IBM® DB2 Universal Database™

Data Recovery and High Availability Guide and Reference

Version 8

SC09-4831-00

IBM® DB2 Universal Database™

Data Recovery and High Availability Guide and Reference

Version 8

SC09-4831-00

Before using this information and the product it supports, be sure to read the general information under *Notices*.

You can order IBM publications online or through your local IBM representative.

- To order publications online, go to the IBM Publications Center at www.ibm.com/shop/publications/order
- To find your local IBM representative, go to the IBM Directory of Worldwide Contacts at www.ibm.com/planetwide

To order DB2 publications from DB2 Marketing and Sales in the United States or Canada, call 1-800-IBM-4YOU (426-4968).

Contents

About This Book

This book provides detailed information about, and shows you how to use, the IBM DB2 Universal Database (UDB) backup, restore, and recovery utilities. The book also explains the importance of high availability, and describes DB2 failover support on several platforms.

Who Should Use this Book

This manual is for database administrators, application programmers, and other DB2 UDB users who are responsible for, or who want to understand, backup, restore, and recovery operations on DB2 database systems.

It is assumed that you are familiar with DB2 Universal Database, Structured Query Language (SQL), and with the operating system environment in which DB2 UDB is running. This manual does not contain instructions for installing DB2, which depend on your operating system.

How this Book is Structured

The following topics are covered:

Data Recovery

Chapter 1, "Developing a Good Backup and Recovery Strategy"
Discusses factors to consider when choosing database and table space recovery methods, including backing up and restoring a database or table space, and using rollforward recovery.

Chapter 1, "Developing a Good Backup and Recovery Strategy"
Describes the DB2 backup utility, used to create backup copies of a database or table spaces.

Chapter 3, "Database Restore"
Describes the DB2 restore utility, used to rebuild damaged or corrupted databases or table spaces that were previously backed up.

Chapter 4, "Rollforward Recovery"
Describes the DB2 rollforward utility, used to recover a database by applying transactions that were recorded in the database recovery log files.

High Availability

Chapter 5, "Introducing High Availability and Failover Support"
Presents an overview of the high availability failover support that is provided by DB2.

Chapter 6, "High Availability on AIX"
Discusses DB2 support for high availability failover recovery on AIX, which is currently implemented through the Enhanced Scalability (ES) feature of High Availability Cluster Multi-processing (HACMP) for AIX.

Chapter 7, "High Availability on the Windows Operating System"
Discusses DB2 support for high availability failover recovery on Windows operating systems which is currently implemented through Microsoft Cluster Server (MSCS).

Chapter 8, "High Availability in the Solaris Operating Environment"
Discusses DB2 support for high availability failover recovery in the Solaris Operating Environment, which is currently implemented through Sun Cluster 3.0 (SC3.0) or Veritas Cluster Server (VCS).

Appendixes

Appendix A, "How to Read the Syntax Diagrams"
Explains the conventions used in syntax diagrams.

Appendix B, "Warning, Error and Completion Messages"
Provides information about interpreting messages generated by the database manager when a warning or error condition has been detected.

Appendix C, "Additional DB2 Commands"
Describes recovery-related DB2 commands.

Appendix D, "Additional APIs and Associated Data Structures"
Describes recovery-related APIs and their data structures.

Appendix E, "Recovery Sample Program"
Provides the code listing for a sample program containing recovery-related DB2 APIs and embedded SQL calls, and information on how to use them.

Appendix F, "Tivoli Storage Manager"
Provides information about the Tivoli Storage Manager (TSM, formerly ADSM) product, which you can use to manage database or table space backup operations.

Appendix G, "User Exit for Database Recovery"
Discusses how user exit programs can be used with database log files, and describes some sample user exit programs.

Appendix H, “Backup and Restore APIs for Vendor Products”
Describes the function and use of APIs that enable DB2 to interface with other vendor software.

Part 1. Data Recovery

Chapter 1. Developing a Good Backup and Recovery Strategy

This section discusses factors to consider when choosing database and table space recovery methods, including backing up and restoring a database or table space, and using rollforward recovery.

The following topics are covered:

Developing a Backup and Recovery Strategy

A database can become unusable because of hardware or software failure, or both. You may, at one time or another, encounter storage problems, power interruptions, and application failures, and different failure scenarios require different recovery actions. Protect your data against the possibility of loss by having a well rehearsed recovery strategy in place. Some of the questions that you should answer when developing your recovery strategy are: Will the database be recoverable? How much time can be spent recovering the database? How much time will pass between backup operations? How much storage space can be allocated for backup copies and archived logs? Will table space level backups be sufficient, or will full database backups be necessary?

A database recovery strategy should ensure that all information is available when it is required for database recovery. It should include a regular schedule for taking database backups and, in the case of partitioned database systems, include backups when the system is scaled (when database partition servers or nodes are added or dropped). Your overall strategy should also include procedures for recovering command scripts, applications, user-defined functions (UDFs), stored procedure code in operating system libraries, and load copies.

Different recovery methods are discussed in the sections that follow, and you will discover which recovery method is best suited to your business environment.

