated will include those since the later of:

- The connection to the database (for database-level information, this is the time of the first connection)
- The last reset of the database monitor counters.

This calculation can be done at the application or the database level.

Internal Rollbacks Due To Deadlock

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Basic
Application	sqlm_appl	Basic
Resettable	Yes	
Event Type	**Event Record(s)**	
Connection	sqlm_conn_event	
API Element Name	int_deadlock_rollbacks	
Element Type	counter	
Related Information	• "When Counters are Initialized" on page 20 • "Deadlocks Detected" on page 133 • "Rollback Statements Attempted" on page 167 • "Internal Rollbacks" on page 172	

Description: The total number of forced rollbacks initiated by the database manager due to a deadlock. A rollback is performed on the current unit of work in an application selected by the database manager to resolve the deadlock.

Usage: This element shows the number of deadlocks that have been broken and can be used as an indicator of concurrency problems. It is important, since internal rollbacks due to deadlocks lower the throughput of the database.

This value is included in the value given by Internal Rollbacks.

SQL Requests Since Last Commit

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_appl	Basic
Resettable	No	
API Element Name	sql_reqs_since_commit	
Element Type	information	
Related Information	• None	

Description: Number of SQL requests that have been submitted since the last commit.

Usage: You can use this element to monitor the progress of a transaction.

Note: This element is similar to the *cur_reqs* field in the *sqlestat* output. See Appendix C, "DB2 Version 1 sqlestat Users" on page 271 for more information on sqlestat equivalent data elements.

Statement Node

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_stmt	Statement
Resettable	No	
API Element Name	stmt_node_number	
Element Type	information	
Related Information	• None	

Description: Node where the statement was executed.

Usage: Used to correlate each statement with the node where it was executed.

Binds/Precompiles Attempted

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Basic
Application	sqlm_appl	Basic
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Connection	sqlm_conn_event	
API Element Name	binds_precompiles	
Element Type	counter	
Related Information	• "When Counters are Initialized" on page 20 • "Internal Automatic Rebinds" on page 170	

Description: The number of binds and pre-compiles attempted.

Usage: You can use this element to gain insight into the current level of activity within the database manager.

This value does not include the count of *Internal Automatic Rebinds,* but it does include binds that occur as a result of the REBIND PACKAGE command.

SQL Statement Details

The following elements provide details about the SQL statements:

- "Statement Type" on page 175
- "Statement Operation" on page 176
- "Package Name" on page 177
- "Section Number" on page 177
- "Cursor Name" on page 178
- "Application Creator" on page 179
- "Statement Operation Start Timestamp" on page 179
- "Statement Operation Stop Timestamp" on page 180
- "SQL Dynamic Statement Text" on page 180
- "Statement Sorts" on page 181

- "Number of Successful Fetches" on page 182
- "SQL Communications Area (SQLCA)" on page 182
- "Query Number of Rows Estimate" on page 183
- "Query Cost Estimate" on page 183

Statement Type

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_appl sqlm_stmt	Statement Statement
Resettable	No	
Event Type Statement	**Event Record(s)** sqlm_stmt_event	
API Element Name **Element Type**	stmt_type information	
Related Information	• "Resetting Monitor Data" on page 21 • "SQL Dynamic Statement Text" on page 180 • "Application Creator" on page 179 • "Section Number" on page 177 • "Package Name" on page 177	

Description: The type of statement processed.

Usage: You can use this element to determine the type of statement that is executing. It can be one of the following:

- A static SQL statement
- A dynamic SQL statement
- An operation other than an SQL statement; for example, a bind or pre-compile operation.

For the snapshot monitor, this element describes the statement that is currently being processed or was most recently processed.

Note: API users should refer to the *sqlmon.h* header file containing definitions of database system monitor constants.

Statement Operation

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_appl sqlm_stmt	Basic Statement
Resettable	No	
Event Type	**Event Record(s)**	
Statement	sqlm_stmt_event	
API Element Name	stmt_operation (Snapshot) operation (Event)	
Element Type	information	
Related Information	• "Resetting Monitor Data" on page 21 • "Statement Type" on page 175 • "SQL Dynamic Statement Text" on page 180 • "Application Creator" on page 179 • "Section Number" on page 177 • "Package Name" on page 177 • "Number of Successful Fetches" on page 182	

Description: The statement operation currently being processed or most recently processed (if none currently running).

Usage: You can use this element to determine the operation that is executing or recently finished.

It can be one of the following.

For SQL operations:

- SELECT
- PREPARE
- EXECUTE
- EXECUTE IMMEDIATE
- OPEN
- FETCH
- CLOSE
- DESCRIBE
- STATIC COMMIT
- STATIC ROLLBACK
- FREE LOCATOR

For non-SQL operations:

- RUN STATISTICS
- REORG
- REBIND
- REDISTRIBUTE
- GET TABLE AUTHORIZATION
- GET ADMINISTRATIVE AUTHORIZATION

Note: API users should refer to the *sqlmon.h* header file containing definitions of database system monitor constants.

Package Name

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_appl sqlm_stmt	Statement Statement
Resettable	No	
Event Type Statement	**Event Record(s)** sqlm_stmt_event	
API Element Name **Element Type**	package_name information	
Related Information	• "Resetting Monitor Data" on page 21 • "Application Creator" on page 179 • "Section Number" on page 177 • "SQL Dynamic Statement Text" on page 180	

Description: The name of the package that contains the SQL statement currently executing.

Usage: You may use this element to help identify the application program and the SQL statement that is executing.

If you are using the database system monitor APIs, note that the API constant SQLM_IDENT_SZ is used to define the length of this element. Only the first 8 characters are currently used.

Section Number

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_appl sqlm_stmt	Statement Statement
Resettable	No	
Event Type Statement	**Event Record(s)** sqlm_stmt_event	
API Element Name **Element Type**	section_number information	
Related Information	• "Resetting Monitor Data" on page 21 • "SQL Dynamic Statement Text" on page 180 • "Application Creator" on page 179 • "Package Name" on page 177	

Description: The internal section number in the package for the SQL statement currently processing or most recently processed.

Usage: For static SQL, you can use this element along with Application Creator and Package Name to query the SYSCAT.STATEMENTS system catalog table and obtain the static SQL statement text, using the sample query as follows:

```
SELECT SEQNO, SUBSTR(TEXT,1,120)
       FROM SYSCAT.STATEMENTS
       WHERE PKGNAME   = 'package_name' AND
             PKGSCHEMA = 'creator'      AND
             SECTNO    = section_number
       ORDER BY SEQNO
```

Note: Exercise caution in obtaining static statement text, because this query against the system catalog table could cause lock contentions. Whenever possible, only use this query when there is little other activity against the database.

Cursor Name

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_appl sqlm_stmt	Statement Statement
Resettable	No	
Event Type Statement	**Event Record(s)** sqlm_stmt_event	
API Element Name **Element Type**	cursor_name information	
Related Information	• “Resetting Monitor Data” on page 21 • “SQL Dynamic Statement Text” on page 180 • “Statement Type” on page 175 • “Number of Successful Fetches” on page 182	

Description: The name of the cursor corresponding to this SQL statement.

Usage: You may use this element to identify the SQL statement that is processing. This name will be used on an OPEN, FETCH, CLOSE, and PREPARE of an SQL SELECT statement. If a cursor is not used, this field will be blank.

If you are using the database system monitor APIs, note that the API constant SQLM_IDENT_SZ is used to define the length of this element. Only the first 8 characters are currently used.

Application Creator

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_appl sqlm_stmt	Statement Statement
Resettable	No	
Event Type	**Event Record(s)**	
Statement	sqlm_stmt_event	
API Element Name	creator	
Element Type	information	
Related Information	• "Resetting Monitor Data" on page 21 • "Package Name" on page 177 • "Section Number" on page 177	

Description: The authorization ID of the user that pre-compiled the application.

Usage: You may use this element to help identify the SQL statement that is processing, in conjunction with the CREATOR column of the package section information in the catalogs.

If you are using the database system monitor APIs, note that the API constant SQLM_IDENT_SZ is used to define the length of this element. Only the first 8 characters are currently used.

Statement Operation Start Timestamp

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_appl sqlm_stmt	Statement Statement
Resettable	No	
API Element Name	stmt_start	
Element Type	timestamp	
Related Information	• "Resetting Monitor Data" on page 21 • "Statement Operation Stop Timestamp" on page 180 • "Statement Operation" on page 176	

Description: The date and time when the Statement Operation started executing.

Usage: You can use this element with Statement Operation Stop Timestamp to calculate the elapsed statement operation execution time.

Statement Operation Stop Timestamp

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_appl sqlm_stmt	Statement Statement
Resettable	No	
Event Type	**Event Record(s)**	
Statement	sqlm_stmt_event	
API Element Name	stmt_stop (Snapshot) stop_time (event)	
Element Type	Timestamp	
Related Information	• "Resetting Monitor Data" on page 21 • "Statement Operation Start Timestamp" on page 179 • "Statement Operation" on page 176	

Description: The date and time when the Statement Operation stopped executing.

Usage: You can use this element with Statement Operation Start Timestamp to calculate the elapsed statement operation execution time.

SQL Dynamic Statement Text

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_stmt	Statement
Resettable	No	
Event Type	**Event Record(s)**	
Statement	sqlm_stmt_event	
API Element Name	Relative offsets are used to return the text. See data structures in sqlmon.h>	
Element Type	stmt_text (Event) information	
Related Information	• "Resetting Monitor Data" on page 21 • "Statement Operation" on page 176 • "Cursor Name" on page 178 • "Input Database Alias" on page 193 • "Application Creator" on page 179 • "Package Name" on page 177 • "Section Number" on page 177	

Description: This is the text of the dynamic SQL statement.

Usage: For snapshots, this statement text helps you identify what the application was executing when the snapshot was taken, or most recently processed if no statement was being processed right at the time the snapshot was taken.

For event monitors, it is returned in the Statement event record for all dynamic statements.

See Section Number for information on how to query the system catalog tables to obtain static SQL statement text that is not provided due to performance considerations.

Statement Sorts

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_appl sqlm_stmt	Statement Statement
Resettable	No	
API Element Name **Element Type**	stmt_sorts counter	
Related Information	• "Resetting Monitor Data" on page 21 • "Total Sorts" on page 82	

Description: The total number of times that a set of data was sorted in order to process the statement operation.

Usage: You can use this element to help identify the need for an index, since indexes can reduce the need for sorting of data. Using the related elements in the above table you can identify the SQL statement for which this element is providing sort information, and then analyze this statement to determine index candidates by looking at columns that are being sorted (for example, columns used in ORDER BY and GROUP BY clauses and join columns). See **explain** in the *Administration Guide* for information on checking whether your indexes are used to optimize sort performance.

This count includes sorts of temporary tables that were generated internally by the database manager to execute the statement. The number of sorts is associated with the first FETCH operation of the SQL statement. This information is returned to you when the operation for the statement is the first FETCH. You should note that for blocked cursors several fetches may be performed when the cursor is opened. In these cases it can be difficult to use the snapshot monitor to obtain the number of sorts, since a snapshot would need to be taken while DB2 was internally issuing the first FETCH.

A more reliable way to determine the number of sorts performed when using a blocked cursor would be with an event monitor declared for statements. The total sorts counter, in the statement event for the CLOSE cursor, contains the total number of sorts that were performed while executing the statement for which the cursor was defined.

Number of Successful Fetches

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_stmt	Statement
Resettable	Yes	
Event Type	**Event Record(s)**	
Statement	sqlm_stmt_event	
API Element Name	fetch_count	
Element Type	counter	
Related Information	• "Statement Type" on page 175 • "Statement Operation" on page 176 • "Cursor Name" on page 178 • "Statement Operation Start Timestamp" on page 179 • "Statement Operation Stop Timestamp" on page 180	

Description: The number of successful fetches performed on a specific cursor.

Usage: You can use this element to gain insight into the current level of activity within the database manager.

For performance reasons, a statement event monitor does not generated a statement event record for every FETCH statement. A record event is only generated when a FETCH returns a non-zero SQLCODE.

SQL Communications Area (SQLCA)

Event Type	Event Record(s)
Statement	sqlm_stmt_event
API Element Name	sqlca
Element Type	information
Related Information	• "Statement Operation" on page 176

Description: The SQLCA data structure that was returned to the application at statement completion.

Usage: The SQLCA data structure can be used to determined if the statement completed successfully. See the *SQL Reference* or *API Reference* for information about the content of the SQLCA.

Query Number of Rows Estimate

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_stmt	Statement
Resettable	No	
API Element Name	query_card_estimate	
Element Type	information	
Related Information	• "Resetting Monitor Data" on page 21 • "Query Cost Estimate" on page 183	

Description: An estimate of the number of rows that will be returned by a query.

Usage: This estimate by the SQL compiler can be compared with the run time actuals.

Query Cost Estimate

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_stmt	Statement
Resettable	No	
API Element Name	query_cost_estimate	
Element Type	information	
Related Information	• None	

Description: Estimated cost, in timerons, for a query, as determined by the SQL compiler.

Usage: This allows correlation of actual run-time with the compile-time estimates.

Subsection Details

When a statement is executed against a partitioned database, it is divided into subsections that may be executed on different nodes. An application may have several subsections simultaneously executing on a node. See "Monitoring Subsections" on page 26 and the *Administration Guide* for more information on subsections.

For problem determination, you may have to locate the problem subsection. For example, a subsection may be waiting on a tablequeue, because one of the writers to this tablequeue is in lock wait on another node. To get the overall picture for an application, you may have to issue an application snapshot on each node where the application is running.

The following database system monitor elements provide information about Subsections:

- "Subsection Number" on page 184
- "Subsection Node Number" on page 184
- "Subsection Status" on page 185
- "Execution Elapsed Time" on page 185
- "Number of Agents Working on a Subsection" on page 186

- "Waiting for Any Node to Send on a Tablequeue" on page 186
- "Waited for Node on a Tablequeue" on page 187
- "Total Number of Tablequeue Buffers Overflowed" on page 187
- "Current Number of Tablequeue Buffers Overflowed" on page 188
- "Number of Rows Read from Tablequeues" on page 188
- "Number of Rows Written to Tablequeues" on page 189

Subsection Number

Snapshot Information Level	**API Structure(s)**	**Monitor Switch**
Application	sqlm_subsection	Statement
Resettable	No	
Event Type	**Event Record(s)**	
Statement	sqlm_subsection_event	
API Element Name	ss_number	
Element Type	information	
Related Information	• None	

Description: Identifies the subsection associated with the returned information.

Usage: This number relates to the subsection number in the access plan that can be obtained with db2expln (see *Administration Guide*).

Subsection Node Number

Snapshot Information Level	**API Structure(s)**	**Monitor Switch**
Application	sqlm_subsection	Statement
Resettable	No	
Event Type	**Event Record(s)**	
Statement	sqlm_subsection_event	
API Element Name	ss_node_number	
Element Type	information	
Related Information	• None	

Description: Node where the subsection was executed.

Usage: Use to correlate each subsection with the database partition where it was executed.

Subsection Status

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_subsection	Statement
Resettable	No	
API Element Name	ss_status	
Element Type	information	
Related Information	• "Waited for Node on a Tablequeue" on page 187 • "Waiting for Any Node to Send on a Tablequeue" on page 186	

Description: The current status of an executing subsection.

Usage: The current status values can be:

- executing
- waiting for a lock
- waiting to receive data on a tablequeue
- waiting to send data on a tablequeue

Execution Elapsed Time

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_subsection	Statement
Resettable	No	
Event Type	**Event Record(s)**	
Statement	sqlm_subsection_event	
API Element Name	ss_exec_time	
Element Type	counter	
Related Information	• None	

Description: The time in seconds that it took a subsection to execute.

Usage: Allows you to track the progress of a subsection.

Number of Agents Working on a Subsection

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_subsection	Statement
Resettable	No	
Event Type	**Event Record(s)**	
Statement	sqlm_subsection_event	
API Element Name	num_subagents	
Element Type	gauge	
Related Information	• None	

Description: Total number of subagents currently working on a subsection.

Usage: Indicates the current degree of parallelism. Helps you track how execution is progressing.

Waiting for Any Node to Send on a Tablequeue

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_subsection	Statement
Resettable	No	
API Element Name	tq_wait_for_any	
Element Type	information	
Related Information	• "Subsection Status" on page 185 • "Waited for Node on a Tablequeue" on page 187	

Description: This flag is used to indicate that the subsection is blocked because it is waiting to receive rows from any node.

Usage: If Subsection Status indicates *waiting to receive data on a tablequeue* and this flag is TRUE, then the subsection is waiting to receive rows from any node. This generally indicates that the SQL statement has not processed to the point it can pass data to the waiting agent. For example, the writing agent may be performing a sort and will not write rows until the sort has completed. From the db2expln output, determine the subsection number associated with the tablequeue that the agent is waiting to receive rows from. You can then examine the status of that subsection by taking a snapshot on each node where it is executing.

Waited for Node on a Tablequeue

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_subsection	Statement
Resettable	No	
API Element Name **Element Type**	tq_node_waited_for information	
Related Information	• "Subsection Status" on page 185 • "Waiting for Any Node to Send on a Tablequeue" on page 186	

Description: If the subsection status Subsection Status is *waiting to receive* or *waiting to send* and Waiting for Any Node to Send on a Tablequeue is FALSE, then this is the number of the node that this agent is waiting for.

Usage: This can be used for troubleshooting. You may want to take an application snapshot on the node that the subsection is waiting for. For example, the application could be in a lock wait on that node.

Total Number of Tablequeue Buffers Overflowed

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_subsection	Statement
Resettable	No	
Event Type	**Event Record(s)**	
Statement	sqlm_subsection_event	
API Element Name **Element Type**	tq_tot_send_spills counter	
Related Information	• "Subsection Status" on page 185 • "Current Number of Tablequeue Buffers Overflowed" on page 188	

Description: Total number of tablequeue buffers overflowed to a temporary table.

Usage: Indicates the total number of tablequeue buffers that have been written to a temporary table. See "Current Number of Tablequeue Buffers Overflowed" on page 188 for more information.

Current Number of Tablequeue Buffers Overflowed

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_subsection	Statement
Resettable	No	
API Element Name **Element Type**	tq_cur_send_spills gauge	
Related Information	• "Subsection Status" on page 185 • "Total Number of Tablequeue Buffers Overflowed" on page 187	

Description: Current number of tablequeue buffers residing in a temporary table.

Usage: An agent writing to a tablequeue may be sending rows to several readers. The writing agent will overflow buffers to a temporary table when the agent that it is currently sending rows to is not accepting rows and another agent requires rows in order to proceed. Overflowing to temporary table allows both the writer and the other readers to continue processing.

Rows that have been overflowed will be sent to the reading agent when it is ready to accept more rows.

If this number is high, and queries fail with sqlcode -968, and there are messages in *db2diad.log* indicating that your ran out of temporary space in the TEMP table space, then tablequeue overflows may be the cause. This could indicate a problem on another node (such as locking). You would investigate by taking snapshots on all the partitions for this query.

There are also cases, perhaps because of the way data is partitioned, where many buffers need to be overflowed for the query. In these cases you will need to add more disk to the temporary table space.

Number of Rows Read from Tablequeues

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_subsection	Statement
Resettable	No	
Event Type	**Event Record(s)**	
Statement	sqlm_subsection_event	
API Element Name **Element Type**	tq_rows_read counter	
Related Information	• None	

Description: Total number of rows read from tablequeues.

Usage: If monitoring does not indicate that this number is increasing, then processing progress is not taking place.

If there is significant differences in this number between nodes, then some nodes may be over utilized while others are being under utilized.

If this number is large, then there is a lot of data being shipped between nodes, suggest that optimization might improve the access plan.

Number of Rows Written to Tablequeues

Snapshot Information Level	**API Structure(s)**	**Monitor Switch**
Application	sqlm_subsection	Statement
Resettable	No	
Event Type	**Event Record(s)**	
Statement	sqlm_subsection_event	
API Element Name	tq_rows_written	
Element Type	counter	
Related Information	• None	

Description: Total number of rows written to tablequeues.

Usage: If monitoring does not indicate that this number is increasing, then processing progress is not taking place.

If there is significant differences in this number between nodes, then some nodes may be over utilized while others are being under utilized.

If this number is large, then there is a lot of data being shipped between nodes, suggest that optimization might improve the access plan.

Intra-query Parallelism

The following database system monitor elements provide information about queries for which the degree of parallelism is greater than 1:

- “Number of Agents Working on a Statement” on page 190
- “Number of Agents Created” on page 190
- “Degree of Parallelism” on page 191

Number of Agents Working on a Statement

Snapshot Information Level	API Structure(s)	Monitor Switch
Statement	sqlm_stmt	Statement
	sqlm_subsection	Statement
Resettable	No	
API Element Name	num_agents	
Element Type	gauge	
Related Information	• "Number of Agents Created" on page 190 • "Degree of Parallelism" on page 191	

Description: Number of concurrent agents currently executing a statement or subsection.

Usage: An indicator how well the query is parallelized. This is useful for tracking the progress of query execution, by taking successive snapshots.

Number of Agents Created

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Statement
Application	sqlm_stmt	Statement
Resettable	No	
API Element Name	agents_top	
Element Type	water mark	
Related Information	• "Number of Agents Working on a Statement" on page 190 • "Degree of Parallelism" on page 191	

Description: This is the maximum number of agents that were used when executing the statement.

Usage: An indicator how well intra-query parallelism was realized.

Degree of Parallelism

Snapshot Information Level	API Structure(s)	Monitor Switch
Statement	sqlm_stmt	Statement
Resettable	No	
API Element Name	degree_parallelism	
Element Type	information	
Related Information	• "Number of Agents Working on a Statement" on page 190 • "Number of Agents Created" on page 190	

Description: The degree of parallelism requested when the query was bound.

Usage: Use with "Number of Agents Created" on page 190, to determine if the query achieved maximum level of parallelism.

CPU Usage

The CPU usage for an application is broken down into **user CPU**, which is the CPU consumed while executing application code, and **system** CPU, which is the CPU consumed executing system calls.

CPU consumption is available at the application, transaction, statement, and subsection levels.

- "CPU Time Used"

CPU Time Used

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_appl	Basic
Statement	sqlm_stmt	Statement
Subsection	sqlm_subsection	Statement
Resettable	Yes, at the application level No, at other levels	
Event Type	**Event Record(s)**	
Connection	sqlm_conn_event	
Transaction	sqlm_xaction_event	
Statement	sqlm_stmt_event	
Subsection	sqlm_subsection_event	
API Element Name	agent_usr_cpu_time agent_sys_cpu_time stmt_usr_cpu_time stmt_sys_cpu_time ss_usr_cpu_time ss_sys_cpu_time user_cpu_time system_cpu_time	
Element Type	time	
Related Elements	• None	

Description: The total CPU time (in seconds and microseconds) used by database manager agents, while working on behalf of the application, transaction, statement, or subsection.

System CPU represents the time spent in system calls. User CPU represents time spent executing database manager code.

These counters include time spent on both SQL and non-SQL statements, as well as any fenced user defined functions (UDF) or stored procedures executed by the application.

Usage: These elements can help you identify applications or queries that consume large amounts of CPU.

Note: If this information is not available for your operating system, these elements will be returned as 0. For example, they are not available on OS/2.

Snapshot Monitoring Elements

The following elements provide information about monitoring applications. They are returned as output for every snapshot:

- "Last Reset Timestamp"
- "Input Database Alias" on page 193
- "Snapshot Time" on page 193

Last Reset Timestamp

Snapshot Information Level	API Structure(s)	Monitor Switch
Database Manager	sqlm_db2	Basic
Database	sqlm_dbase	Basic
Application	sqlm_appl	Basic
Table Space	sqlm_tablespace_header	Buffer Pool
Table	sqlm_table_header	Table
Resettable	No	
API Element Name	last_reset	
Element Type	timestamp	
Related Information	• "Resetting Monitor Data" on page 21 • "Input Database Alias" on page 193	

Description: Indicates the date and time that the monitor counters were reset for the application issuing the GET SNAPSHOT.

Usage: You can use this element to help you determine the scope of information returned by the database system monitor.

If the counters have never been reset, this element will be zero.

The database manager counters will only be reset if you reset all active databases.

Input Database Alias

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Basic
Application	sqlm_appl_id_info	Basic
Table Space	sqlm_tablespace_header	Buffer Pool
Buffer Pool	sqlm_bufferpool	Buffer Pool
Table	sqlm_table_header	Table
Lock	sqlm_dbase_lock	Basic

Resettable	No
API Element Name	input_db_alias
Element Type	information
Related Information	• "Resetting Monitor Data" on page 21 • "Last Reset Timestamp" on page 192 • "Database Alias Used by Application" on page 54

Description: The alias of the database provided when calling the snapshot function.

Usage: This element can be used to identify the specific database to which the monitor data applies. It contains blanks unless you requested monitor information related to a specific database.

The value of this field may be different than the value of the *Database Alias Used by Application* monitor element since a database can have many different aliases. Different applications and users can use different aliases to connect to the same database.

If you are using the database system monitor APIs, note that the API constant SQLM_IDENT_SZ is used to define the length of this element. Only the first 8 characters are currently used.

Snapshot Time

Snapshot Information Level	API Structure(s)	Monitor Switch
Database Manager	sqlm_collected	Basic

Resettable	No
API Element Name	time_stamp
Element Type	timestamp
Related Information	• None

Description: The date and time when the database system monitor information was collected.

Usage: You can use this element to help relate data chronologically if you are saving the results in a file or database for ongoing analysis.

Chapter 4. Event Monitor Output

This chapter explains the contents and format of the trace produced by an event monitor and different options that can be specified on the CREATE EVENT MONITOR statement that can influence the trace. It shows how to program for reading this trace, through the use of code samples.

Output Stream Format

The output of an event monitor is a binary stream of data structures that are exactly the same for both pipe and file event monitors. You can format this trace using the db2evmon productivity tool.

Event Monitor records are defined in the **sqlmon.h** header file. You can look at the comments included in that file to see exactly which data elements are returned for each event type.

The following table illustrates the order in which records may appear in the event monitor stream. See “Information Available from Event Monitors” on page 17 for a list of events that trigger the writing of event records. Records in a trace are logically divided into three sections:

1. Prologue records - generated when an event monitor is activated.
2. Actual content records - generated as events occur.
3. Epilogue records - generated when a database is deactivated.

Record type	Record name	Information returned
Prologue		
Event Log Header	sqlm_event_log_header	Characteristics of the trace, for example server type and memory layout.
Database Header	sqlm_dbheader_event	Database name, path and activation time.
Event Monitor Start	sqlm_evmon_start_event	Time when the monitor was started or restarted.
Connection Header	sqlm_connheader_event	One for each current connection, includes connection time and application name.
Actual Contents (may appear mixed in with other connections).		
Connection Header	sqlm_connheader_event	One for each connection after activation, includes connection time and application identification.
Statement Event	sqlm_stmt_event	Statement level data, including text for dynamic statements.
Transaction Event	sqlm_xaction_event	Transaction level data.
Connection Event	sqlm_conn_event	Connection level data.
Deadlock Event	sqlm_deadlock_event	Deadlock level data.

Record type	Record name	Information returned
Deadlocked Connection Event	sqlm_dlconn_event	One for each connection involved, includes applications involved and locks in contention.
Overflow	sqlm_overflow_event	Number of records lost - generated when reader cannot keep up with a (non-blocked) event monitor.
Epilogue		
Database Event	sqlm_db_event	Database level data.
Buffer Pool Event	sqlm_bufferpool_event	Buffer pool level data.
Table Space Event	sqlm_tablespace_event	Table space level data.
Table Event	sqlm_table_event	Table level data.

Note: Event records may be generated for any connection and may therefore appear in mixed order in the stream. This means that you may get a transaction event for Connection 1, immediately followed by a connection event for Connection 2. However, records belonging to a single connection or a single event, will appear in their logical order. For example, a statement record (end of statement) always precedes a transaction record (end of UOW), if any. Similarly, a deadlock event record always precedes the deadlocked connection event records for each connection involved in the deadlock. The **application id** or **application handle (agent_id)** can be used to match records with a connection.

For example, using the following event monitor,

```
db2 connect to sample
db2 "create event monitor ALL for
  statements, transactions, connections,
  deadlocks, database, bufferpools,
  tablespaces, tables, write to
  file '/tmp/all'"
mkdir /tmp/all
db2 connect reset
```

the following workload,

Application 1

```
db2 set event monitor ALL state 1
db2 select evmonname from
  syscat.eventmonitors
db2 connect reset
```

Application 2

```
db2 connect to sample

db2 +c connect reset
```

the following trace might be generated. Listed in this sample are some of the fields in each event record to give a flavor of the type of information contained in a trace. See “Event Monitors” on page 10 for an example of deadlock events. Note, the numbers in this sample are used to illustrate the order in which records have been written.

PROLOGUE

The Prologue information is generated when `set event monitor all state 1` is executed. If this event monitor had been AUTOSTART, it would have been generated when the database was activated.

```
1) sqlm_event_log_header
      version:            SQLM_DBMON_VERSION5  - Trace was produced by UDB V5
      num_nodes_in_db2_instance: 1             - for a standalone system,
      byte_order:         SQLM_BIG_ENDIAN      - on a UNIX or AIX box,
      event_monitor_name: ALL                  - by event monitor: 'ALL'

2) sqlm_dbheader_event
      db_name:            SAMPLE               - for database 'SAMPLE'

3) sqlm_connheader_event
      agent_id: 14                             - Application 1 - handle
      appl_id:  *LOCAL.bourbon.970602180712    - Application 1 - id with timestamp
```

CONTENTS

Generated when Application 1 issues `select name from syscat.eventmonitors`. At the time that the event monitor is turned on, Application 2 has not yet connected.

```
4) sqlm_stmt_event
      agent_id: 14
      appl_id:  *LOCAL.bourbon.970602180712
      operation:         SQLM_PREPARE
      package_name:      SQLC2BA4
      cursor:            SQLCUR201
      @stmt_text_offset: SELECT EVMONNAME FROM SYSCAT.EVENTMONITORS

5) sqlm_stmt_event
      agent_id: 14
      appl_id:  *LOCAL.bourbon.970602180712
      operation:         SQLM_OPEN
      package_name:      SQLC2BA4
      cursor:            SQLCUR201
      @stmt_text_offset: SELECT EVMONNAME FROM SYSCAT.EVENTMONITORS

6) sqlm_stmt_event
      agent_id: 14
      appl_id:  *LOCAL.bourbon.970602180712
      operation:         SQLM_FETCH
      package_name:      SQLC2BA4
      cursor:            SQLCUR201
      @stmt_text_offset: SELECT EVMONNAME FROM SYSCAT.EVENTMONITORS
      fetch_count:       2
      sqlca.sqlcode:     100  - (all rows in the SYSCAT.EVENTMONITORS table)
      SQL0100W  No row was found for FETCH, UPDATE or DELETE; or the result of a
      query is an empty table.  SQLSTATE=02000
NOTE - A fetch event is generated only if the fetch fails or encounters end of table

7) sqlm_stmt_event
      agent_id: 14
      appl_id:  *LOCAL.bourbon.970602180712
      operation:         SQLM_DESCRIBE
      package_name:      SQLC2BA4
      cursor:            SQLCUR201
      @stmt_text_offset: SELECT EVMONNAME FROM SYSCAT.EVENTMONITORS

8) sqlm_stmt_event
      agent_id: 14
      appl_id:  *LOCAL.bourbon.970602180712
      operation:         SQLM_CLOSE
      package_name:      SQLC2BA4
      cursor:            SQLCUR201
      @stmt_text_offset: SELECT EVMONNAME FROM SYSCAT.EVENTMONITORS
      fetch_count:       2

9) sqlm_stmt_event
      agent_id: 14
      appl_id:  *LOCAL.bourbon.970602180712
      operation:    SQLM_STATIC_COMMIT       - generated by CLP after the SELECT
      package_name: SQLC2BA4

10) sqlm_xaction_event
      agent_id: 14
      appl_id:  *LOCAL.bourbon.970602180712
      status:    SQLM_UOWCOMMIT
      rows_read: 7
```

Application 2 is connecting to the database. Output is interleaved, as the DB2 agents are executing simultaneously:

```
11) sqlm_connheader_event
      agent_id: 15                              - Application 2 - handle
      appl_id:  *LOCAL.bourbon.970602180714    - Application 2 - id with timestamp

12) sqlm_stmt_event
      agent_id: 15
      appl_id:  *LOCAL.bourbon.970602180714
      operation:  SQLM_STATIC_COMMIT          - generated by CLP on CONNECT

13) sqlm_xaction_event
      agent_id: 15
      appl_id:  *LOCAL.bourbon.970602180714
      status:    SQLM_UOWCOMMIT

14) sqlm_stmt_event
      agent_id: 15
      appl_id:  *LOCAL.bourbon.970602180714
      operation:  SQLM_STATIC_COMMIT          - generated on CONNECT RESET

15) sqlm_xaction_event
      agent_id: 15
      appl_id:  *LOCAL.bourbon.970602180714
      status:    SQLM_UOWCOMMIT

16) sqlm_conn_event
      agent_id: 15
      appl_id:  *LOCAL.bourbon.970602180714
      commit_sql_stmts:    2

17) sqlm_stmt_event
      agent_id: 14
      appl_id:  *LOCAL.bourbon.970602180712
      operation:    SQLM_STATIC_COMMIT       - generated on CONNECT RESET
      package_name: SQLC2BA4

18) sqlm_xaction_event
      agent_id: 14
      appl_id:  *LOCAL.bourbon.970602180712
      status:    SQLM_UOWCOMMIT
      rows_read: 2
      locks_held_top: 7

19) sqlm_conn_event
      agent_id: 14
      appl_id:  *LOCAL.bourbon.970602180712
      select_sql_stmts: 1
      rows_selected:    2
```

Epilogue

The Epilogue information is generated during database deactivation (last application finished disconnecting):

```
20) sqlm_table_event
      table_schema: SYSIBM
      table_name:   SYSTABLES
      table_type:   SQLM_CATALOG_TABLE
      rows_read: 2

21) sqlm_table_event
      table_schema: SYSIBM
      table_name:   SYSDBAUTH
      table_type:   SQLM_CATALOG_TABLE
      rows_read: 3

22) sqlm_tablespace_event
      tablespace_name: SYSCATSPACE

23) sqlm_tablespace_event
      tablespace_name: TEMPSPACE1

24) sqlm_tablespace_event
      tablespace_name: USERSPACE1

25) sqlm_bufferpool_event
      bp_name: IBMDEFAULTBP

26) sqlm_db_event
      connections_top: 2
```

Note: A WHERE clause on the CREATE EVENT MONITOR SQL statement can be used to restrict the applications that will generate events; see Appendix A, "Database System Monitor Interfaces" on page 213 for details.

Matching Event Records with Their Application

Each record includes the application handle and application ID. These allow you to correlate each record with the application for which the record was generated.

The application handle (**agent_id**) is unique system-wide for the duration of the application. However, it will eventually be reused (a 16 bit counter is used to generate this identifier). In most cases, this reuse is not a problem, since an application reading records from the trace is able to detect a connection that was terminated. For example, encountering (in the trace) a connection header with a known agent_ID implies that the previous connection with this agent_ID was terminated.

The application ID is a string identifier that includes a timestamp and is guaranteed to remain unique, even after stopping and restarting the database manager.

File Event Monitor Buffering

The event monitor output thread buffers records, using two internal buffers, before writing them to disk. Records are written to the trace only when a buffer is full. To force an event monitor to flush its buffers you must turn it off. The size of these buffers can be specified on the CREATE EVENT MONITOR statement with the BUFFERSIZE argument. Specifying larger buffers reduces the number of disk accesses.

Figure 5 illustrates how event records are generated for a FILE statement event monitor: 2 applications are connected to a database, each having a single agent working on its behalf.

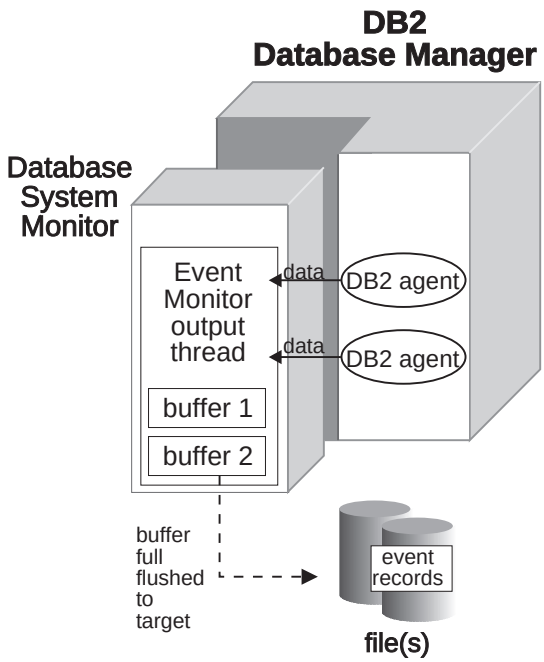


Figure 5. Event Monitor Buffers

In this example, each application agent has just finished executing a statement and is reporting the monitor data it has collected for its statement to the event monitor output thread. The output thread formats the records and writes them into one of its two buffers. The buffer gets written to a file when it is full. Having two buffers allows the output thread to continue receiving data from database agents, while a buffer is being written.

Blocked Event Monitors

A blocked event monitor will suspend the agent(s) sending monitor data, when both of its buffers are full, until a buffer has been written. This can introduce a significant performance overhead, depending on the type of workload and the speed of the I/O device. But, a blocked event monitor never discards event records, as long as it is running. This is the default.

Non-Blocked Event Monitors

A non-blocked event monitor will simply discard monitor data coming from the agents when the data is coming faster than it can write it. The following is an example of creating a non-blocked event monitor:

```
db2 "create event monitor STMT for
statements write to file '/tmp/all'
NONBLOCKED"
```

Overflows

An event monitor that has discarded event records generates an **overflow event**. It specifies the start and stop time during which the monitor was discarding events, and the number of events that were discarded during that period.

Unwritten Overflow Data: It is possible for an event monitor to terminate or be deactivated with a pending overflow to report. If this occurs, the following message is written to the db2diag.log:

```
DIA1603I Event Monitor monitor-name had a pending overflow
record when it was deactivated.
```

File Event Monitor Target

All the output of the event monitor goes in the directory supplied to the FILE argument on the CREATE EVENT MONITOR statement.

When a file event monitor is first activated, a control file is created in this directory. This binary file contains control information that is used to prevent two event monitors from simultaneously writing to the same target, and to keep track of the file and file location where the event monitor is supposed to write its next record. It is named *db2event.ctl*; do not remove or modify this file.

Limiting Trace Size

By default, an event monitor writes its trace to a single file, called *00000000.evt*. This file will keep growing as long as there is space on the file system. You can limit the maximum size of a trace using the MAXFILESIZE and MAXFILES arguments of the create event monitor statement.

Number of Files: The trace produced by an event monitor can be quite large, and you may want to break it down into several files of a fixed size. This also allows you to remove files after processing them, while the event monitor is still running.

Files are numbered sequentially, beginning with *00000000.evt*. If you are using several files, then when a file is full, output is automatically directed to the next file. For example, the following event monitor will break down its trace into 4MB files. It keeps creating files as long as there is space on the file system.

```
db2 "create event monitor BIGONE
for statements, transactions, connections,
deadlocks write to file '/tmp/bigevmon'
MAXFILESIZE 1000
MAXFILES NONE"
```

This might result in the following files in its target directory.

File	size (bytes)
/tmp/bigevmon/db2evmon.ctl	300
/tmp/bigevmon/00000000.evt	4079766
/tmp/bigevmon/00000001.evt	4095128
/tmp/bigevmon/00000002.evt	4095602

The highest numbered file is always the active file. When the number of files reaches the maximum defined by MAXFILES, the event monitor deactivates itself and the following message is written to the DB2DIAG.LOG.

```
DIA1601I Event Monitor monitor-name was deactivated when
it reached its preset MAXFILES and MAXFILESIZE limit
```

You can avoid this situation by removing full files (see "Processing Data While Monitor is Active"). Any event file except the active file can be removed while the event monitor is still running.

Running out of Disk Space

When a File event monitor runs out of disk space, it shuts itself down, after logging a system-error-level message in the error logs, *db2diag.log* and *db2err.log*.

Processing Data While Monitor is Active

You may want an event monitor to collect data continuously so that no events are ever missed. For example, if you have a usage account system that uses an event monitor to collect data, you may want to process the data each night beginning at 2:00 AM, at which point you delete the files that have been processed.

An event monitor cannot be forced to switch to the next file unless you stop and restart it. It must also be in APPEND mode. In order to keep track of which events have been

processed in the active file, you can create an application that simply keeps track of the file number and location of the last record processed. When processing the trace the next time around, the application can simply seek to that file location.

Using Pipe event monitors is an easy way to read data produced by an active event monitor (see “Using Pipe Event Monitors” on page 18).

Restarting a File Event Monitor

When a File event monitor is restarted, it can either erase any existing data, or append to it.

An APPEND event monitor starts writing at the end of the file it was last using (the file number is indicated in the *db2evmon.ctl* control file). If you have removed that file, then the next file number in sequence is used. For example, in the example above, if you remove all **.evt** files, and restart the event monitor, then event records will be written into *00000003.evt*. If you had not removed the files, then they would go into or append to *00000002.evt*. When an append event monitor is restarted, only the start_event is generated. The event log header and database header are only generated for the first activation.

A REPLACE event monitor always deletes existing event files, and starts writing at *00000000.evt*.

Programming to Read an Event Monitor Trace

Each record in the binary event monitor stream, except for the log header, starts with the size and type of the record. While reading the trace, it is extremely important that the size of a record is used for skipping a record in the trace, both to ensure that your application will be able to handle the traces produced by future releases of DB2, and because byte padding is sometimes used in the output stream. **WARNING: You should never use sizeof() on event monitor records**. Similarly, the type of every record should be checked. Skipping unknown or unwanted records will allow your application to handle any event monitor trace.

The log header describes the characteristics of the trace, containing information such as the memory model (for example big endian) of the server where the trace was collected, and the codepage of the database. You may have to do byte swapping on numerical values, if the system where you read the trace has a different memory model than the server (for example, Windows NT to UNIX). Codepage translation may also need to be done, if the database is configured in a different language than the machine from which you read the trace.