The concept of a database *backup* is the same as any other data backup: taking a copy of the data and then storing it on a different medium in case of failure or damage to the original. The simplest case of a backup involves shutting down the database to ensure that no further transactions occur, and then simply backing it up. You can then rebuild the database if it becomes damaged or corrupted in some way.

The rebuilding of the database is called *recovery*. *Version recovery* is the restoration of a previous version of the database, using an image that was created during a backup operation. *Rollforward recovery* is the reapplication of transactions recorded in the database log files after a database or a table space backup image has been restored.

Crash recovery is the automatic recovery of the database if a failure occurs before all of the changes that are part of one or more units of work (transactions) are completed and committed. This is done by rolling back incomplete transactions and completing committed transactions that were still in memory when the crash occurred.

Recovery log files and the recovery history file are created automatically when a database is created (Figure 1 on page 5). These log files are important if you need to recover data that is lost or damaged. You cannot directly modify a recovery log file or the recovery history file; however, you can delete entries from the recovery history file using the PRUNE HISTORY command. You can also use the *rec_his_retentn* database configuration parameter to specify the number of days that the recovery history file will be retained.

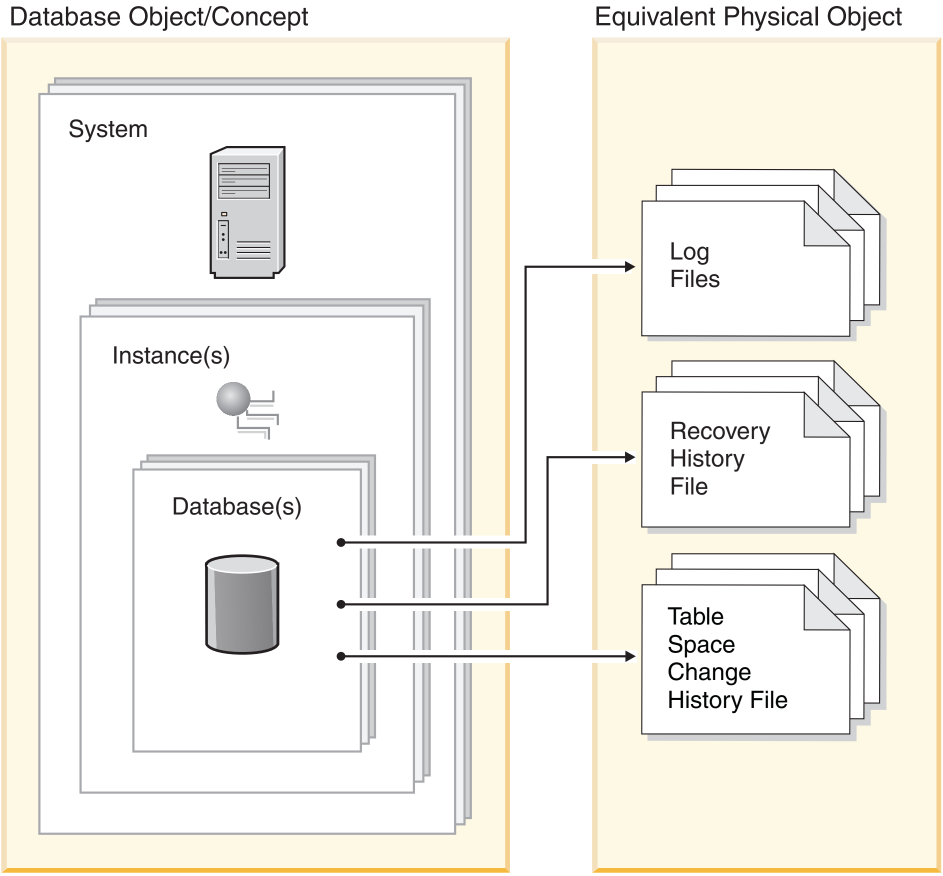


Figure 1. Recovery Log Files and the Recovery History File

Each database includes *recovery logs*, which are used to recover from application or system errors. In combination with the database backups, they are used to recover the consistency of the database right up to the point in time when the error occurred.

The *recovery history file* contains a summary of the backup information that can be used to determine recovery options, if all or part of the database must be recovered to a given point in time. It is used to track recovery-related events such as backup and restore operations, among others. This file is located in the database directory.

The *table space change history file*, which is also located in the database directory, contains information that can be used to determine which log files are required for the recovery of a particular table space.

Data that is easily recreated can be stored in a non-recoverable database. This includes data from an outside source that is used for read-only applications, and tables that are not often updated, for which the small amount of logging does not justify the added complexity of managing log files and rolling forward after a restore operation. *Non-recoverable databases* have both the *logretain* and the *userexit* database configuration parameters disabled. This means that the only logs that are kept are those required for crash recovery. These logs are known as *active logs*, and they contain current transaction data. Version recovery using *offline* backups is the primary means of recovery for a non-recoverable database. (An offline backup means that no other application can use the database when the backup operation is in progress.) Such a database can only be restored offline. It is restored to the state it was in when the backup image was taken and rollforward recovery is not supported.