The following annotated fragments from the code samples in sqllib/samples/c_AIX illustrates the most important considerations in programming to read an event monitor trace. Some error handling is omitted, for simplicity. This code should run on all platforms, except for PIPE I/O routines (however, more platforms are addressed in the actual code samples).

Reading the Data Stream

The following routines illustrate how you can open, read, or skip bytes from a PIPE or FILE on a UNIX platform.

```
//------------------------------------------------------------------------------
// File functions  - Using the ANSI C library
//------------------------------------------------------------------------------
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
//------------------------------------------------------------------------------
FILE* openFile(char *file_name) {
   return fopen(file_name,"rb"); /* returns NULL if file cannot be opened */
}
//------------------------------------------------------------------------------
int closeFile(FILE* handle) {
  return fclose(handle);
}
//------------------------------------------------------------------------------
int readFromFile(char* buffer, int size, FILE* fp) {
   int rc=0;                              // returns 0 (success); EOF; or errno
   int records_read = fread(buffer, size, 1, fp);
   if (records_read != 1) {
      if (feof(fp))
         rc = EOF;
      else rc = errno;
   } /* end if no data was returned */
   return rc;
} /* end readFromFile */

//------------------------------------------------------------------------------
// Pipe functions  -  for AIX
//------------------------------------------------------------------------------
#include <unistd.h>    /* for pipe functions on AIX */
#include <fcntl.h>     /* for definition of O_RDONLY and open() */
//------------------------------------------------------------------------------
int openNamedPipe (char   *pipe_name) {
   return open(pipe_name, O_RDONLY);
}
//------------------------------------------------------------------------------
int closeNamedPipe (int handle) {
  return close(handle);
}
//------------------------------------------------------------------------------
int readFromPipe(int handle, char* buffer, int size) {
   int rc=0;
   int num_bytes;
   num_bytes = read(handle, buffer, size);
   if (num_bytes != size) {
      if (num_bytes==0)
         rc=EOF;
      else rc = num_bytes;
   } /* end did not get the expected number of bytes back from read() */
   return rc;
} /* end readFromPipe */

//------------------------------------------------------------------------------
// Read data from Event Monitor trace (FILE or PIPE) returns 0 (success) or EOF
//------------------------------------------------------------------------------
int read_data(EventLog* evtlog,
              char*     buffer,
              int       size) {
   int rc=0;
   if (evtlog->type == EVMFile) {
      rc = readFromFile(buffer, size, evtlog->current_fp);
      if (rc && rc!=EOF) {
```

```
         fprintf(stderr, "ERROR: Could not read from: %s\n",
                         evtlog->current_fn);
         exit(1);
      } /* end cannot read the log header from the file */
   } /* end if the Event Monitor Log is read from a file */
   else {
      rc = readFromPipe(evtlog->handle, buffer, size);
      if (rc && rc!=EOF) {
         fprintf(stderr, "ERROR: Could not read a data from: %s\n",
                          evtlog->target);
         exit(2);
      } /* end cannot read from the pipe */
   } /* end else the Event Log is read from a pipe */
   return rc;
} /* end of read_data */

//------------------------------------------------------------------------------
// Skip n bytes from current position in the trace
//------------------------------------------------------------------------------
void skip_data(EventLog* evtlog, int n) {
   if (evtlog->type == EVMFile)
      fseek(evtlog->current_fp, n, SEEK_CUR);
   else if (evtlog->type == EVMPipe) {
      lseek(evtlog->handle, n, SEEK_CUR);
   } /* end else pipe event monitor */
} /* end skip_data */
```

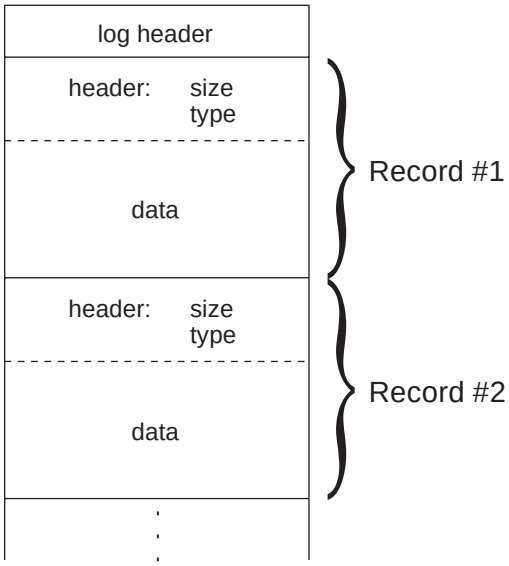
Swapping Bytes in Numerical Values

This code is required when transferring data between systems using different conventions for storing numerical values (for example, UNIX to Windows NT).

```
#include <sqlmon.h>   // DB2 Database Monitor interface
//------------------------------------------------------------------------------
// Byte conversion macros
//------------------------------------------------------------------------------
#define SWAP2(s) ((((s) >> 8) & 0xFF) | (((s) << 8) & 0xFF00))

#define SWAP4(l) ((((l) >> 24) & 0xFF) | ((((l) & 0xFF0000) >> 8) & 0xFF00) \
                     | ((((l) & 0xFF00) << 8) | (((l) << 24))

//------------------------------------------------------------------------------
void swapBytes_sqlm_event_log_header(sqlm_event_log_header* r) {
   r->size =                      SWAP4(r->size);
   r->version =                   SWAP4(r->version);
   r->codepage_id =               SWAP2(r->codepage_id);
   r->country_code =              SWAP2(r->country_code);
} // end of swapBytes_sqlm_event_log_header)

   // . . .
```

Reading the Event Records

After the log header has been read, all remaining records in the trace follow the following layout. Each record starts and terminates on a 4 byte boundary; and records are never split across 2 files.



The **size** accounts for both header and data.

```
//------------------------------------------------------------------------------
// Read an event record  - returns:  0 (success) or EOF
//------------------------------------------------------------------------------
int read_event_record(EventLog *evtlog, char *buffer, int *event_type) {

   sqlm_event_rec_header*  pHeader = (sqlm_event_rec_header*) buffer;

   //------------------------------------------------------------------------
   // Read the record header
   //------------------------------------------------------------------------
   int rc;
   rc=read_data(evtlog, (char *) pHeader, pHeader->size);
   if (rc)
      return rc;  /* could be at EOF */

   *event_type   = pHeader->type; // The event type is specified in the header

   if (evtlog->needByteReversal)
      swapBytes_sqlm_event_rec_header(pHeader);

   //------------------------------------------------------------------------
   // Read the rest of the data
   //------------------------------------------------------------------------
   rc=read_data(evtlog, buffer        + pHeader->size,
                        pHeader->size - pHeader->size);

   if (rc==0 && evtlog->needByteReversal)
      swapBytes(pHeader->type, buffer);

   return rc;
} /* end of read_event_record */
```

Reading the Log Header

You must take care of byte reversal, and possibly code page conversion. This example only handles byte reversal.

```
// From sqlmon.h, the DB2 system monitor header file:
// LOG HEADER
typedef struct sqlm_event_log_header
{
  int            byte_order;                  /* Big Endian or Little Endian */
  unsigned long  size;                        /* Size of this record         */
  unsigned long  version;                     /* Event Monitor Version       */
  char           event_monitor_name[SQLM_IDENT_SZ];  /* Name of the Event Mon */
  unsigned short codepage_id;                 /* Code page of Database       */
  unsigned short country_code;                /* Country Code of Database    */
  char           server_prdid[SQLM_IDENT_SZ];  /* Server Product Id          */
  char           server_instance_name[SQLM_IDENT_SZ]; /*instance name of DB2  */
  unsigned long  number_nodes_in_system;
}sqlm_event_log_header;

//------------------------------------------------------------------------------
// Attributes of an Event Log
// This structure is used to keep all information that is required to
// process the output of any Event Monitor (file or pipe).
//------------------------------------------------------------------------------
typedef enum   EventMonitorType { EVMPipe, EVMFile } EventMonitorType;
typedef struct EventLog {
   sqlm_event_log_header   header;
   char*                   target;
   Boolean                 needByteReversal; // True if running on a machine
                                             // with a different memory model
   EventMonitorType        type;             // type: file or pipe
   int                     handle;           // For Named pipe only
   FILE*                   current_fp;       // File currently open
   char                    current_fn[512];  // It's name.
} EventLog;

//------------------------------------------------------------------------------
// Read the log header - store its attributes into an 'EventLog' structure.
//------------------------------------------------------------------------------
#ifdef _AIX    // Compiler pre-defined macro on AIX
#define BIG_ENDIAN_MEMORY
#else
// Assume an INTEL platform
#define LITTLE_ENDIAN_MEMORY
#endif
void read_log_header( /* output */ EventLog* evtlog) {

   // Read the Event Trace header
   read_data(evtlog,                        // input: event trace (or log)
             (char*) &evtlog->header,       // output: log header
             sizeof(sqlm_event_log_header)); // input: number of bytes to read
                                             //        is what we can handle
   // Check if the memory model is different and we need byte-reveral
   switch (evtlog->header.byte_order) {
      case SQLM_BIG_ENDIAN:
      #ifdef LITTLE_ENDIAN_MEMORY
         evtlog->needByteReversal = 1;
      #else
         evtlog->needByteReversal = 0;
      #endif
         break;
      case SQLM_LITTLE_ENDIAN:
      #ifndef LITTLE_ENDIAN_MEMORY
         evtlog->needByteReversal = 1;
      #else
         evtlog->needByteReversal = 0;
```

```
      #endif
         break;
   } // end switch

   // Convert the header, if the server had a different memory model than ours
   if (evtlog->needByteReversal)
      swapBytes_sqlm_event_log_header(&evtlog->header);

   // Skip extra bytes, if the record is bigger than what we can handle
   // (which may become the case in subsequent releases of the product)
   if (evtlog->header.size > sizeof(sqlm_event_log_header)) {
      skip_data(evtlog, evtlog->header.size - sizeof(sqlm_event_log_header));
   } // end if more bytes than expected for this record in the log file
} // end read_log_header
```

Printing Event Records

All timestamps in event monitor records are **GMT time** since January 1, 1970.

All strings in event monitor records are padded with blanks, up to their maximum size. **Strings are NEVER NULL terminated.**

The following routines illustrate one method of handling blank-padded strings and converting GMT time into local time. It also shows how to print any event monitor.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>  /* To use cprint() function */
#include <time.h>
#include <sqlmon.h>   // DB2 Database Monitor interface

//--------------------------------------------------------------------------
// Convert GMT time into local time, and format it into a printable string.
//--------------------------------------------------------------------------
char* time_STRING(const sqlm_timestamp timestamp, char *timeString) {

   // Event Monitor returns GMT time, adjust it to local time
   struct tm *pstTm;
   pstTm = localtime((signed long*) &timestamp.seconds);
   if (timestamp.seconds == 0)
      strcpy(timeString, "");
   else {
      if (timestamp.microsec < 0) {
         sprintf(timeString, "%02d-%02d-%04d %02d:%02d:%02d",
              pstTm->tm_mon + 1, pstTm->tm_mday, pstTm->tm_year + 1900,
              pstTm->tm_hour, pstTm->tm_min, pstTm->tm_sec);
      } else {
         sprintf(timeString, "%02d-%02d-%04d %02d:%02d:%02d.%06.6ld",
              pstTm->tm_mon + 1, pstTm->tm_mday, pstTm->tm_year + 1900,
              pstTm->tm_hour, pstTm->tm_min, pstTm->tm_sec, timestamp.microsec);
     } /* end else micro seconds are not null */
  } /* end if the timestamp is non-zero */
  return timeString;
} // end of time_STRING

 All strings in  an event monitor traces are BLANK PADDED up to their maximum
 size.  They *are not* null terminated.

//--------------------------------------------------------------------------
// Print a Blank Padded String of maximum length SZ
//--------------------------------------------------------------------------
// note: strings returned by DB2 are NOT NULL-TERMINATED, they are all
//       blank padded up to some maximum length.
//--------------------------------------------------------------------------
```

```
// For example, given:
//                  char   str[20] = "Contents of 1       ";
// the following invocation:
//                  fpBPSTR(stdout, " String 1", str, sizeof(str));
// will print:
//                  " String 1: Contents of 1"
//------------------------------------------------------------------------------
#define fpBPSTR(fp, txt, str, SZ) \
{ \
    char newstr[SZ]; \
    int k=0; \
    while (str[k]!=' '&&k<SZ) { newstr[k]=str[k]; k++;} \
    if (k<SZ) newstr[k]='\0'; \
    fprintf(fp, txt ": %0.*s\n", SZ, newstr); \
}

//------------------------------------------------------------------------------
char* byte_order_STRING(int val) {
   switch (val) {
   case SQLM_LITTLE_ENDIAN:      return "SQLM_LITTLE_ENDIAN";
   case SQLM_BIG_ENDIAN:         return "SQLM_BIG_ENDIAN";
   }
   return "";
} // end of byte_order_STRING

//------------------------------------------------------------------------------
// Print the log header record
//------------------------------------------------------------------------------
void print_sqlm_event_log_header(FILE* fp,
                                 const sqlm_event_log_header *event_header){
   fprintf(fp,
"------------------------------------------------------------------------\n"
"                          EVENT LOG HEADER\n");
   fpBPSTR(fp, "  Event Monitor name",
                  event_header->event_monitor_name, SQLM_IDENT_SZ);
   fpBPSTR(fp, "  Server Product ID",
                  event_header->server_prdid, SQLM_IDENT_SZ);
   fprintf(fp, "  Version of event monitor data: %ld\n", event_header->version);
   fprintf(fp, "  Byte order: %s\n",
                  byte_order_STRING(event_header->byte_order));
   fprintf(fp, "  Size of record: %ld\n", event_header->size);
   fprintf(fp, "  Codepage of database: %d\n", event_header->codepage_id);
   fprintf(fp, "  Country code of database: %d\n", event_header->country_code);
   fpBPSTR(fp, "  Server instance name",
                  event_header->server_instance_name, SQLM_IDENT_SZ);
   fprintf(fp,
"------------------------------------------------------------------------\n");
   fflush(fp);
} /* end of print_sqlm_event_log_header */

//------------------------------------------------------------------------------
// Print an event record
//------------------------------------------------------------------------------
void print_event_record(FILE* fp, int event_type, char* rec, int rec_no) {
   switch (event_type) {
   case SQLM_EVENT_DB:
//      print_sqlm_db_event(fp, (const sqlm_db_event*) rec, rec_no);
      break;
   case SQLM_EVENT_CONN:
//      print_sqlm_conn_event(fp, (sqlm_conn_event*) rec, rec_no);
      break;
   case SQLM_EVENT_TABLE:
//      print_sqlm_table_event(fp, (sqlm_table_event*) rec, rec_no);
      break;
   case SQLM_EVENT_STMT:
//      print_sqlm_stmt_event(fp, (sqlm_stmt_event*) rec, rec_no);
      break;
```

```
   case SQLM_EVENT_STMTTEXT:
//      print_sqlm_stmttext_event(fp, (sqlm_stmttext_event*) rec, rec_no);
      break;
   case SQLM_EVENT_XACT:
//      print_sqlm_xaction_event(fp, (sqlm_xaction_event*) rec, rec_no);
      break;
   case SQLM_EVENT_DEADLOCK:
//      print_sqlm_deadlock_event(fp, (sqlm_deadlock_event*) rec, rec_no);
      break;
   case SQLM_EVENT_DLCONN:
//      print_sqlm_dlconn_event(fp, (sqlm_dlconn_event*) rec, rec_no);
      break;
   case SQLM_EVENT_TABLESPACE:
//      print_sqlm_tablespace_event(fp, (sqlm_tablespace_event*) rec, rec_no);
      break;
   case SQLM_EVENT_DBHEADER:
//      print_sqlm_dbheader_event(fp, (sqlm_dbheader_event*) rec, rec_no);
      break;
   case SQLM_EVENT_CONNHEADER:
//      print_sqlm_connheader_event(fp, (sqlm_connheader_event*) rec, rec_no);
      break;
   case SQLM_EVENT_OVERFLOW:
//      print_sqlm_overflow_event(fp, (sqlm_overflow_event*) rec, rec_no);
      break;
   case SQLM_EVENT_START:
//      print_sqlm_evmon_start_event(fp,(sqlm_evmon_start_event*) rec, rec_no);
      break;
   default:
//      print_unknown_event(fp, rec, rec_no);
      break;
   } /* end switch on event type */
   fflush (fp);
} /* end of print_event_record */
```

Reading Events from a FILE Trace

This sample illustrates how to handle multiple files.

A file event monitor writes to files that are created in the directory identified by its target. When initially turned on, it starts writing to file: 00000000.evt, when this file is full (as specified by the MAXFILESIZE parameter on create event monitor), it moves on to file: 00000001.evt, and so on.

```
//------------------------------------------------------------------------------
// Build a fully qualified Event Monitor file name, given a file number
//------------------------------------------------------------------------------
#if _AIX
   #define PATH_SEP     '/'
   #define PATH_SEP_STR "/"
#else /* Assume Intel platform */
   #define PATH_SEP     '\\'
   #define PATH_SEP_STR "\\"
#endif
void build_event_monitor_file_name(const char* target, int fnum, char *fn) {

   // Build the full filename (path + filename)
   int len = strlen(target);
   if (target[len-1] == PATH_SEP) {
      sprintf(fn, "%s%0.8d.evt", target, fnum);
   } else {
      sprintf(fn, "%s%s%0.8d.evt", target, PATH_SEP_STR, fnum);
   } // end else need to append path delimiter to directory name
} /* end of build_event_monitor_file_name */
```

```
//------------------------------------------------------------------------------
// Read Events from files
//------------------------------------------------------------------------------
void read_events_from_file(char* target) {
   EventLog                evtlog;       /* Attributes for this event log     */
   int record_no;                          /* current record number */
   int file_no;                            /* current file number */
   int rc=0;
   int end_of_trace=0;                   /* True when no more files to read   */
   char buffer[4096];                    /* buffer for reading Event records  */

   //---------------------------------------------------------------------------
   // Initialize the attributes of this Event Log
   //---------------------------------------------------------------------------
   evtlog.type   = EVMFile;   // A File log
   evtlog.target = target;    // Directory where the files reside

   //---------------------------------------------------------------------------
   // Open the first file and read the log header
   //---------------------------------------------------------------------------
   file_no=0;                            /* First file to open is 00000000.evt */
   build_event_monitor_file_name(evtlog.target, file_no, evtlog.current_fn);
   read_log_header(&evtlog);
   print_sqlm_event_log_header(stdout, &evtlog.header);  // Print it

   //---------------------------------------------------------------------------
   // Read/print events from the trace file(s)
   //---------------------------------------------------------------------------
   record_no=0;                          /* Number of event records processed */
   while (!end_of_trace) {
      int event_type;                    /* Type of event read from the trace */
      rc=read_event_record(&evtlog, buffer, &event_type);
      if (rc == EOF) {

         /* Try to open the next trace file, if any */
         closeFile(evtlog.current_fp);
         build_event_monitor_file_name(evtlog.target,
                                       ++file_no, evtlog.current_fn);
         if ((evtlog.current_fp = openFile(evtlog.current_fn))==NULL)
            end_of_trace=true;      /* No more files to read */
         else {
            rc=read_event_record(&evtlog, buffer, &event_type);
            if (rc==EOF)
               end_of_trace = true;
         } /* else read the event from the next file */
      } else if (rc) end_of_trace = true;

      if (rc==0) {
         // Process the event
         print_event_record(stdout, event_type, buffer, ++record_no);
      } /* end if we got an event record */
   } /* end while there are more files in the Event Monitor trace */
} // end of read_events_from_file
```

See evm.c sample application in engn/samples/c_AIX for the complete source.

Appendix A. Database System Monitor Interfaces

This appendix contains reference material for the commands, SQL statements, and APIs associated with the database system monitor. The following tables list the commands or APIs to use for a given task.

Table 1. Snapshot Monitor Commands and APIs for a Given Task

Snapshot Monitoring Task	API	Command
Taking a snapshot	"sqlmonss - Get Snapshot API" on page 248 "sqlmonsz - Estimate Size Required for sqlmonss() Output Buffer API" on page 260	"GET SNAPSHOT Command" on page 232 "LIST ACTIVE DATABASES Command" on page 235 "LIST APPLICATIONS - Command" on page 237 "LIST DCS APPLICATIONS - Command" on page 239 "GET DATABASE MANAGER MONITOR SWITCHES Command" on page 228
Getting monitor switch status for an application or session	"sqlmon - Get/Update Monitor Switches API" on page 244	"GET MONITOR SWITCHES Command" on page 230
Updating monitor switch status for an application or session	"sqlmon - Get/Update Monitor Switches API" on page 244	"UPDATE MONITOR SWITCHES Command" on page 266
Getting monitor switch status for the database manager level	"sqlmonss - Get Snapshot API" on page 248	"GET DATABASE MANAGER MONITOR SWITCHES Command" on page 228
Resetting counters for an application or session	"sqlmrset - Reset Monitor API" on page 263	"RESET MONITOR Command" on page 241

Table 2. Event Monitor Commands (or SQL) for a Given Task

Event Monitor Task	Command
Creating an event monitor	"CREATE EVENT MONITOR Command and SQL" on page 214
Determining if an event monitor is active	"EVENT_MON_STATE SQL Function" on page 227
Activating/Deactivating an event monitor	"SET EVENT MONITOR STATE Command and SQL" on page 242
Deleting an event monitor	"DROP EVENT MONITOR Command and SQL" on page 226
Formatting trace to stdout	"db2evmon - Event Monitor Trace Formatter Command" on page 224
Reading trace with the GUI	"db2eva - Event Analyzer Command" on page 222

CREATE EVENT MONITOR Command and SQL

Purpose

Stores an Event Monitor definition in the database catalogs. When activated (see “SET EVENT MONITOR STATE Command and SQL” on page 242), an event monitor will log monitor data when certain events occur while using the database. You should read “Event Monitors” on page 10 and Chapter 4, “Event Monitor Output” on page 195 before using this command.

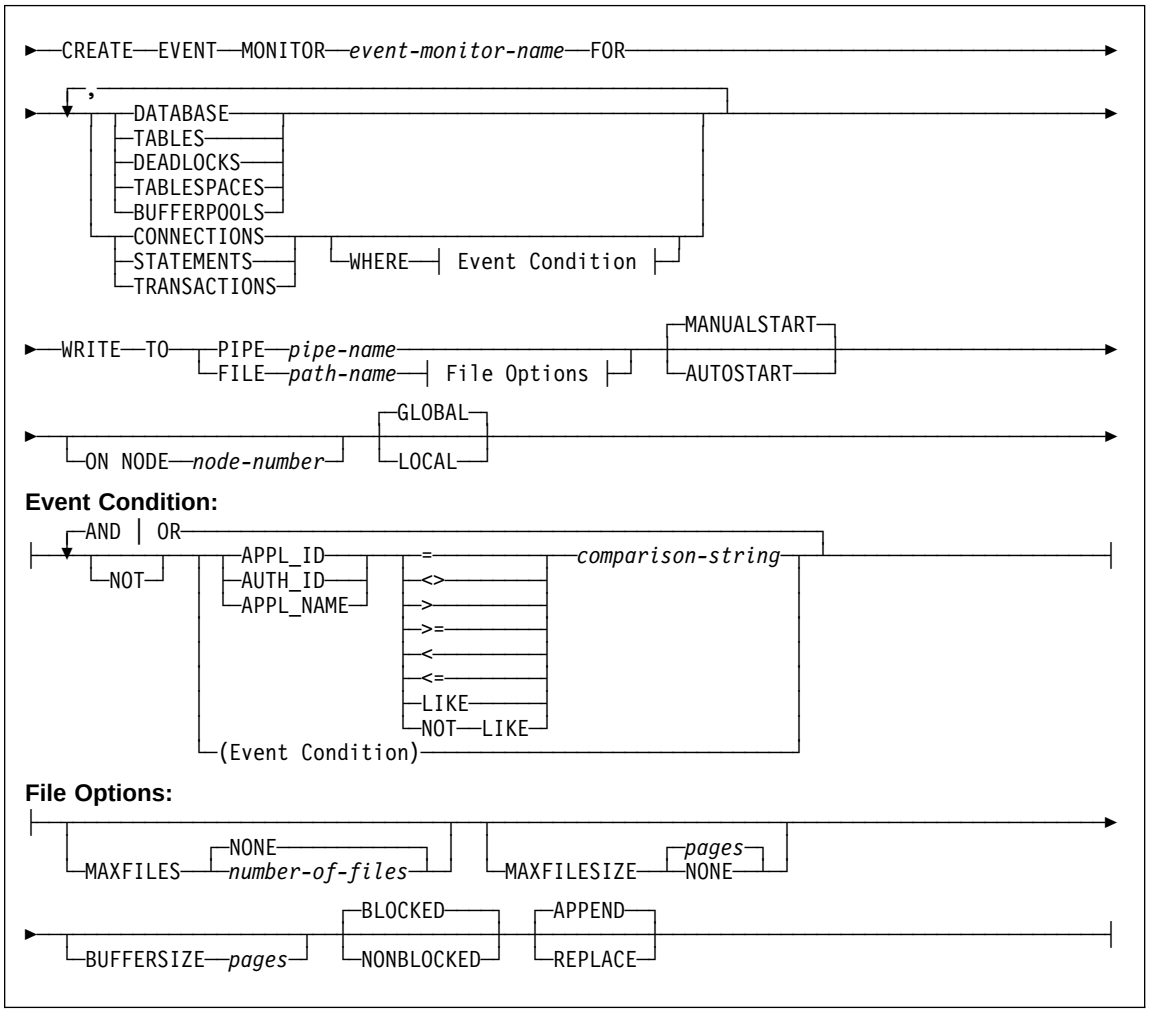
Context

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

Authorization

The privileges held by the authorization ID must include either SYSADM or DBADM authority (SQLSTATE 42502).

Format



Parameters

event-monitor-name
Names the event monitor. This is a one-part name. It is an SQL identifier (either ordinary or delimited). The *event-monitor-name* must not identify an event monitor that already exists in the catalog (SQLSTATE 42710).

FOR
Introduces the type of events to record. See “Event Types” on page 18 for a list of the records produced for each event type and the events that trigger writing them.

DATABASE
Specifies that the event monitor writes a database record when the database is deactivated.

TABLES
Specifies that the event monitor writes a table record for each table that has been accessed since database activation, when the database is deactivated. An active table is a table that has changed since the first connection to the database.

DEADLOCKS
Specifies that the event monitor writes a deadlock record whenever a deadlock occurs.

TABLESPACES
Specifies that the event monitor writes a table space record for each table space when the database is deactivated.

BUFFERPOOLS
Specifies that the event monitor writes a buffer pool record when the database is deactivated.

CONNECTIONS
Specifies that the event monitor writes a connection record when an application disconnects from the database.

STATEMENTS
Specifies that the event monitor writes a statement record whenever a SQL statement finishes executing.

TRANSACTIONS
Specifies that the event monitor writes a transaction record whenever a transaction completes (that is, whenever there is a commit or rollback operation).

WHERE *event condition*
Defines a filter that determines which will be monitored. If the result of the event condition is TRUE for a particular connection, then the event monitor will generate the requested event records for that connection.

This clause is a special form of the WHERE clause that should not be confused with a standard search condition.

If no WHERE clause is specified then all connections will be monitored.

APPL_ID
Specifies that the comparison string is an application ID of a connection.

AUTH_ID
Specifies that the comparison string is the authorization ID of a connection.

APPL_NAME
Specifies that the comparison string is the application program name of the connection.

comparison-string
A string to be compared with the APPL_ID, AUTH_ID, or APPL_NAME of each application that connects to the database. *comparison-string* must be

a string constant (that is, host variables and other string expressions are not permitted).

WRITE TO
Introduces the target for the data.

PIPE
Specifies that the target for the event monitor data is a named pipe. The event monitor writes the data to the pipe in a single stream (that is, as if it were a single, infinitely long file). When writing the data to a pipe, an event monitor does not perform blocked writes. If there is no room in the pipe buffer, then the event monitor will discard the data. It is the monitoring application's responsibility to read the data promptly if it wishes to ensure no data loss. See “Using Pipe Event Monitors” on page 18 for more details and examples.

pipe-name
The name of the pipe (FIFO on AIX) to which the event monitor will write the data.

The naming rules for pipes are platform specific. On UNIX operating systems pipe names are treated like file names. As a result, relative pipe names are permitted, and are treated like relative path-names (see *path-name* below). However, on OS/2, Windows 95 and Windows NT, there is a special syntax for a pipe name. As a result, on OS/2, Windows 95 and Windows NT absolute pipe names are required.

The existence of the pipe will not be checked at event monitor creation time. It is the responsibility of the user to have created and opened the pipe for reading at the time that the event monitor is activated. If the pipe is not available at this time, then the event monitor will turn itself off, and will log an error. (That is, if the event monitor was activated at database start time as a result of the AUTOSTART option, then the event monitor will log an error in the system error log.) If the event monitor is activated via the SET EVENT MONITOR STATE SQL statement, then that statement will fail (SQLSTATE 58030).

FILE
Indicates that the target for the event monitor data is a file (or set of files). The event monitor writes out the stream of data as a series of 8 character numbered files, with the extension “evt”. (for example, 00000000.evt, 00000001.evt, 00000002.evt, etc). The data should be considered to be one logical file even though the data is broken up into smaller pieces (that is, the start of the data stream is the first byte in the file 00000000.evt; the end of the data stream is the last byte in the file nnnnnnnn.evt).

The maximum size of each file can be defined as well as the maximum number of files. An event monitor will never split a single event record across two files. However, an event monitor may write related records in two different files. It is the responsibility of the application that uses this data to keep track of such related information when processing the event files. See “File Event Monitor Target” on page 202 for more information.

path-name
The name of the directory in which the event monitor should write the event files data. The path must be known at the server, however, the path itself could reside on another partition (or node) (for example, in a UNIX-based system, this might be an NFS mounted file). A string constant must be used when specifying the *path-name*.

The directory does not have to exist at CREATE EVENT MONITOR time. However, a check is made for the existence of the target path when the event monitor is activated. At that time, if the target path does not exist, an error (SQLSTATE 428A3) is raised.

If an absolute path (a path that starts with the root directory on AIX, or a disk identifier on OS/2, Windows 95 and Windows NT) is specified, then the specified path will be the one used. If a relative path (a path that does not start with the root) is specified, then the path relative to the DB2EVENT directory in the database directory will be used.

When a relative path is specified, the DB2EVENT directory is used to convert it into an absolute path. Thereafter, no distinction is made between absolute and relative paths. The absolute path is stored in the SYSCAT.EVENTMONITORS catalog view.

It is possible to specify two or more event monitors that have the same target path. However, once one of the event monitors has been activated for the first time, and as long as the target directory is not empty, it will be impossible to activate any of the other event monitors.

File Options
Specifies the options for the file format.

MAXFILES NONE
Specifies that there is no limit to the number of event files that the event monitor will create. This is the default.

MAXFILES *number-of-files*
Specifies that there is a limit on the number of event monitor files that will exist for a particular event monitor at any time. Whenever an event monitor has to create another file, it will check to make sure that the number of .evt files in the directory is less than *number-of-files*. If this limit has already been reached, then the event monitor will turn itself off.

If an application removes the event files from the directory after they have been written, then the total number of files that an event monitor can produce can exceed *number-of-files*. This option has been provided to allow a user to guarantee that the event data will not consume more than a specified amount of disk space.

MAXFILESIZE *pages*
Specifies that there is a limit to the size of each event monitor file. Whenever an event monitor writes a new event record to a file, it checks that the file will not grow to be greater than *pages* (in units of 4K pages). If the resulting file would be too large, then the event monitor switches to the next file. The default for this option is:

- OS/2, Windows 95 and Windows NT - 200 4K pages
- UNIX - 1000 4K pages

The number of pages must be greater than at least the size of the event buffer in pages. If this requirement is not met, then an error (SQLSTATE 428A4) is raised.

MAXFILESIZE NONE
Specifies that there is no set limit on a file's size. If MAXFILESIZE NONE is specified, then MAXFILES 1 must also be specified. This option means that one file will contain all of the event data for a particular event monitor. In this case the only event file will be 00000000.evt. This is the default.

BUFFERSIZE *pages*
Specifies the size of the event monitor buffers (in units of 4K pages). All FILE event monitor I/O is buffered to improve the performance of the event monitors. The larger the buffers, the less I/O will be performed by the event monitor. Highly active event monitors should have larger buffers than relatively inactive event monitors. When the monitor is started, two buffers of the specified size are allocated. Event monitors use double buffering to permit asynchronous I/O. See “File Event Monitor Buffering” on page 201 for more information.

The minimum and default size of each buffer (if this option is not specified) is 1 page (that is, 2 buffers, each 4 K in size). The maximum size of the buffers is limited by the size of the database manager monitor heap (MON_HEAP_SZ) since the buffers are allocated from this heap.

BLOCKED
Specifies that each agent that generates an event should wait for the event monitor to finish writing a buffer out to disk when both are full. BLOCKED should be selected to guarantee no event data loss. This is the default option. See “Blocked Event Monitors” on page 202 and “Non-Blocked Event Monitors” on page 202 for more information.

NONBLOCKED
Specifies that each agent that generates an event should not wait when the event monitor buffers are full. NONBLOCKED event monitors do not slow down database operations to the extent of

BLOCKED event monitors. However, NONBLOCKED event monitors are subject to data loss on highly active systems. See "Blocked Event Monitors" on page 202 and "Non-Blocked Event Monitors" on page 202 for more information.

APPEND
Specifies that if event data files already exist when the event monitor is turned on, then the event monitor will append the new event data to the existing stream of data files. When the event monitor is re-activated, it will resume writing to the event files as if it had never been turned off. APPEND is the default option. See "Restarting a File Event Monitor" on page 204 for more information.

The APPEND option does not apply at CREATE EVENT MONITOR time, if there is existing event data in the directory where the newly created event monitor is to write its event data.

REPLACE
Specifies that if event data files already exist when the event monitor is turned on, then the event monitor will erase all of the event files and start writing data to file 00000000.evt.

MANUALSTART
Specifies that the event monitor not be started automatically each time the database is started. Event monitors with the MANUALSTART option must be activated manually using the SET EVENT MONITOR STATE statement. This is the default option.

AUTOSTART
Specifies that the event monitor be started automatically each time the database is started.

ON NODE

node-number
Specifies a node number where the event monitor output thread or process runs. If defined as global, the event monitor output data, from all nodes, will be directed to that node. The default is the node where the command was issued.

LOCAL
Event monitor reports activity only for the node where it is running (the monitor node). This is the default.

GLOBAL
Event monitor reports activity from all nodes. In DB2 Version 5, for a partitioned database only deadlock event monitors can be defined as GLOBAL. The GLOBAL event monitor will report deadlocks involving applications running on any node in the system.

Usage

- Creating an event monitor **does not** activate it or create the target directory.

Comments

- Event monitor definitions are recorded in the SYSCAT.EVENTMONITORS catalog view. The events themselves are recorded in the SYSCAT.EVENTS catalog view.

Related Information

"SET EVENT MONITOR STATE Command and SQL" on page 242
"DROP EVENT MONITOR Command and SQL" on page 226

db2eva - Event Analyzer Command

Purpose

Starts the event analyzer GUI from the command line, allowing the user to view traces produced an event monitor.

Authorization

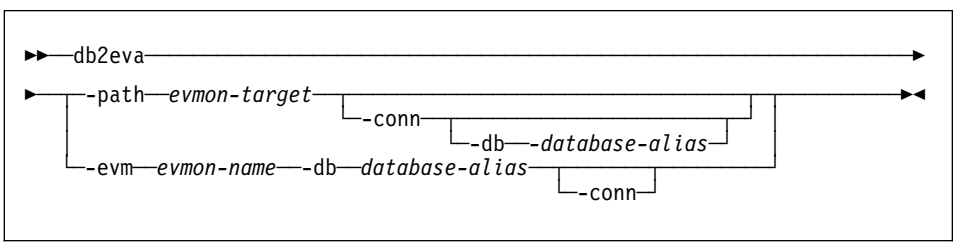
None, unless connecting to the database (`-evm` and `-db`), then one of the following is required:

sysadm
sysctrl
sysmaint
dbadm

Required Connection

None

Format



Parameters

Two methods of operation are provided for reading traces with db2eva, you can:

1. Specify the directory where the trace files reside (using -path) This mode allow users to move trace files from a server and analyze them locally, even if the Event Monitor has been dropped.
2. Specify the database and event monitor names; db2eva then automatically locates the trace files. When this mode is used, db2eva connects to the database, and issues a *select target from syscat.eventmonitors* to locate the directory where the Event Monitor writes the event records. The connection is then released, unless -conn is specified. This method cannot be used if the event monitor has been dropped.

-path *evmon-target*
Specifies the absolute path of the event monitor target, which can either be a directory or a named pipe.

-evm *evmon-name* **-db** *database-alias*

When -evm and -db are supplied, db2eva connects to the database, and obtains the directory or pipe to which the event monitor is writing, by issuing an SQL select from the syscat.eventmonitor catalog. For FILE event monitors, this means that you cannot move the trace files to a different directory. Specify the database for which the event monitor is defined, as catalogued on the machine where the trace is analyzed. Using database-alias overrides the database name specified in the trace.

-conn

Requests db2eva to maintain a connection to the database specified by the `-db` option. Or if `-db` is not supplied, then to the database specified in the trace file header. Keeping a connection allows the event analyzer to obtain information that is not contained in the trace files, for example the text for static SQL statements. (The statement text events for static SQL contain package creator, package number and section number. When the -conn option is specified, db2eva connects to the database and retrieves the text from the database system catalog using these fields). The default is not to keep a connection.

Comments

This tool is only available on OS/2 and Windows platforms.

It does not display database, table space, buffer pool, and table event records, but properly reads traces containing them.

db2evmon - Event Monitor Trace Formatter Command

Purpose

Formats the trace produced by file or pipe event monitors, and writes it to standard output. This tool is located in the `misc` subdirectory of the `sqllib` directory of the instance.

Authorization

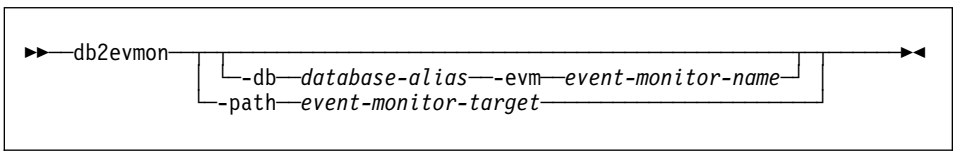
None, unless connecting to the database (`-evm` and `-db` are specified), then one of the following is required:

sysadm
sysctrl
sysmaint
dbadm

Required Connection

None

Format



Parameters

Two methods of operation are provided for formatting a trace:

1. Specify where the trace resides,using the `-path` option.
2. Let **db2evmon** locate the trace by selecting the event monitor target from the SYSCAT.EVENTMONITORS database catalog. You need to use `-evm` and `-db`.

-path *event-monitor-target*
Specifies the name of the directory containing the event monitor trace files, or the name of the pipe where the event monitor is writing its records.

-db *database-alias* **-evm** *event-monitor-name*
Specifies the database where the event monitor is defined, `event-monitor-name` one-part name of the event monitor. An ordinary or delimited SQL identifier.

Comments

If the data is being written to a pipe, the tool formats the output for display using standard output as event records are written occur. In this case, the tool must be started **before** the monitor is turned on.

See Also

"Programming to Read an Event Monitor Trace" on page 204

DROP EVENT MONITOR Command and SQL

Purpose

Removes an event monitor definition from the Database catalogs. Whenever an object is deleted, its description is deleted from the catalog and any packages that reference the object are invalidated.

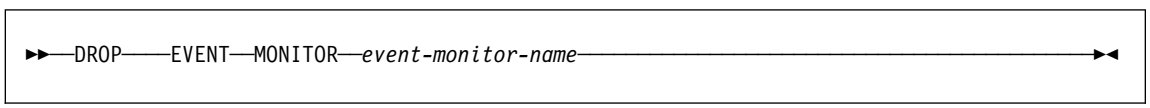
Context

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

Authorization

The authorization ID of the DROP statement when dropping an event monitor must have SYSADM or DBADM authority

Format



Parameters

EVENT MONITOR *event-monitor-name*

Identifies the event monitor that is to be dropped. The *event-monitor-name* must identify an event monitor that is described in the catalog (SQLSTATE 42704).

If the identified event monitor is ON, an error (SQLSTATE 55034) is raised. Otherwise, the event monitor is deleted.

If there are event files in the target path of the event monitor when the event monitor is dropped, the event files are not deleted.

Usage

An event monitor must be stopped or OFF before it can be deleted. Dropping an event monitor does not erase the target directory.

EVENT_MON_STATE SQL Function

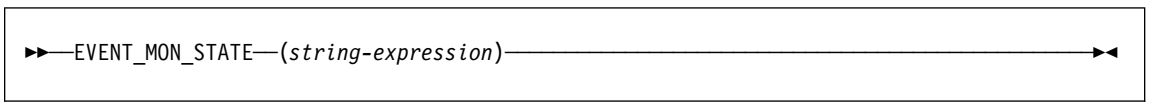
Purpose

The EVENT_MON_STATE function returns the current state of an event monitor.

Context

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

Format



Parameters

string-expression
The argument is a string expression with a resulting type of CHAR or VARCHAR and a value that is the name of an event monitor. If the named event monitor does not exist in the SYSCAT.EVENTMONITORS catalog table, SQLSTATE 42704 will be returned.

Usage

The schema is SYSIBM.

The result is an integer with one of the following values:

0 The event monitor is inactive.

1 The event monitor is active.

If the argument can be null, the result can be null; if the argument is null, the result is the null value.

Example:

- The following example selects all of the defined event monitors, and indicates whether each is active or inactive:

```
SELECT EVMONNAME,
   CASE
      WHEN EVENT_MON_STATE(EVMONNAME) = 0 THEN 'Inactive'
      WHEN EVENT_MON_STATE(EVMONNAME) = 1 THEN 'Active'
   END
FROM SYSCAT.EVENTMONITORS
```

GET DATABASE MANAGER MONITOR SWITCHES Command

Purpose

Displays the status of the database manager monitor switches. Monitor switches instruct the database system manager to collect statistics about its operation and performance and that of the applications using it. A database manager-level switch is on when any monitoring application has turned it on, an active event monitor requires this switch to be on, or it has been set in the database manager configuration. This command is used to determine if the database manager is currently collecting data.

Authorization

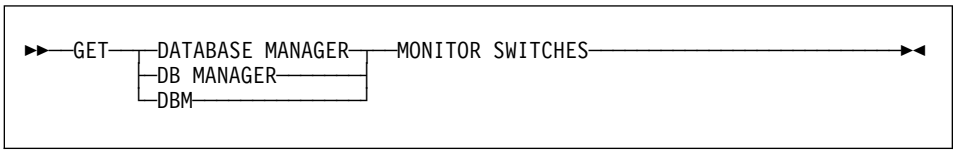
One of the following:

sysadm
sysctrl
sysmaint

Required Connection

Instance. To display the settings for a remote instance, or for a different local instance, it is necessary to first attach to that instance.

Format



Parameters

None

Example

The following is sample output from GET DATABASE MANAGER MONITOR SWITCHES:

```
            DBM System Monitor Information Collected

Buffer Pool Activity Information (BUFFERPOOL) = ON   06-11-1997 10:11:01.738377
Lock Information                         (LOCK) = OFF
Sorting Information                      (SORT) = ON   06-11-1997 10:11:01.738400
SQL Statement Information           (STATEMENT) = OFF
Table Activity Information              (TABLE) = OFF
Unit of Work Information                  (UOW) = ON   06-11-1997 10:11:01.738353
```

Comments

This command returns the settings for the database manager, indicating whether the database manager is currently collecting monitor data. To see the switch settings for your session issue the GET MONITOR SWITCHES command.

Default switch settings can be set in the database manager configuration file. If switches are set ON in the database manager configuration file, then the database manager will always be collecting monitor data, even if all monitoring applications have turned off their switches.

See Also

"GET MONITOR SWITCHES Command" on page 230
"GET SNAPSHOT Command" on page 232
"RESET MONITOR Command" on page 241
"UPDATE MONITOR SWITCHES Command" on page 266
"sqlmonss - Get Snapshot API" on page 248.

GET MONITOR SWITCHES Command

Purpose

Displays the status of the database system monitor switches for the current session. Monitor switches instruct the database system manager to collect database activity information. Each application using the database system monitor interface has its own set of monitor switches. This command displays them. To display the database manager-level switches, use “GET DATABASE MANAGER MONITOR SWITCHES Command” on page 228. If a particular switch is on, this command also displays the time stamp for when the switch was turned on.

Authorization

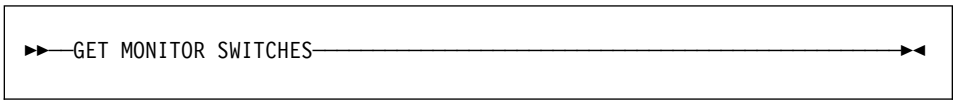
One of the following:

sysadm
sysctrl
sysmaint

Required Connection

Instance.

Format



Parameters

None

Example

The following is sample output from GET MONITOR SWITCHES:

```
            Monitor Recording Switches

Buffer Pool Activity Information  (BUFFERPOOL) = ON  02-20-1997 16:04:30.070073
Lock Information                         (LOCK) = OFF
Sorting Information                      (SORT) = OFF
SQL Statement Information           (STATEMENT) = ON  02-20-1997 16:04:30.070073
Table Activity Information              (TABLE) = OFF
Unit of Work Information                  (UOW) = ON  02-20-1997 16:04:30.070073
```

Comments

When a database system monitor command is first issued the session inherits the switch settings of the database manager configuration. The settings can be overridden using “UPDATE MONITOR SWITCHES Command” on page 266.

See Also

“GET DATABASE MANAGER MONITOR SWITCHES Command” on page 228
“GET SNAPSHOT Command” on page 232
“RESET MONITOR Command” on page 241
“UPDATE MONITOR SWITCHES Command” on page 266
“sqlmon - Get/Update Monitor Switches API” on page 244.

GET SNAPSHOT Command

Purpose

Collects some of the data that the database manager maintains about its operation and performance. The information returned represents a snapshot of this data at the time the command is issued.

Scope

Returns data only for the node to which the session is attached.

Authorization

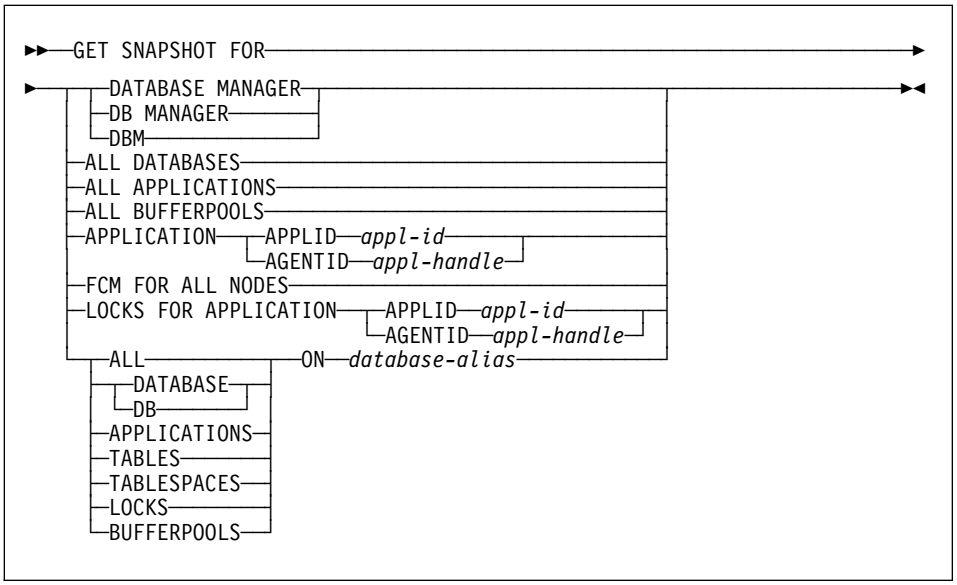
One of the following:

sysadm
sysctrl
sysmaint

Required Connection

Instance. To obtain a snapshot of a remote instance, it is necessary to first attach to that instance.

Format



Parameters

Parameters are group by the way you would use them.

Note: The appl-id and appl-handle can be obtained by issuing a LIST APPLICATIONS command (see "LIST APPLICATIONS - Command" on page 237).

DATABASE MANAGER
Database manager level information, including internal monitor switch settings. On multi-node systems FCM information is also returned.

FCM FOR ALL NODES
Provides Fast Communication Manager (FCM) statistics with for this node with respect to all other nodes.

DATABASE ON *database-alias*
Database level information and counters for a database. Information is returned only if there is at least one application connected to the database.

ALL DATABASES
Same information for each database active on the node.

ALL ON *database-alias*
For the specified database, returns: database snapshot, lock snapshot, buffer pool snapshot, and application snapshot(for each connection to the database).

APPLICATION APPLID *appl-id*
Application level information, includes cumulative counters, status information. If the statement switch is ON, it also returns statistics about each cursor currently open and the most recent SQL statement executed.

APPLICATION AGENTID *appl-handle*
Same as APPLICATION APPLID.

APPLICATIONS ON *database-alias*
Same information as APPLICATION APPLID, for each application that is connected to the database on this node.

ALL APPLICATIONS
Same information as APPLICATION APPLID, for each application that is connected to a database on the current node.

TABLES ON *database-alias*
Returns Table activity information at the database and application level for each application connected to the database, and at the table level for each table that *was accessed* by an application connected to the database. Requires table switch to be ON.

LOCKS FOR APPLICATION APPLID *appl-id*
Provides information about each lock held by the application. Provides lock wait information, if application is waiting for a lock. Requires lock switch to be ON.

LOCKS FOR APPLICATION AGENTID *appl-handle*
Same information as LOCKS FOR APPLICATION APPLID.

LOCKS ON *database-alias*
Same information as LOCKS FOR APPLICATION APPLID for each application connected to the mentioned database. Plus a database level summary. Requires lock switch

TABLESPACES ON *database-alias*
Information about tablespace activity at the database level; the application level for each application connected to the database; and the tablespace level for each tablespace that has been accessed by an application connected to the database. Requires buffer pool switch.

ALL BUFFERPOOLS
Provides buffer pool activity counters. Requires buffer pool switch to be ON.

BUFFERPOOLS ON *database-alias*
Same information as ALL BUFFERPOOLS, but only for the specified database.

Comments

INSTANCE CONNECTION: If not attached to an instance, issuing this command will automatically attach your session to the instance specified by the DB2INSTANCE environment variable.

To obtain a snapshot from a remote instance (or a different local instance), it is necessary to first attach to that instance. If an alias for a database residing at a different instance is specified, an error message is returned.

DATA COLLECTED UNDER SWITCH CONTROL: Data elements that are collected by the DBMS only if a monitor switch is ON are either not returned, or returned as 'Not Collected'. Check Chapter 3, “Database System Monitor Data Elements” on page 31 to determine if a switch needs to be turned on for a data element.

See Also

“GET MONITOR SWITCHES Command” on page 230
“RESET MONITOR Command” on page 241
“UPDATE MONITOR SWITCHES Command” on page 266
“LIST ACTIVE DATABASES Command” on page 235
“LIST APPLICATIONS - Command” on page 237
“LIST DCS APPLICATIONS - Command” on page 239
“sqlmonss - Get Snapshot API” on page 248

LIST ACTIVE DATABASES Command

Purpose

Displays the list of databases that are active on this instance. This is a subset of the information listed by the GET SNAPSHOT FOR ALL DATABASES command (see "GET SNAPSHOT Command" on page 232). For each active database, this command displays the following:

- Database name
- Number of applications currently connected to the database
- Database path.

Scope

Returns data only for the node to which the session is attached.

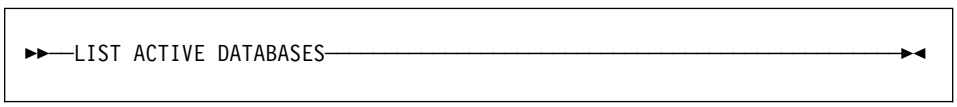
Authorization

None

Required Connection

Instance. To obtain a snapshot of a remote instance, it is necessary to first attach to that instance.

Format



Parameters

None

Examples

Following is sample output from the LIST ACTIVE DATABASES command:

```
                        Active Databases

Database name                              = TEST
Applications connected currently           = 0
Database path                              = /home/smith/smith/NODE0000/SQL00002/

Database name                              = SAMPLE
Applications connected currently           = 1
Database path                              = /home/smith/smith/NODE0000/SQL00001/
```

LIST ACTIVE DATABASES

See Also

"GET SNAPSHOT Command" on page 232.
"LIST APPLICATIONS - Command" on page 237
"sqlmonss - Get Snapshot API" on page 248

LIST APPLICATIONS - Command

Purpose

Displays the list of applications connected to a database on the instance, including secondary connections established to access a partitioned database.

Scope

This command only returns information for the node on which it is issued.

Authorization

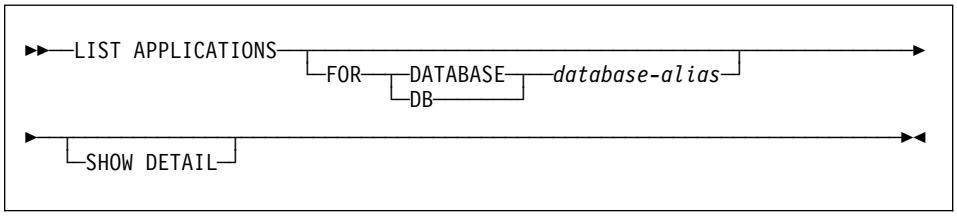
One of the following:

sysadm
sysctrl
sysmaint

Required Connection

Instance. To list applications for a remote instance, it is necessary to first attach to that instance.

Format



Parameters

FOR DATABASE *database-alias*

Information for each application that is connected to the specified database is to be displayed. Database name information is not displayed. If this option is not specified, the command displays the information for each application that is currently connected to any database at the node to which the user is currently attached.

The default application information is comprised of the following:

- Authorization ID
- Application program name
- Application handle
- Application ID
- Database name.

SHOW DETAIL

Output will include the following additional information:

- Sequence #
- Application status
- Status change time
- Database path.

Note: If this option is specified, it is recommended that the output be redirected to a file, and that the report be viewed with the help of an editor. The output lines may wrap around when displayed on the screen.

Example

The following is sample output from LIST APPLICATIONS:

```
Auth Id  Application    Appl.      Application Id                 DB       # of
         Name           Handle                                    Name    Agents
-------- -------------- ---------- ------------------------------ -------- -----
smith    db2bp_32       12         *LOCAL.smith.970220191502      TEST     1
smith    db2bp_32       11         *LOCAL.smith.970220191453      SAMPLE   1
```

Note: For more information about these fields, see Chapter 3, “Database System Monitor Data Elements” on page 31.

Comments

The database administrator can use the output from this command as an aid to problem determination. In addition, this information is required if the database administrator wants to use “GET SNAPSHOT Command” on page 232 or FORCE an application.

To list applications at a remote instance (or a different local instance), it is necessary to first attach to that instance. If FOR DATABASE is specified when an attachment exists, and the database resides at an instance which differs from the current attachment, the command will fail.

See Also

“sqlmonss - Get Snapshot API” on page 248
“GET SNAPSHOT Command” on page 232

LIST DCS APPLICATIONS - Command

Purpose

Displays the contents of the Database Connection Services (DCS) directory to standard output.

Authorization

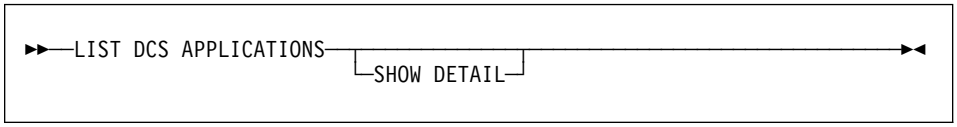
One of the following:

sysadm
sysctrl
sysmaint

Required Connection

Instance. To list the DCS applications at a remote instance, it is necessary to first attach to that instance.

Format



Parameters

LIST DCS APPLICATIONS

The default application information includes:

- Host authorization ID (*username*)
- Application program name
- Agent ID
- Outbound application ID (*luwid*).

SHOW DETAIL

Specifies that output include the following additional information:

- Application ID
- Application sequence number
- Client database alias
- Client node name (*nname*)
- Client product ID
- Code page ID
- Code page
- Outbound sequence number
- Host database name
- Host product ID.

Example

The following is sample output from LIST DCS APPLICATIONS:

```
Auth Id  Application Name     Agent Id   Outbound Application Id
-------- -------------------- ---------- --------------------------------

DDCSUS1  db2bp                89330      CAIBMOML.OMXT4H08.A79EAA3C6E29
```

Note: For more information about these fields, see Chapter 3, “Database System Monitor Data Elements” on page 31.

Comments

The database administrator can use this command to match client application connections *to* the gateway with corresponding host connections *from* the gateway.

The database administrator can also use agent ID information to force specified applications off a DDCS server.

RESET MONITOR Command

Purpose

Resets the counters maintained by the database system monitor for the current session for specified database, or for all active databases, to zero.

Authorization

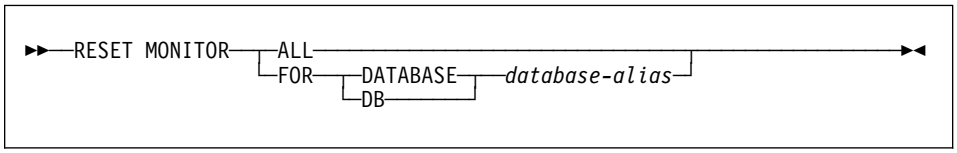
One of the following:

sysadm
sysctrl
sysmaint

Required Connection

Instance.

Format



Parameters

ALL

This option indicates that the internal counters should be reset for all databases. Some database manager information is also reset.

FOR DATABASE *database-alias*

This option indicates that only the counters for the database with alias *database-alias* should be reset.

Comments

Each session (instance) has its own private view of the monitor data. If one user resets, other users are not affected.

This resets the data for all monitor switches. To reset data for a single switch turn it OFF, then ON, using “UPDATE MONITOR SWITCHES Command” on page 266.

See Also

“GET SNAPSHOT Command” on page 232
“GET MONITOR SWITCHES Command” on page 230
“UPDATE MONITOR SWITCHES Command” on page 266
“sqlmrset - Reset Monitor API” on page 263

SET EVENT MONITOR STATE Command and SQL

Purpose

The SET EVENT MONITOR STATE statement activates or deactivates an event monitor. The SET EVENT MONITOR STATE statement is not under transaction control.

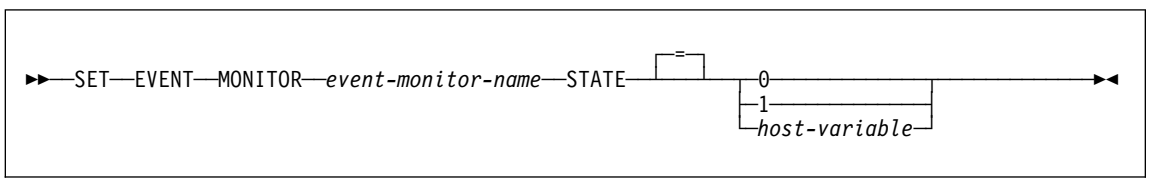
Context

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared.

Authorization

The authorization ID of the statement most hold either SYSADM or DBADM authority (SQLSTATE 42815).

Format



Parameters

event-monitor-name
Identifies the event monitor to activate or deactivate. The name must identify an event monitor that exists in the catalog (SQLSTATE 42704).

new-state
new-state can be specified either as an integer constant or as the name of a host variable that will contain the appropriate value at run time. The following may be specified:

0	Indicates that the specified event monitor should be deactivated.
1	Indicates that the specified event monitor should be activated. The event monitor should not already be active; otherwise a warning (SQLSTATE 01598) is issued.
host-variable	The data type is INTEGER. The value specified must be 0 or 1 (SQLSTATE 42815). If *host-variable* has an associated indicator variable, the value of that indicator variable must not indicate a null value (SQLSTATE 42815).

Usage

- Although an unlimited number of event monitors may be defined, there is a limit of 32 event monitors that can be simultaneously active (SQLSTATE 54030).
- In order to activate an event monitor, the transaction in which the event monitor was created must have been committed (SQLSTATE 55033). This rule prevents (in one unit of work) creating an event monitor, activating the monitor, then rolling back the transaction.
- If the number or size of the event monitor files exceeds the values specified for MAXFILES or MAXFILESIZE on the CREATE EVENT MONITOR statement, an error (SQLSTATE 54031) is raised.
- If the target path of the event monitor (that was specified on the CREATE EVENT MONITOR statement) is already in use by another event monitor, an error (SQLSTATE 51026) is raised.

Comments

- Activating an event monitor performs a reset of any counters associated with it.

The following example activates an event monitor called SMITHPAY.

```
SET EVENT MONITOR SMITHPAY STATE = 1
```

The current state of an event monitor (active or inactive) is determined by using the EVENT_MON_STATE built-in function.

sqlmon - Get/Update Monitor Switches API

Purpose

Selectively turns on or off switches for groups of monitor data to be collected by the database manager. Returns the current state of these switches for the application issuing the call.

To get the current state of the switches at the database manager level use “sqlmonss - Get Snapshot API” on page 248.

Scope

This API only affects the application making the call.

Authorization

One of the following:

sysadm
sysctrl
sysmaint

Required Connection

Instance.

API Include File

sqlmon.h

C API Syntax

```
/* File: sqlmon.h */
/* API: Get/Update Monitor Switches */
/* ... */
int SQL_API_FN
  sqlmon (
    unsigned long         version,
    _SQLOLDCHAR           *reserved,
    sqlm_recording_group  group_states[],
    struct sqlca          *sqlca);
/* ... */
```

API Parameters

sqlca

Output. A pointer to the *sqlca* structure, that returns error information.

group_states

Input/Output. Specifies the monitor switches to update and returns their values. It is an array of sqlm_recording_group, one for each monitor switch:

input_state — To request the setting for a switch use SQLM_HOLD. To turn a switch OFF use SQLM_OFF. To turn a switch ON use SQLM_ON.

output_state — The current setting for the switch, either SQLM_OFF or SQLM_ON.

start_time — A timestamp indicating the time a switch was turned ON. A value of 0 (zero) is returned if the switch is OFF.

reserved

Reserved for future use. Users should set this value to NULL.

version

Input. Version ID of the database monitor data to collect. The database monitor only returns data that was available for the requested version. Set this parameter to one of the following symbolic constants:

- `SQLM_DBMON_VERSION1`
- `SQLM_DBMON_VERSION2`
- `SQLM_DBMON_VERSION5`

If requesting data for a version higher than the current server, the database monitor only returns data for its level.

Note: If `SQLM_DBMON_VERSION1` is specified as the version, the APIs cannot be run remotely.

Comments

To obtain the status of the switches at the database manager level, call "sqlmonss - Get Snapshot API" on page 248, specifying `SQMA_DB2` for *OBJ_TYPE* (get snapshot for database manager).

See Also

"sqlmonss - Get Snapshot API" on page 248
"sqlmonsz - Estimate Size Required for sqlmonss() Output Buffer API" on page 260
"sqlmrset - Reset Monitor API" on page 263
"UPDATE MONITOR SWITCHES Command" on page 266
"GET MONITOR SWITCHES Command" on page 230

Code Sample

The following example illustrates how to update the monitor switches and print their current settings.

```
/*
     Database Monitor Switch API
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "sqlca.h"
#include "sqlutil.h"             // for using sqlaintp
#include "sqlmon.h"
//------------------------------------------------------------------------------
// Database Monitor Switch API Sample
//------------------------------------------------------------------------------
char* sw_status_string(int val) {
    switch (val) {
    case SQLM_OFF: return "OFF";
    case SQLM_ON:  return "ON";
    }
    return "";
}
void print_sws(sqlm_recording_group group_states[SQLM_NUM_GROUPS]);
void print_sw_set_times(sqlm_recording_group group_states[SQLM_NUM_GROUPS]);
int main() {
    //--------------------------------------------------------------------
    // Set Table switch ON, UOW switch OFF, and query the current (default)
    // values for the other switches.  (see Database Manager Configuration)
    //--------------------------------------------------------------------
    sqlm_recording_group group_states[SQLM_NUM_GROUPS];
    struct sqlca sqlca;
    group_states[SQLM_TABLE_SW].input_state =       SQLM_ON;
    group_states[SQLM_UOW_SW].input_state =         SQLM_OFF;
    group_states[SQLM_STATEMENT_SW].input_state =   SQLM_HOLD;
    group_states[SQLM_BUFFER_POOL_SW].input_state = SQLM_HOLD;
    group_states[SQLM_LOCK_SW].input_state =        SQLM_HOLD;
    group_states[SQLM_SORT_SW].input_state =        SQLM_HOLD;
    //--------------------------------------------------------------------
    // Perform the call
    //--------------------------------------------------------------------
    sqlmon(SQLM_DBMON_VERSION5, NULL, group_states, &sqlca);
    if (sqlca.sqlcode<0) {
        // get and display a printable error message
        char msg[1024];
        sqlaintp (msg, sizeof(msg), 60, &sqlca);
        printf("%s", msg);
    }
    //--------------------------------------------------------------------
    // Print the output
    //--------------------------------------------------------------------
    print_sws(group_states);          // Print the switch values
    print_sw_set_times(group_states); // Print their switch set time (if ON)
} // end of Database Monitor Switch API sample
//------------------------------------------------------------------------------
// print switch values
//------------------------------------------------------------------------------
void print_sws(sqlm_recording_group group_states[SQLM_NUM_GROUPS]) {
    printf("SQLM_UOW_SW:         %s\n",
         sw_status_string(group_states[SQLM_UOW_SW].output_state));
    printf("SQLM_STATEMENT_SW:   %s\n",
         sw_status_string(group_states[SQLM_STATEMENT_SW].output_state));
    printf("SQLM_TABLE_SW:       %s\n",
         sw_status_string(group_states[SQLM_TABLE_SW].output_state));
    printf("SQLM_BUFFER_POOL_SW: %s\n",
         sw_status_string(group_states[SQLM_BUFFER_POOL_SW].output_state));
    printf("SQLM_LOCK_SW:        %s\n",
         sw_status_string(group_states[SQLM_LOCK_SW].output_state));
    printf("SQLM_SORT_SW:        %s\n",
```

```
        sw_status_string(group_states[SQLM_SORT_SW].output_state));
} // end print_sws
//------------------------------------------------------------------------------
// print switch set times (if ON)
//------------------------------------------------------------------------------
void print_sw_set_times(
          sqlm_recording_group group_states[SQLM_NUM_GROUPS]) {
    if (group_states[SQLM_UOW_SW].start_time.seconds) {
        printf("SQLM_UOW_SW start_time:         %s\n", ctime((time_t *)
        &group_states[SQLM_UOW_SW].start_time.seconds));
    }
    if (group_states[SQLM_STATEMENT_SW].start_time.seconds) {
        printf("SQLM_STATEMENT_SW start_time:   %s\n", ctime((time_t *)
        &group_states[SQLM_STATEMENT_SW].start_time.seconds));
    }
    if (group_states[SQLM_TABLE_SW].start_time.seconds) {
        printf("SQLM_TABLE_SW start_time:       %s\n", ctime((time_t *)
        &group_states[SQLM_TABLE_SW].start_time.seconds));
    }
    if (group_states[SQLM_BUFFER_POOL_SW].start_time.seconds) {
        printf("SQLM_BUFFER_POOL_SW start_time: %s\n", ctime((time_t *)
        &group_states[SQLM_BUFFER_POOL_SW].start_time.seconds));
    }
    if (group_states[SQLM_LOCK_SW].start_time.seconds) {
        printf("SQLM_LOCK_SW start_time:        %s\n", ctime((time_t *)
        &group_states[SQLM_LOCK_SW].start_time.seconds));
    }
    if (group_states[SQLM_SORT_SW].start_time.seconds) {
        printf("SQLM_SORT_SW start_time:        %s\n", ctime((time_t *)
        &group_states[SQLM_SORT_SW].start_time.seconds));
    }
} // end print_sw_set_times (if ON)
```

sqlmonss - Get Snapshot API

Purpose

Collects database manager monitor information and returns it to a user-allocated data buffer. The information returned represents a *snapshot* of the database manager operational status at the time the API was called.

Scope

This API returns information only for the instance.

Authorization

One of the following:

sysadm
sysctrl
sysmaint

Required Connection

Instance. To obtain a snapshot from a remote instance (or a different local instance), it is necessary to first attach to that instance.

API Include File

sqlmon.h

C API Syntax

```
/* File: sqlmon.h */
/* API: Get Snapshot */
/* ... */
int SQL_API_FN
  sqlmonss (
    unsigned long   version,
    _SQLOLDCHAR     *reserved,
    sqlma           *sqlma_ptr,
    unsigned long   buffer_length,
    void            *buffer_area,
    sqlm_collected  *collected,
    struct sqlca    *sqlca);
/* ... */
```

API Parameters

version

Input. Version ID of the database monitor data to collect. The database monitor only returns data that was available for the requested version. Set this parameter to one of the following symbolic constants:

- SQLM_DBMON_VERSION1
- SQLM_DBMON_VERSION2
- SQLM_DBMON_VERSION5

If requesting data for a version higher than the current server, the database monitor only returns data for its level.

Note: If SQLM_DBMON_VERSION1 is specified as the version, the APIs cannot be run remotely.

reserved

Input. Reserved for future use. Must be set to NULL.

sqlma_ptr

Input. Pointer to the user-allocated *sqlma* (monitor area) structure. This structure specifies the **snapshot requests** to be returned by this invocation of sqlmonss().

buffer_length

Input. The length of the data buffer. You may want to first call sqlmonsz() to estimate which size would be required for a given sqlmonss() invocation. However, if you will be issuing frequent sqlmonss() calls, and especially if peak activity is predictable on your system, then you will get better performance by allocating a fixed size buffer in your application.

buffer_area

Output. Pointer to the user-defined data buffer into which the snapshot information will be returned.

collected

Output. A pointer to the *sqlm_collected* structure which provides information about the server and the number of top-level structures returned in the output buffer area.

sqlca

Output. A pointer to the *sqlca* structure where error information is returned. For more information see the *API Reference*.

Snapshot Requests Supported

The following table lists the Snapshot request types that are supported.

API request type	Data structures that may be returned (Record type)	Information returned
SQLMA_APPLINFO_ALL	sqlm_applinfo (SQLM_APPLINFO_SS)	Application identification information for all applications currently connected to a database that is managed by the DB2 instance on the node where snapshot is taken.

API request type	Data structures that may be returned (Record type)	Information returned
SQLMA_DBASE_APPLINFO	sqlm_applinfo (SQLM_APPLINFO_SS)	Same as SQLMA_APPLINFO_ALL for each application currently connected to the specified database.
SQLMA_DCS_APPLINFO_ALL	sqlm_dcs_applinfo (SQLM_DCS_APPLINFO_SS)	Application identification info for all DCS applications currently connected to a database that is managed by the DB2 instance on the node where snapshot is taken.
SQLMA_DB2	sqlm_db2 (SQLM_DB2_SS)	Database manager level information, including internal monitor switch settings.
	sqlm_fcm (SQLM_FCM_SS) sqlm_fcm_node (SQLM_NODE_SS)	Fast communication manager information for each node in a partitioned database that this node is communicating with.
SQLMA_DBASE	sqlm_dbase (SQLM_DBASE_SS)	Database level information and counters for a database. Information is returned only if there is at least one application connected to the database..
	sqlm_rollfwd_info (SQLM_DBASE_ROLLFWD_SS) sqlm_rollfwd_ts_info (SQLM_DBASE_ROLLFWD_TS_SS)	Rollforward information if a rollforward is in progress.
SQLMA_DBASE_ALL		Same as SQLMA_DBASE for each database active on the node.
SQLMA_APPL	sqlm_appl (SQLM_APPL_SS) sqlm_lock_wait (SQLM_LOCK_WAIT_SS)	Application level information, includes cumulative counters, status information. Lock information for every agent working for this application that is waiting for a lock.
	sqlm_stmt (SQLM_STMT_SS)	Statement information for each open cursor and the last statement executed.
	sqlm_subsection (SQLM_SUBSECTION_SS)	Subsection information that immediately follows its parent sqlm_stmt.
	sqlm_subagent (SQLM_SUBAGENT_SS)	In partitioned databases, an sqlm_subagent always follows its parent sqlm_subsection or sqlm_stmt. An sqlm_subagent is not returned for a coordinator agent.
SQLMA_AGENT_ID		Same as SQLMA_APPL.
SQLMA_DBASE_APPLS		Same as SQLMA_APPL, for each application that is connected to the database on the node.
SQLMA_APPL_ALL		Same as SQLMA_APPL. for each application that is active on the node.

API request type	Data structures that may be returned (Record type)	Information returned
SQLMA_DBASE_TABLES	sqlm_table_header (SQLM_TABLE_HEADER_SS) sqlm_table (SQLM_TABLE_SS)	Table activity information for each table that was accessed, sqlm_table_header followed by an sqlm_table for each table.
SQLMA_APPL_LOCKS	sqlm_appl_lock (SQLM_APPL_LOCK_SS) sqlm_lock_wait (SQLM_LOCK_WAIT_SS) sqlm_lock (SQLM_LOCK_SS)	List of locks held by the application, and any lock wait information, sqlm_appl_lock is followed by sqlm_lock_wait if an application is in a lock wait, followed by an sqlm_lock for each lock held.
SQLMA_APPL_LOCKS_AGENT_ID		Same as SQLMA_APPL_LOCKS.
SQLMA_DBASE_LOCKS	sqlm_dbase_lock (SQLM_DBASE_LOCK_SS) (SQLMA_APPL_LOCKS)	Lock information at the database level, and application level for each application connected to the database, followed by an SQLMA_APPL_LOCKS for each application.
SQLMA_DBASE_TABLESPACES	sqlm_tablespace_header (SQLM_TABLESPACE_HEADER_SS) sqlm_tablespace (SQLM_TABLESPACE_SS)	Information about table space activity the database level, the application level for each application connected to the database, and the table space level for each table space that has been accessed by an application connected to the database, sqlm_tablespace_header followed by an sqlm_tablespace for each table space.
SQLMA_BUFFERPOOLS_ALL	sqlm_bufferpool (SQLM_BUFFERPOOL_SS)	Bufferpool activity counters.
SQLMA_DBASE_BUFFERPOOLS		Same as SQLMA_BUFFERPOOLS_ALL, but for specified database only.

Specifying the Snapshot Requests

An invocation of sqlmonss() can specify several requests.

The *sqlma* supplied as input argument to sqlmonss() contains an array of **sqlm_obj_struct**. Each sqlm_obj_struct is an individual snapshot request.

sqlm_obj_struct is defined as follows:

```
typedef struct sqlm_obj_struct          /* SNAPSHOT REQUEST  */
{
   unsigned long agent_id;              /* used for requests based on agentid */
   unsigned long obj_type;              /* Snapshot Request Type (SQLMA_XXXX) */
   char          object[SQLM_OBJECT_SZ];/* used for requests based on object  */
                                        /* name, such as 'get snapshot for database' */
}sqlm_obj_struct;
```

Where *agent_id* and *object* are only required if applicable for the request type, and are mutually exclusive. For example: a database name is required when the type is SQLMA_DBASE_LOCKS (get snapshot for locks on database), whereas an agent_id is required when the type is SQLMA_APPL_LOCKS_AGENT_ID. Both agent_id and object are ignored for requests such as SQLMA_APPLINFO_ALL (list applications).

Note that agent_id is the **application handle** for an application. It does not correspond to any Operating System process Id (it is named this way for source compatibility with older releases of DB2).

Setting up the sqlma and issuing the snapshot call

The following example sets up the sqlma for a call to sqlmonss() that requests two different snapshots. The first request requires an object name, the database alias, the second request requires an agent_id, the application handle:

```
#include "string.h"
#include "stdlib.h"
#include "stdio.h"
#include "sqlutil.h"
#include "sqlmon.h"  // System Monitor interface
  main() {
      struct sqlca sqlca;
      int rc;

      #define BUFFER_SZ 4096          // Use a fixed size output buffer
      char snap_buffer[BUFFER_SZ];    // Snapshot output buffer
      sqlm_collected collected;

      //-------------------------------------------------------------------------
      // Request SQLMA_DBASE, and SQLMA_APPL_LOCKS_AGENT_ID in the sqlma
      //-------------------------------------------------------------------------
      unsigned long agent_id = 0; // STUB: Obtain by issuing 'list application'

      // Allocate the variable size sqlma structure
      struct sqlma* sqlma = (struct sqlma *) malloc(SQLMASIZE(2));

      // Request 2 different snapshots in same call
      sqlma->obj_num = 2;
      sqlma->obj_var[0].obj_type      = SQLMA_DBASE;
      strcpy(sqlma->obj_var[0].object, "SAMPLE");

      sqlma->obj_var[1].obj_type      = SQLMA_APPL_LOCKS_AGENT_ID;
      sqlma->obj_var[1].agent_id      = agent_id;

      //-------------------------------------------------------------------------
      // Perform the snapshot
      //-------------------------------------------------------------------------
      rc = sqlmonss(SQLM_DBMON_VERSION5, NULL, sqlma,
                    BUFFER_SZ, snap_buffer,
                    &collected,
                    &sqlca);
      if (sqlca.sqlcode < 0) {
          // get and display a printable error message
          char msg[1024];
          sqlaintp (msg, sizeof(msg), 60, &sqlca);
          printf("%s", msg);
      }
      free(sqlma);
      return rc;
  }
```

Application handles can be retrieved by issuing an SQLMA_APPLINFO_ALL request (list applications). An application connecting to the database can also retrieve its application handle (agent_id) from the sqlca of the CONNECT request (See "Obtaining application handle (AGENT_ID) from the CONNECT request").

Reading the Snapshot Output Buffer

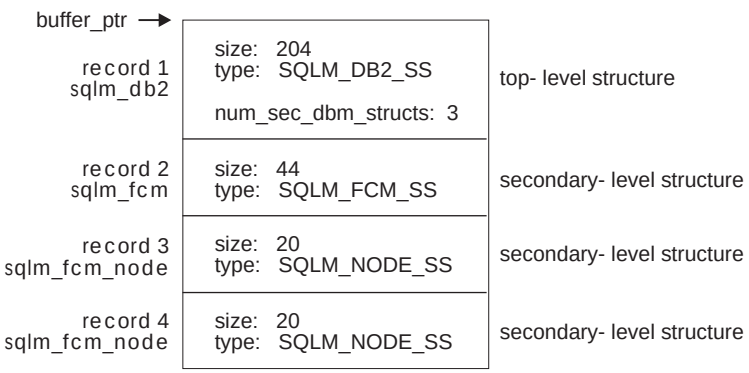
The sqlmonss() routine returns data as contiguous data structures in the user supplied buffer.

The sqlmon.h header file contains the definitions for all records returned by the sqlmonss() routine. It is the first place you should look for record information. It contains comments that explain how records are laid out in the output buffer. You may want to print a copy of this file, and reference it as you read this section.

The data structures returned in the snapshot output buffer are arranged in a two-level hierarchy:

- top-level structures
- secondary-level structures. These always come in the buffer following their parent top-level structure.

Each record contains a field that specifies its **type** and a **size** field that specifies its total size in bytes. The size **must be used** to read and skip this record in the output buffer. For example, following an SQLMA_DB2 snapshot request (GET SNAPSHOT FOR DATABASE MANAGER) on a parallel system. The buffer could contain:



The **sqlm_collected** output structure indicates the number of top-level structures returned in the buffer. Each top-level structure indicates the number of secondary-level structures that may follow it.

Attention: WARNING: It is imperative that you always use the size field for skipping a record in the output buffer. **Never use sizeof() on a snapshot record.**

Your application should also always read the record type. For some snapshot requests, the order in which records are returned is not guaranteed, and some record may not be

returned when a monitor switch is OFF (see “sqlmrset - Reset Monitor API” on page 263 and “sqlmon - Get/Update Monitor Switches API” on page 244).

Loop for reading snapshot output buffer

The following code illustrates how an application should be reading and skipping the records returned in the snapshot output buffer.

```
#include "stdlib.h"
#include "stdio.h"
#include "sqlutil.h"
#include "string.h"
#include "sqlmon.h"  // System Monitor interface
   //-------------------------------------------------------------------------
   // PROCESS EACH RECORD THAT MAY BE RETURNED IN THE SNAPSHOT OUTPUT BUFFER
   //-------------------------------------------------------------------------
   while (collected.num_top_level_structs--) {

      // Check the record type, (5th byte of any top-level structure)
      switch ((unsigned char) *(snap+sizeof(unsigned long))) {
      case SQLM_DB2_SS: {
          sqlm_db2 *db2_snap;
          db2_snap = (sqlm_db2*) snap;
          // Process the database manager snapshot
          printf("Processing database manager snapshot\n");
          // ...

          // Skip all its records in the output buffer
          snap += db2_snap->size;

          // Skip the secondary level entries
          while (db2_snap->num_sec_dbm_structs--) snap+=(*(unsigned long*)snap);
          } break;
      case SQLM_DBASE_SS: {
          sqlm_dbase *db_snap = (sqlm_dbase*) snap;
          // Process the snapshot ...
          printf("Processing database snapshot\n");

          // Skip the database snapshot and any secondary structures
          snap += db_snap->size;
          while (db_snap->num_sec_dbase_structs--) snap+=(*(unsigned long*)snap);
          } break;
      case SQLM_APPL_SS: {
          sqlm_appl *appl_snap =  (sqlm_appl*) snap;
          printf("Processing application snapshot\n");
          while (appl_snap->num_sec_appl_structs--) snap+=(*(unsigned long*)snap);
          } break;
      case SQLM_APPLINFO_SS: {
          sqlm_applinfo *appinfo_snap = (sqlm_applinfo*) snap;
          printf("Processing list application\n");
          snap+=appinfo_snap->size;
          } break;
      case SQLM_DCS_APPLINFO_SS: {
          sqlm_dcs_applinfo *dcs_snap = (sqlm_dcs_applinfo*) snap;
          printf("Processing list dcs application\n");
          snap+=dcs_snap->size;
          } break;
      case SQLM_TABLE_HEADER_SS: {
          sqlm_table_header *tabh_snap = (sqlm_table_header*) snap;
          int numtabs  = tabh_snap->num_tables;
          printf("Processing list tables\n");
          //   ...
          // Skip it in the output buffer
          snap += tabh_snap->size;
          while (numtabs--) {
```

```
        sqlm_table *tab_snap= (sqlm_table*) snap;
        snap += tab_snap->size;
      }
      } break;
   case SQLM_DBASE_LOCK_SS: {
      sqlm_dbase_lock *dbase_lock_snap = (sqlm_dbase_lock*) snap;
      printf("Processing snapshot for locks on database\n");
      dump_dbase_lock(stdout,, dbase_lock_snap);
      // this routine provided the following section Printing Snapshot Output Records

      // Skip it in the snapshot output buffer
      snap = skip_db_lock_snap(dbase_lock_snap);
      } break;
   case SQLM_APPL_LOCK_SS: {
      sqlm_appl_lock* appl_lock_snap = (sqlm_appl_lock*) snap;
      printf("Processing snapshot for locks for application\n");

      // Skip it in the snapshot output buffer
      snap = skip_appl_lock_snap(appl_lock_snap);
      } break;
   case SQLM_TABLESPACE_HEADER_SS: {
      sqlm_tablespace_header *tspace_snap = (sqlm_tablespace_header*) snap;
      printf("Processing snapshot for tablespaces\n");

      // Skip it in the snapshot output buffer
      snap = skip_tspace_snap(tspace_snap);
      } break;
   default:
      printf("%s:%d: Unexpected record type %d in snapshot output buffer!\n",
            __FILE__, __LINE__, (unsigned char) *(snap+sizeof(unsigned long)));
   } // end check the current snapshot buffer structure
} // end while there are top-level structures in the snapshot output buffer
```

Note an anomaly to the interface: some top-level data structures do not have a field that returns the number of all generic secondary structures that follows. Instead, they specify the number of structures returned for a specific secondary-level type. Special treatment is required for these snapshot requests. For example, the following routines, referenced in the main loop above, skip lock and table space snapshot requests in the snapshot output buffer.

Routines to skip specific snapshot requests in the snapshot output buffer

```
//------------------------------------------------------------------
// Skip a get snapshot for locks for application (SQLM_APPL_LOCK_SS)
//------------------------------------------------------------------
char * skip_appl_lock_snap(sqlm_appl_lock * appl_lock_snap) {
   char *snap= (char*) appl_lock_snap;

   int numlocks=appl_lock_snap->num_locks;
   snap+=appl_lock_snap->size;

   // Skip the lock entries for this application
   while (numlocks--) snap+=(*(unsigned long*)snap);
   return snap;
} // end skip_appl_lock_snap

//------------------------------------------------------------------
// Skip a get snapshot for locks for database (SQLM_DBASE_LOCK_SS)
//------------------------------------------------------------------
char * skip_db_lock_snap(sqlm_dbase_lock * dbase_lock_snap) {
   char *snap= (char*) dbase_lock_snap;

   int numaplocks=dbase_lock_snap->num_appls;
   snap+=(*(unsigned long*)snap);
```

```
   // Skip the appl lock entries
   while (numaplocks--) {
      snap = skip_appl_lock_snap((sqlm_appl_lock*) snap);
   }
   return snap;
} // end skip_db_lock_snap

//------------------------------------------------------------------
// Skip a get snapshot for table spaces (SQLM_DBASE_LOCK_SS)
//------------------------------------------------------------------
char * skip_tspace_snap(sqlm_tablespace_header * ts_header) {
   char *snap= (char*) ts_header;

   int numtspaces = ts_header->num_tablespaces;
   snap+=ts_header->size;  // Skip the header

   // Skip the sqlm_tablespace entries
   while (numtspaces--) snap+=(*(unsigned long*)snap);
   return snap;
} // end skip_db_lock_snap
```

Printing Snapshot Output Records

Attention: No string is NULL-terminated.

All strings returned in snapshot output records are blank-padded up to their maximum length.

The following example illustrates how a GET SNAPSHOT FOR LOCKS on database can be printed.

```
//------------------------------------------------------------------------------
// Print a Blank Padded String of maximum length SZ
//------------------------------------------------------------------------------
// note: strings returned by sqlmonss are NOT NULL-TERMINATED, they are all
//       blank padded up to some maximum length.
//------------------------------------------------------------------------------
#define dump_BPSTRING(fp, str, SZ) \
{ \
    int k=0; \
    while (str[k]!=' '&&k<SZ) k++; \
    if (k<SZ) str[k]='\0'; \
    fprintf(fp, #str": %0.*s\n", SZ, str); \
}
//------------------------------------------------------------------------------
// Map Application Status to a printable string
//------------------------------------------------------------------------------
// note: These #define may be found in sqlmon.h
//------------------------------------------------------------------------------
char* appl_status_string(int val) {
    switch (val) {
    case SQLM_CONNECTPEND:        return "SQLM_CONNECTPEND";
    case SQLM_CONNECTED:          return "SQLM_CONNECTED";
    case SQLM_UOWEXEC:            return "SQLM_UOWEXEC";
    case SQLM_UOWWAIT:            return "SQLM_UOWWAIT";
    case SQLM_LOCKWAIT:           return "SQLM_LOCKWAIT";
    case SQLM_COMMIT_ACT:         return "SQLM_COMMIT_ACT";
    case SQLM_ROLLBACK_ACT:       return "SQLM_ROLLBACK_ACT";
    case SQLM_RECOMP:             return "SQLM_RECOMP";
    case SQLM_COMP:               return "SQLM_COMP";
    case SQLM_INTR:               return "SQLM_INTR";
    case SQLM_DISCONNECTPEND:     return "SQLM_DISCONNECTPEND";
    case SQLM_TPREP:              return "SQLM_TPREP";
    case SQLM_THCOMT:             return "SQLM_THCOMT";
```

```
    case SQLM_THABRT:             return "SQLM_THABRT";
    case SQLM_TEND:               return "SQLM_TEND";
    case SQLM_CREATE_DB:          return "SQLM_CREATE_DB";
    case SQLM_RESTART:            return "SQLM_RESTART";
    case SQLM_RESTORE:            return "SQLM_RESTORE";
    case SQLM_BACKUP:             return "SQLM_BACKUP";
    case SQLM_LOAD:               return "SQLM_LOAD";
    case SQLM_UNLOAD:             return "SQLM_UNLOAD";
    case SQLM_IOERROR_WAIT:       return "SQLM_IOERROR_WAIT";
    case SQLM_QUIESCE_TABLESPACE: return "SQLM_QUIESCE_TABLESPACE";
    }
    return "";
} // end of appl_status_string
//----------------------------------------------------------------------------
// Map lock_object_type to a printable string
//----------------------------------------------------------------------------
char* lock_object_type_string(int val) {
    char *result="";
    switch (val) {
    case SQLM_TABLE_LOCK:        result = "SQLM_TABLE_LOCK";
    case SQLM_ROW_LOCK:          result = "SQLM_ROW_LOCK";
    case SQLM_INTERNAL_LOCK:     result = "SQLM_INTERNAL_LOCK";
    case SQLM_TABLESPACE_LOCK:   result = "SQLM_TABLESPACE_LOCK";
    case 0:                      result = "No lock wait";
    }
    return result;
} // end of lock_object_type_string
//----------------------------------------------------------------------------
// Map lock_mode to a printable string
//----------------------------------------------------------------------------
char* lock_mode_string(int val) {
    char result="";
    switch (val) {
    case SQLM_LNON: result = "NO";
    case SQLM_LOIS: result = "IS";
    case SQLM_LOIX: result = "IX";
    case SQLM_LOOS: result = "S";
    case SQLM_LSIX: result = "SIX";
    case SQLM_LOOX: result = "X";
    case SQLM_LOIN: result = "IN";
    case SQLM_LOOZ: result = "Z";
    case SQLM_LOOU: result = "U";
    case SQLM_LONS: result = "NS";
    case SQLM_LONX: result = "NX";
    }
    return result;
}
//----------------------------------------------------------------------------
// Map lock_status to a printable string
//----------------------------------------------------------------------------
inline char* lock_status_string(int val) {
    switch (val) {
    case SQLM_LRBGRNT: result = "Granted";
    case SQLM_LRBCONV: result = "Converting";
    }
    return result;
}

//----------------------------------------------------------------------------
// Print an sqlm_dbase_lock
//----------------------------------------------------------------------------
void dump_sqlm_dbase_lock(FILE* fp, sqlm_dbase_lock* db_lock) {
    fprintf(fp,"\nsqlm_dbase_lock contains: \n");
    fprintf(fp,"db_lock->info_type: %s\n",        "SQLM_DBASE_LOCK_SS");
    fprintf(fp,"db_lock->locks_held: %ld\n",      db_lock->locks_held);
    fprintf(fp,"db_lock->appls_cur_cons: %ld\n",  db_lock->appls_cur_cons);
    fprintf(fp,"db_lock->num_appls: %ld\n",       db_lock->num_appls);
```

```
    fprintf(fp,"db_lock->locks_waiting: %ld\n",   db_lock->locks_waiting);
    dump_BPSTRING(fp, db_lock->input_db_alias, SQLM_DBPATH_SZ);
    dump_BPSTRING(fp, db_lock->db_name, SQLM_IDENT_SZ);
    dump_BPSTRING(fp, db_lock->db_path, SQLM_DBPATH_SZ);
} // end of dump_sqlm_dbase_lock
//----------------------------------------------------------------------------
// Print an sqlm_appl_lock
// routine referred to in earlier section 'Loop for reading snapshot output buffer'
//----------------------------------------------------------------------------
void dump_sqlm_appl_lock(FILE* fp, sqlm_appl_lock* appl_lock) {
    fprintf(fp,"\nsqlm_appl_lock contains: \n");
    fprintf(fp,"appl_lock->info_type: %s\n",    "SQLM_APPL_LOCK_SS");
    fprintf(fp,"appl_lock->agent_id: %6.6ld\n", appl_lock->agent_id);
    fprintf(fp,"appl_lock->appl_status: %s\n",
                                appl_status_string(appl_lock->appl_status));
    fprintf(fp,"appl_lock->codepage_id: %ld\n", appl_lock->codepage_id);
    fprintf(fp,"appl_lock->locks_held: %ld\n",  appl_lock->locks_held);
    fprintf(fp,"appl_lock->num_locks: %ld\n",   appl_lock->num_locks);
    // Print the status change time, only if non-zero
    if (appl_lock->status_change_time.seconds) {
        fprintf(fp,"appl_lock->status_change_time: %s\n",
                ctime((time_t*) &appl_lock->status_change_time.seconds));
    } // end if status changed
    dump_BPSTRING(fp, appl_lock->appl_id,         SQLM_APPLID_SZ);
    dump_BPSTRING(fp, appl_lock->sequence_no,     SQLM_SEQ_SZ);
    dump_BPSTRING(fp, appl_lock->appl_name,       SQLM_IDENT_SZ);
    dump_BPSTRING(fp, appl_lock->auth_id,         SQLM_IDENT_SZ);
    dump_BPSTRING(fp, appl_lock->client_db_alias, SQLM_IDENT_SZ);
    // The following information is returned only if the application is in
    // a lock wait (otherwise it is zeroed, or set to blank spaces if a string):
    if (appl_lock->appl_status==SQLM_LOCKWAIT) {
        fprintf(fp,"appl_lock->lock_object_name: %ld\n",
                   appl_lock->lock_object_name);
        fprintf(fp,"appl_lock->agent_id_holding_lk: %6.6ld\n",
                   appl_lock->agent_id_holding_lk);
        fprintf(fp,"appl_lock->lock_object_type: %s\n",
                  lock_object_type_string( appl_lock->lock_object_type));
        fprintf(fp,"appl_lock->table_file_id: %ld\n", appl_lock->table_file_id);
        dump_BPSTRING(fp, appl_lock->appl_id_holding_lk,     SQLM_APPLID_SZ);
        dump_BPSTRING(fp, appl_lock->sequence_no_holding_lk, SQLM_SEQ_SZ);
        dump_BPSTRING(fp, appl_lock->table_name,      SQLM_IDENT_SZ);
        dump_BPSTRING(fp, appl_lock->table_schema,    SQLM_IDENT_SZ);
        dump_BPSTRING(fp, appl_lock->tablespace_name, SQLM_IDENT_SZ);
    } // end if this application is in a lock_wait
} // end of dump_sqlm_appl_lock

//----------------------------------------------------------------------------
// Print an sqlm_lock
//----------------------------------------------------------------------------
void dump_sqlm_lock(FILE* fp, sqlm_lock* lock) {
    fprintf(fp,"\nsqlm_lock contains: \n");
    fprintf(fp,"lock->info_type: %s\n", "SQLM_LOCK_SS");
    fprintf(fp,"lock->lock_object_type: %s\n",
                      lock_object_type_string(lock->lock_object_type));
    // Print the object of this lock
    switch (lock->lock_object_type) {
    case SQLM_ROW_LOCK:
    case SQLM_TABLE_LOCK:
        dump_BPSTRING(fp, lock->table_name,   SQLM_IDENT_SZ);
        dump_BPSTRING(fp, lock->table_schema, SQLM_IDENT_SZ);
        fprintf(fp,"lock->table_file_id: %ld\n", lock->table_file_id);
        break;
    case SQLM_TABLESPACE_LOCK:
        dump_BPSTRING(fp, lock->tablespace_name, SQLM_IDENT_SZ);
        break;
    case SQLM_INTERNAL_LOCK:
        break;
```

```
    }
    fprintf(fp,"lock->lock_mode: %s\n",  lock_mode_string(lock->lock_mode));
    fprintf(fp,"lock->lock_status: %s\n",lock_status_string(lock->lock_status));
    fprintf(fp,"lock->lock_object_name: %ld\n", lock->lock_object_name);
} // end of dump_sqlm_lock
```

Sample Programs

C \sqllib\samples\c\dbsnap.c

See Also

"sqlmon - Get/Update Monitor Switches API" on page 244
"sqlmonsz - Estimate Size Required for sqlmonss() Output Buffer API" on page 260
"sqlmrset - Reset Monitor API" on page 263.
"GET SNAPSHOT Command" on page 232.

sqlmonsz - Estimate Size Required for sqlmonss() Output Buffer API

Purpose

Estimates the buffer size needed by “sqlmonss - Get Snapshot API” on page 248.

Scope

This API only affects the instance to which the calling application is attached.

Authorization

One of the following:

sysadm
sysctrl
sysmaint

Required Connection

Instance. To obtain information from a remote instance (or a different local instance), it is necessary to first attach to that instance. If an attachment does not exist, an implicit instance attachment is made to the node specified by the **DB2INSTANCE** environment variable.

API Include File

sqlmon.h
sqlca.h

C API Syntax

```
/* File: sqlmon.h */
/* API: Estimate Database System Monitor Buffer Size */
/* ... */
int SQL_API_FN
  sqlmonsz (
    unsigned long  version,
    _SQLOLDCHAR    *reserved,
    sqlma          *sqlma_ptr,
    unsigned long  *buff_size,
    struct sqlca   *sqlca);
/* ... */
```

API Parameters

sqlca

Output. A pointer to the *sqlca* structure where the database manager returns error information. See *API Reference* for more information.

buff_size

Output. A pointer to the returned estimated buffer size needed by the GET SNAPSHOT API.

sqlma_ptr

Input. Pointer to the user-allocated *sqlma* data structure containing an array of **sqlm_obj_struct**, each one being a request for monitored data (see "sqlmonss - Get Snapshot API" on page 248).

reserved

Reserved for future use. Must be set to NULL.

version

Input. Version ID of the database monitor data to collect. The database monitor only returns data that was available for the requested version. Set this parameter to one of the following symbolic constants:

- `SQLM_DBMON_VERSION1`
- `SQLM_DBMON_VERSION2`
- `SQLM_DBMON_VERSION5`

If requesting data for a version higher than the current server, the database monitor only returns data for its level.

Note: If `SQLM_DBMON_VERSION1` is specified as the version, the APIs cannot be run remotely.

Comments

The size returned by *sqlmonsz* is always a little larger than what is actually required.

This function generates a significant amount of overhead (it basically takes a snapshot that does not return any data). Allocating and freeing memory dynamically for every **sqlmonss** call is also expensive. If calling **sqlmonss** repeatedly, for example, when sampling data over a period of time, it may be preferable to allocate a buffer of fixed size (for example 32K), possibly from the stack, and continue reusing that same buffer, rather than call **sqlmonsz**. If SQLM_RC_BUFFER_FULL is returned then you can call *sqlmonsz* and allocate a new larger buffer.

If the database system monitor finds no active databases or applications, it may return a buffer size of zero (if, for example, lock information related to a database that is not active is requested). Verify that the estimated buffer size returned by this API is non-zero before calling "sqlmonss - Get Snapshot API" on page 248.

See Also

"sqlmon - Get/Update Monitor Switches API" on page 244
"sqlmonss - Get Snapshot API" on page 248
"sqlmrset - Reset Monitor API" on page 263.

Code Sample

The following example estimates the buffer size required when issuing a snapshot for locks, tables, and database level information.

```
/*
    Database Monitor - Estimate Buffer Size for Snapshot
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "sqlca.h"
#include "sqlutil.h"
#include "sqlmon.h"
main () {
    char *dbname = "SAMPLE";   // Name of the database to monitor
    int rc;
    struct sqlca sqlca;
    unsigned long buffer_sz;
    //----------------------------------------------------------------------------
    // Request SQLMA_DBASE, SQLM_DBASE_TABLES, and SQLMA_DBASE_LOCKS in sqlma
    //----------------------------------------------------------------------------
    struct sqlma* sqlma = (struct sqlma *) malloc(SQLMASIZE(3));
    sqlma->obj_num = 3;
    sqlma->obj_var[0].obj_type = SQLMA_DBASE;
    strcpy(sqlma->obj_var[0].object, dbname);
    sqlma->obj_var[1].obj_type = SQLMA_DBASE_LOCKS;
    strcpy(sqlma->obj_var[1].object, dbname);
    sqlma->obj_var[2].obj_type = SQLMA_DBASE_TABLES;
    strcpy(sqlma->obj_var[2].object, dbname);
    //----------------------------------------------------------------------------
    // Estimate Buffer size required for this request
    //----------------------------------------------------------------------------
    sqlmonsz(SQLM_DBMON_VERSION5, NULL, sqlma, &buffer_sz, &sqlca);
    if (sqlca.sqlcode) {   // note: Positive return codes indicate a Warning
        // get and display a printable error message
        char msg[1024];
        sqlaintp (msg, sizeof(msg), 60, &sqlca);
        printf("%s", msg);
    }
    else printf ("\nBuffer size required for this snapshot is: %d\n",buffer_sz);
    //----------------------------------------------------------------------------
    // ...Take the Snapshot...
    //----------------------------------------------------------------------------
    {
        char*  buffer_ptr;
        sqlm_collected collected;
        buffer_ptr = (char *) malloc(buffer_sz);   // Allocate the buffer
        rc = sqlmonss(SQLM_DBMON_VERSION5, NULL, sqlma, buffer_sz, buffer_ptr,
                 &collected, &sqlca);
        // ....  Process snapshot output in buffer_ptr ...
        delete buffer_ptr;  // Free the buffer
    }
}
```

sqlmrset - Reset Monitor API

Purpose

Resets the database system monitor data of a specified database, or of all active databases for the application issuing the call.

Scope

This API only affects the application making the call.

Authorization

One of the following:

sysadm
sysctrl
sysmaint

Required Connection

Instance. To reset the monitor switches for a remote instance (or a different local instance), it is necessary to first attach to that instance.

API Include File

sqlmon.h

C API Syntax

```
/* File: sqlmon.h */
/* API: Reset Monitor */
/* ... */
int SQL_API_FN
  sqlmrset (
    unsigned long  version,
    _SQLOLDCHAR    *reserved,
    unsigned long  reset_all,
    _SQLOLDCHAR    *db_alias,
    struct sqlca   *sqlca);
/* ... */
```

API Parameters

reset_all

Input. An input value of 0 (zero) resets monitor counters for the database specified in **db_alias**. An input value of 1 resets monitor counters for all databases and at the database manager level (db_alias is then ignored).

sqlmrset - Reset Monitor API

db_alias
Input. A null terminated string that identifies the alias of the database being reset. It is ignored if **reset_all** is set to 1.

sqlca
Output. A pointer to the *sqlca* structure where the database manager returns error information.

version
Input. Version ID of the database monitor data to reset. Use the same value that you would specify for the sqlmonss call.

Sample Programs

C \sqllib\samples\c\monreset.c

Comments

Each process (attachment) has its own private view of the monitor data. If one user resets, or turns off a monitor switch, other users are not affected. When an application first calls any database monitor function, it inherits the default switch settings from the database manager configuration file.These settings can be overridden with “sqlmon - Get/Update Monitor Switches API” on page 244.

If all active databases are reset, some database manager information is also reset to maintain the consistency of the data that is returned.

This API cannot be used to selectively reset specific data items or specific monitor groups. However a specific group can be reset by turning its switch OFF and then ON, using “sqlmon - Get/Update Monitor Switches API” on page 244.

See Also

“sqlmon - Get/Update Monitor Switches API” on page 244
“sqlmonss - Get Snapshot API” on page 248
“sqlmonsz - Estimate Size Required for sqlmonss() Output Buffer API” on page 260.
“RESET MONITOR Command” on page 241.

Code Sample

The following example resets the resettable data elements for a single database.

```
/*
    Database Monitor - Reset Data Elements
*/
#include <stdio.h>
#include "sqlca.h"
#include "sqlutil.h"
#include "sqlmon.h"
//-----------------------------------------------------------------------------
// Reset monitored data for a single database
//-----------------------------------------------------------------------------
   struct sqlca sqlca;
   sqlmrset(SQLM_DBMON_VERSION5, NULL, SQLM_OFF, "SAMPLE", &sqlca);
   if (sqlca.sqlcode) {   // note: Positive return codes indicate a Warning
       // get and display a printable error message
       char msg[1024];
       sqlaintp (msg, sizeof(msg), 60, &sqlca);
       printf("%s", msg);
```

```
} // end if reset was successful
else printf("Database Monitor Reset for '%s' was successful!\n", argv[1]);
```

UPDATE MONITOR SWITCHES Command

Purpose

Turns one or more database monitor recording switches on or off.

The database manager records a base set of information at all times. Users who require more than this basic information can turn on the appropriate switches, but at a cost to system performance. Switches control information that is expensive to collect. See Chapter 3, “Database System Monitor Data Elements” on page 31 to determine if a switch is required for a particular data element.

Authorization

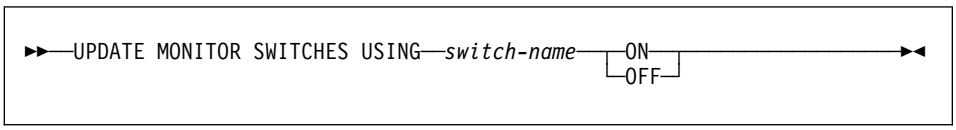
One of the following:

sysadm
sysctrl
sysmaint

Required Connection

Instance.

Format



Parameters

USING *switch-name*
Elements under switch control are grouped as follows:

BUFFERPOOL	Buffer pool activity information
LOCK	Lock information
SORT	Sorting information
STATEMENT	SQL statement information.
TABLE	Table activity information
UOW	Unit of work information.

Comments

To view the switch settings for your session, use “GET MONITOR SWITCHES Command” on page 230.

To clear the information related to a particular switch, set the switch off, then on.

When the application starts, it inherits the settings in the database manager configuration. See GET DATABASE MANAGER CONFIGURATION in the *Command Reference* and *dft_mon_xxx* configuration parameters.

See Also

"RESET MONITOR Command" on page 241
"GET MONITOR SWITCHES Command" on page 230
"UPDATE MONITOR SWITCHES Command" on page 266
"GET SNAPSHOT Command" on page 232
"sqlmonss - Get Snapshot API" on page 248

UPDATE MONITOR SWITCHES Command

Appendix B. Parallel Edition Version 1.2 Users

In DB2 Version 5 the database system monitor interface has been simplified and is now the same for all database and system configurations. This harmonization of the interface means that some of the request types that were available with the Parallel Edition (PE) V1.2 system monitor are no longer supported.

The most significant change affects how you monitor an application. In DB2 Version 5 an application snapshot returns all the relevaent application information, including a breakdown of the application statistics at the subsection or agent level (if applicable). For example, assuming an application is running a query composed of several subsections, a GET SNAPHOT FOR APPLICATION will return:

- Lock wait information for each agent that is working for this application and is waiting for a lock.
- Tablequeue activity for each subsection executed by this application. This allows you to track progression of a query that is against a partitioned database.
- A list of process IDs or thread IDs for each agent associated with the application.

This information is available on both the coordinator and non-coordinator nodes. In PE V1.2, you would have to request information about individual agents or tablequeues and correlate the output obtained at these levels with the application.

Note: PE V1.2 applications are not compatible with DB2 Version 5.

PE V1.2 applications that are not using any of the requests that have are obsolete in DB2 Version 5 can be recompiled after changing the request type from SQLM_DBMON_PARALLEL1 to SQLM_DBMON_VERSION1. No other changes should be required. See the following tables for obsolete requests.

agent_id

You should note that **agent_id** no longer corresponds to the process ID of the agent process. This field has not been renamed in the API to ensure source compatibility with previous versions, however it has become a globally unique identifier for the application.

Agent ID and application handle are synonymous. See “Partitioned Database Considerations” on page 23 for more information.

API Changes

Obsolete sqlmonss() Request Type	Description	Replacement
SQLMA_AGENT_APPL SQLMA_AGENT_AGENTID	Get snapshot for **agent**	Replaced with SQLMA_APPL which will report a breakdown per agent, if and when applicable.
SQLMA_COORD_AGENTS	List all coordinator agents	Replaced with SQLMA_APPLINFO_ALL, which returns sqlm_applinfo for each application. It identifies the node where the coordinator agent runs and provides both its application handle and agent thread or process ID.
SQLMA_FCM_NODE_ALL SQLMA_FCM_NODE	Get Fast Communication Manager	Replaced with SQLMA_DB2, which gets **all** database manager information and returns FCM information (if applicable).
SQLMA_AGENT_ALL SQLMA_COORD_AGENTS	Get snapshot for all agents Get snapshot for coordinator agents	In PE V1.2, returned an SQLMA_AGENT_AGENTID snapshot for all agents, including the coordinators (or the coordinators only). Replaced with SQLMA_APPL_ALL (GET SNAPSHOT FOR APPLICATIONS). Note that information is not returned for agents that are not associated with any applications, as their counts would be zeroes.
SQLMA_DBASE_AGENTS	Get snapshot for all agents for a database	Replaced with SQLMA_DBASE_APPLS.

Obsolete Commands

Obsolete PE V1.2 Command	Replacement
get snapshot for all agents	get snapshot for all applications
get snapshot for all coord agents	get snapshot for all applications
get snapshot for agents on dbname	get snapshot for applications on dbname
get snapshot for agents for application	get snapshot for application
get snapshot for coordinating agent	get snapshot for application
get snapshot for tablequeues	get snapshot for application

Note: GET SNAPSHOT FOR FCM is still supported, however the command processor maps it to a GET SNAPSHOT FOR DBM and extracts the FCM information from the returned output.

Appendix C. DB2 Version 1 sqlestat Users

The following information previously available with the sqlestat API on OS/2 for DB2 Version 1 is now available through snapshot monitoring.

sqlestat Name	Data Element
component_id	“Product Identification” on page 38
corr_serv_lvl	“Service Level” on page 37
curr_reqs_lvl	“SQL Requests Since Last Commit” on page 173
db_type	“Server Operating System” on page 37
location	“Database Location” on page 43
node	“Catalog Node Network Name” on page 42
product_name	“Product Name” on page 38

Appendix D. How the DB2 Library Is Structured

The DB2 Universal Database library consists of SmartGuides, online help, and books. This section describes the information that is provided, and how to access it.

To help you access product information online, DB2 provides the Information Center on OS/2, Windows 95, and the Windows NT operating systems. You can view task information, DB2 books, troubleshooting information, sample programs, and DB2 information on the Web. “About the Information Center” on page 280 has more details.

SmartGuides

SmartGuides help you complete some administration tasks by taking you through each task one step at a time. SmartGuides are available on OS/2, Windows 95, and the Windows NT operating systems. The following table lists the SmartGuides.

SmartGuide	Helps you to...	How to Access...
Add Database	Catalog a database on a client workstation.	From the Client Configuration Assistant, click on **Add**.
Create Database	Create a database, and to perform some basic configuration tasks.	From the Control Center, click with the right mouse button on the **Databases** icon and select **Create**->**New**.
Performance Configuration	Tune the performance of a database by updating configuration parameters to match your business requirements.	From the Control Center, click with the right mouse button on the database you want to tune and select **Configure performance**.
Backup Database	Determine, create, and schedule a backup plan.	From the Control Center, click with the right mouse button on the database you want to backup and select **Backup**->**Database using SmartGuide**.
Restore Database	Recover a database after a failure. It helps you understand which backup to use, and which logs to replay.	From the Control Center, click with the right mouse button on the database you want to restore and select **Restore**->**Database using SmartGuide**.
Create Table	Select basic data types, and create a primary key for the table.	From the Control Center, click with the right mouse button on the **Tables** icon and select **Create**->**Table using SmartGuide**.
Create Table Space	Create a new table space.	From the Control Center, click with the right mouse button on the **Table spaces** icon and select **Create**->**Table space using SmartGuide**.

Online Help

Online help is available with all DB2 components. The following table describes the various types of help.

Type of Help	Contents	How to Access...
Command Help	Explains the syntax of commands in the command line processor.	From the command line processor in interactive mode, enter: **?** *command* where *command* is a keyword or the entire command. For example, **?** *catalog* displays help for all the CATALOG commands, whereas **?** *catalog database* displays help for the CATALOG DATABASE command.
Control Center Help	Explains the tasks you can perform in a window or notebook. The help includes prerequisite information you need to know, and describes how to use the window or notebook controls.	From a window or notebook, click on the **Help** push button or press the F1 key.
Message Help	Describes the cause of a message number, and any action you should take.	From the command line processor in interactive mode, enter: **?** *message number* where *message number* is a valid message number. For example, **?** *SQL30081* displays help about the SQL30081 message. To view message help one screen at a time, enter: **?** *XXXnnnnn* **| more** where *XXX* is the message prefix, such as SQL, and *nnnnn* is the message number, such as 30081. To save message help in a file, enter: **?** *XXXnnnnn* > *filename.ext* where *filename.ext* is the file where you want to save the message help. **Note:** On UNIX-based systems, enter: **\?** *XXXnnnnn* **| more** or **\?** *XXXnnnnn* > *filename.ext*

Type of Help	Contents	How to Access...
SQL Help	Explains the syntax of SQL statements.	From the command line processor in interactive mode, enter: **help** *statement* where *statement* is an SQL statement. For example, **help** *SELECT* displays help about the SELECT statement.
SQLSTATE Help	Explains SQL states and class codes.	From the command line processor in interactive mode, enter: **?** *sqlstate* or **?** *class-code* where *sqlstate* is a valid five digit SQL state and *class-code* is a valid two digit class code. For example, **?** *08003* displays help for the 08003 SQL state, whereas **?** *08* displays help for the 08 class code.

DB2 Books

The table in this section lists the DB2 books. They are divided into two groups:

- Cross-platform books: These books are for DB2 on any of the supported platforms.
- Platform-specific books: These books are for DB2 on a specific platform. For example, there is a separate *Quick Beginnings* book for DB2 on OS/2, Windows NT, and UNIX-based operating systems.

Most books are available in HTML and PostScript format, and in hardcopy that you can order from IBM. The exceptions are noted in the table.

You can obtain DB2 books and access information in a variety of different ways:

View To view an HTML book, you can do the following:

- If you are running DB2 administration tools on OS/2, Windows 95, or the Windows NT operating systems, you can use the Information Center. “About the Information Center” on page 280 has more details.
- Use the open file function of the Web browser supplied by DB2 (or one of your own) to open the following page:

  sqllib/doc/html/index.htm

  The page contains descriptions of and links to the DB2 books. The path is located on the drive where DB2 is installed.

  You can also open the page by double-clicking on the **DB2 Online Books** icon. Depending on the system you are using, the icon is in the main product folder or the Windows Start menu.

Search To search for information in the HTML books, you can do the following:

- Click on **Search the DB2 Books** at the bottom of any page in the HTML books. Use the search form to find a specific topic.
- Click on **Index** at the bottom of any page in an HTML book. Use the Index to find a specific topic in the book.
- Display the Table of Contents or Index of the HTML book, and then use the find function of the Web browser to find a specific topic in the book.
- Use the bookmark function of the Web browser to quickly return to a specific topic.
- Use the search function of the Information Center to find specific topics. “About the Information Center” on page 280 has more details.

Print To print a book on a PostScript printer, look for the file name shown in the table.

Order To order a hardcopy book from IBM, use the form number.

Book Name	Book Description	Form Number File Name
	Cross-Platform Books	
Administration Getting Started	Introduces basic DB2 database administration concepts and tasks, and walks you through the primary administrative tasks.	S10J-8154 db2k0x50
Administration Guide	Contains information required to design, implement, and maintain a database to be accessed either locally or in a client/server environment.	S10J-8157 db2d0x50
API Reference	Describes the DB2 application programming interfaces (APIs) and data structures you can use to manage your databases. Explains how to call APIs from your applications.	S10J-8167 db2b0x50
CLI Guide and Reference	Explains how to develop applications that access DB2 databases using the DB2 Call Level Interface, a callable SQL interface that is compatible with the Microsoft ODBC specification.	S10J-8159 db2l0x50
Command Reference	Explains how to use the command line processor, and describes the DB2 commands you can use to manage your database.	S10J-8166 db2n0x50
DB2 Connect Enterprise Edition Quick Beginnings	Provides planning, installing, configuring, and using information for DB2 Connect Enterprise Edition. Also contains installation and setup information for all supported clients.	S10J-7888 db2cyx50
DB2 Connect Personal Edition Quick Beginnings	Provides planning, installing, configuring, and using information for DB2 Connect Personal Edition.	S10J-8162 db2c1x50
DB2 Connect User's Guide	Provides concepts, programming and general using information about the DB2 Connect products.	S10J-8163 db2c0x50
DB2 Connectivity Supplement	Provides setup and reference information for customers who want to use DB2 for AS/400, DB2 for OS/390, DB2 for MVS, or DB2 for VM as DRDA Application Requesters with DB2 Universal Database servers, and customers who want to use DRDA Application Servers with DB2 Connect (formerly DDCS) application requesters. **Note:** Available in HTML and PostScript formats only.	No form number db2h1x50
Embedded SQL Programming Guide	Explains how to develop applications that access DB2 databases using embedded SQL, and includes discussions about programming techniques and performance considerations.	S10J-8158 db2a0x50
Glossary	Provides a comprehensive list of all DB2 terms and definitions. **Note:** Available in HTML format only.	No form number db2t0x50

Book Name	Book Description	Form Number File Name
Installing and Configuring DB2 Clients	Provides installation and setup information for all DB2 Client Application Enablers and DB2 Software Developer's Kits. **Note:** Available in HTML and PostScript formats only.	No form number db2iyx50
Master Index	Contains a cross reference to the major topics covered in the DB2 library. **Note:** Available in PostScript format and hardcopy only.	S10J-8170 db2w0x50
Message Reference	Lists messages and codes issued by DB2, and describes the actions you should take.	S10J-8168 db2m0x50
Replication Guide and Reference	Provides planning, configuring, administering, and using information for the IBM Replication tools supplied with DB2.	S95H-0999 db2e0x50
Road Map to DB2 Programming	Introduces the different ways your applications can access DB2, describes key DB2 features you can use in your applications, and points to detailed sources of information for DB2 programming.	S10J-8155 db2u0x50
SQL Getting Started	Introduces SQL concepts, and provides examples for many constructs and tasks.	S10J-8156 db2y0x50
SQL Reference	Describes SQL syntax, semantics, and the rules of the language. Also includes information about release-to-release incompatibilities, product limits, and catalog views.	S10J-8165 db2s0x50
System Monitor Guide and Reference	Describes how to collect different kinds of information about your database and the database manager. Explains how you can use the information to understand database activity, improve performance, and determine the cause of problems.	S10J-8164 db2f0x50
Troubleshooting Guide	Helps you determine the source of errors, recover from problems, and use diagnostic tools in consultation with DB2 Customer Service.	S10J-8169 db2p0x50
What's New	Describes the new features, functions, and enhancements in DB2 Universal Database. **Note:** Available in HTML and PostScript formats only.	No form number db2q0x50
	Platform-Specific Books	
Building Applications for UNIX Environments	Provides environment setup information and step-by-step instructions to compile, link, and run DB2 applications on a UNIX system.	S10J-8161 db2axx50
Building Applications for Windows and OS/2 Environments	Provides environment setup information and step-by-step instructions to compile, link, and run DB2 applications on a Windows or OS/2 system.	S10J-8160 db2a1x50

Book Name	Book Description	Form Number File Name
DB2 Extended Enterprise Edition Quick Beginnings	Provides planning, installing, configuring, and using information for DB2 Universal Database Extended Enterprise Edition for AIX.	S72H-9620 db2v3x50
DB2 Personal Edition Quick Beginnings	Provides planning, installing, configuring, and using information for DB2 Universal Database Personal Edition on OS/2, Windows 95, and the Windows NT operating systems.	S10J-8150 db2i1x50
DB2 SDK for Macintosh Building Your Applications	Provides environment setup information and step-by-step instructions to compile, link, and run DB2 applications on a Macintosh system. **Note:** Available in PostScript format and hardcopy for DB2 Version 2.1.2 only.	S50H-0528 sqla7x02
DB2 SDK for SCO OpenServer Building Your Applications	Provides environment setup information and step-by-step instructions to compile, link, and run DB2 applications on a SCO OpenServer system. **Note:** Available for DB2 Version 2.1.2 only.	S89H-3242 sqla9x02
DB2 SDK for Silicon Graphics IRIX Building Your Applications	Provides environment setup information and step-by-step instructions to compile, link, and run DB2 applications on a Silicon Graphics system. **Note:** Available in PostScript format and hardcopy for DB2 Version 2.1.2 only.	S89H-4032 sqlaax02
DB2 SDK for SINIX Building Your Applications	Provides environment setup information and step-by-step instructions to compile, link, and run DB2 applications on a SINIX system. **Note:** Available in PostScript format and hardcopy for DB2 Version 2.1.2 only.	S50H-0530 sqla8x00
Quick Beginnings for OS/2	Provides planning, installing, configuring, and using information for DB2 Universal Database on OS/2. Also contains installing and setup information for all supported clients.	S10J-8147 db2i2x50
Quick Beginnings for UNIX	Provides planning, installing, configuring, and using information for DB2 Universal Database on UNIX-based platforms. Also contains installing and setup information for all supported clients.	S10J-8148 db2ixx50
Quick Beginnings for Windows NT	Provides planning, installing, configuring, and using information for DB2 Universal Database on the Windows NT operating system. Also contains installing and setup information for all supported clients.	S10J-8149 db2i6x50

Notes:

1. The character in the sixth position of the file name indicates the language of a book. For example, the file name `db2d0e50` indicates that the *Administration Guide* is in English. The following letters are used in the file names to indicate the language of a book:

Language	Identifier	Language	Identifier
Brazilian Portuguese	B	Hungarian	H
Bulgarian	U	Italian	I
Czech	X	Norwegian	N
Danish	D	Polish	P
English	E	Russian	R
Finnish	Y	Slovenian	L
French	F	Spanish	Z
German	G	Swedish	S

2. For late breaking information that could not be included in the DB2 books, see the README file. Each DB2 product includes a README file which you can find in the directory where the product is installed.

About the Information Center

The Information Center provides quick access to DB2 product information. The Information Center is available on OS/2, Windows 95, and the Windows NT operating systems. You must install the DB2 administration tools to see the Information Center.

Depending on your system, you can access the Information Center from the:

- Main product folder
- Toolbar in the Control Center
- Windows Start menu.

The Information Center provides the following kinds of information. Click on the appropriate tab to look at the information:

Tasks Lists tasks you can perform using DB2.

Reference Lists DB2 reference information, such as keywords, commands, and APIs.

Books Lists DB2 books.

Troubleshooting Lists categories of error messages and their recovery actions.

Sample Programs Lists sample programs that come with the DB2 Software Developer's Kit. If the Software Developer's Kit is not installed, this tab is not displayed.

Web Lists DB2 information on the World Wide Web. To access this information, you must have a connection to the Web from your system.

When you select an item in one of the lists, the Information Center launches a viewer to display the information. The viewer might be the system help viewer, an editor, or a Web browser, depending on the kind of information you select.

The Information Center provides search capabilities so you can look for specific topics, and filter capabilities to limit the scope of your searches.

Appendix E. Notices

Any reference to an IBM licensed program in this publication is not intended to state or imply that only IBM's licensed program may be used. Any functionally equivalent product, program or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the

IBM Director of Licensing,
IBM Corporation,
500 Columbus Avenue,
Thornwood, NY, 10594
USA.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Canada Limited
Department 071
1150 Eglinton Ave. East
North York, Ontario
M3C 1H7
CANADA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

This publication may contain examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Trademarks

The following terms are trademarks or registered trademarks of the IBM Corporation in the United States and/or other countries:

ACF/VTAM
ADSTAR
AISPO
AIX
AIXwindows
AnyNet
APPN
AS/400
CICS
C Set++
C/370
DATABASE 2
DatagLANce
DataHub
DataJoiner
DataPropagator
DataRefresher
DB2
Distributed Relational Database Architecture
DRDA
Extended Services
FFST
First Failure Support Technology
IBM
IMS
Lan Distance
MVS/ESA
MVS/XA
NetView
OS/400
OS/390
OS/2
PowerPC
QMF
RACF
RISC System/6000
SAA
SP
SQL/DS
SQL/400
S/370
System/370
System/390
SystemView
VisualAge
VM/ESA
VSE/ESA
VTAM
WIN-OS/2

Trademarks of Other Companies

The following terms are trademarks or registered trademarks of the companies listed:

C-bus is a trademark of Corollary, Inc.

HP-UX is a trademark of Hewlett-Packard.

Java and HotJava are trademarks of Sun Microsystems, Inc.

Microsoft, Windows, Windows NT, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation.

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

Solaris is a trademark of Sun Microsystems, Inc.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Other company, product, or service names, which may be denoted by a double asterisk (**), may be trademarks or service marks of others.

Index

A

B

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W

X

Contacting IBM

This section lists ways you can get more information from IBM.

If you have a technical problem, please take the time to review and carry out the actions suggested by the *Troubleshooting Guide* before contacting DB2 Customer Support. Depending on the nature of your problem or concern, this guide will suggest information you can gather to help us to serve you better.

For information or to order any of the DB2 Universal Database products contact an IBM representative at a local branch office or contact any authorized IBM software remarketer.

Telephone

If you live in the U.S.A., call one of the following numbers:

- 1-800-237-5511 to learn about available service options.
- 1-800-IBM-CALL (1-800-426-2255) or 1-800-3IBM-OS2 (1-800-342-6672) to order products or get general information.
- 1-800-879-2755 to order publications.

For information on how to contact IBM outside of the United States, see Appendix A of the IBM Software Support Handbook. You can access this document by selecting the "Roadmap to IBM Support" item at: http://www.ibm.com/support/.

Note that in some countries, IBM-authorized dealers should contact their dealer support structure instead of the IBM Support Center.

World Wide Web
http://www.software.ibm.com/data/
http://www.software.ibm.com/data/db2/library/

The DB2 World Wide Web pages provide current DB2 information about news, product descriptions, education schedules, and more. The DB2 Product and Service Technical Library provides access to frequently asked questions, fixes, books, and up-to-date DB2 technical information. (Note that this information may be in English only.)

Anonymous FTP Sites
ftp.software.ibm.com

Log on as anonymous. In the directory /ps/products/db2, you can find demos, fixes, information, and tools concerning DB2 and many related products.

Internet Newsgroups
comp.databases.ibm-db2, bit.listserv.db2-l

These newsgroups are available for users to discuss their experiences with DB2 products.

CompuServe
GO IBMDB2 to access the IBM DB2 Family forums

All DB2 products are supported through these forums.

To find out about the IBM Professional Certification Program for DB2 Universal Database, go to http://www.software.ibm.com/data/db2/db2tech/db2cert.html

Part Number: 10J8164

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.