Data that *cannot* be easily recreated should be stored in a recoverable database. This includes data whose source is destroyed after the data is loaded, data that is manually entered into tables, and data that is modified by application programs or users after it is loaded into the database. *Recoverable databases* have either the *logretain* database configuration parameter set to "RECOVERY", the *userexit* database configuration parameter enabled, or both. Active logs are still available for crash recovery, but you also have the *archived logs*, which contain committed transaction data. Such a database can only be restored offline. It is restored to the state it was in when the backup image was taken. However, with rollforward recovery, you can roll the database *forward* (that is, past the time when the backup image was taken) by using the active and archived logs to either a specific point in time, or to the end of the active logs.

Recoverable database backup operations can be performed either offline or *online* (online meaning that other applications can connect to the database during the backup operation). Database restore and rollforward operations must always be performed offline. During an online backup operation, rollforward recovery ensures that *all* table changes are captured and reapplied if that backup is restored.

If you have a recoverable database, you can back up, restore, and roll individual table spaces forward, rather than the entire database. When you back up a table space online, it is still available for use, and simultaneous updates are recorded in the logs. When you perform an online restore or rollforward operation on a table space, the table space itself is not available for use until the operation completes, but users are not prevented from accessing tables in other table spaces.

Related concepts:

- "Crash Recovery" on page 11

- "Version Recovery" on page 24
- "Rollforward Recovery" on page 25
- "Data Links server file backups" in the *Post V8 GA*
- "Failure and recovery overview" in the *DB2 Data Links Manager Administration Guide and Reference*

Related reference:

- "Recovery History Retention Period configuration parameter - rec_his_retentn" in the *Administration Guide: Performance*
- "DB2 Data Links Manager system setup and backup recommendations" in the *DB2 Data Links Manager Administration Guide and Reference*

Deciding How Often to Back Up

Your recovery plan should allow for regularly scheduled backup operations, because backing up a database requires time and system resources. Your plan may include a combination of full database backups and incremental backup operations.

You should take full database backups regularly, even if you archive the logs (which allows for rollforward recovery). It is more time consuming to rebuild a database from a collection of table space backup images than it is to recover the database from a full database backup image. Table space backup images are useful for recovering from an isolated disk failure or an application error.

You should also consider not overwriting backup images and logs, saving at least two full database backup images and their associated logs as an extra precaution.

If the amount of time needed to apply archived logs when recovering and rolling a very active database forward is a major concern, consider the cost of backing up the database more frequently. This reduces the number of archived logs you need to apply when rolling forward.

You can initiate a backup operation while the database is either *online* or *offline*. If it is online, other applications or processes can connect to the database, as well as read and modify data while the backup operation is running. If the backup operation is running offline, other applications *cannot* connect to the database.

To reduce the amount of time that the database is not available, consider using online backup operations. Online backup operations are supported only if rollforward recovery is enabled. If rollforward recovery is enabled and you have a complete set of recovery logs, you can rebuild the database, should the need arise. You can only use an online backup image for recovery if you have the logs that span the time during which the backup operation was running.

Offline backup operations are faster than online backup operations, since there is no contention for the data files.

The backup utility lets you back up selected table spaces. If you use DMS table spaces, you can store different types of data in their own table spaces to reduce the time required for backup operations. You can keep table data in one table space, long field and LOB data in another table space, and indexes in yet another table space. If you do this and a disk failure occurs, it is likely to affect only one of the table spaces. Restoring or rolling forward one of these table spaces will take less time than it would have taken to restore a single table space containing all of the data.

You can also save time by taking backups of different table spaces at different times, as long as the changes to them are not the same. So, if long field or LOB data is not changed as frequently as the other data, you can back up these table spaces less frequently. If long field and LOB data are not required for recovery, you can also consider not backing up the table space that contains that data. If the LOB data can be reproduced from a separate source, choose the NOT LOGGED option when creating or altering a table to include LOB columns.

Note: Consider the following if you keep your long field data, LOB data, and indexes in separate table spaces, but do not back them up together: If you back up a table space that does not contain all of the table data, you cannot perform point-in-time rollforward recovery on that table space. All the table spaces that contain any type of data for a table must be rolled forward simultaneously to the same point in time.

If you reorganize a table, you should back up the affected table spaces after the operation completes. If you have to restore the table spaces, you will not have to roll forward through the data reorganization.

The time required to recover a database is made up of two parts: the time required to complete the restoration of the backup; and, if the database is enabled for forward recovery, the time required to apply the logs during the rollforward operation. When formulating a recovery plan, you should take these recovery costs and their impact on your business operations into account. Testing your overall recovery plan will assist you in determining whether the time required to recover the database is reasonable given your business requirements. Following each test, you may want to increase the frequency with which you take a backup. If rollforward recovery is part of your strategy, this will reduce the number of logs that are archived between backups and, as a result, reduce the time required to roll the database forward after a restore operation.

Related concepts:

- "Incremental Backup and Recovery" on page 28

Related reference:

- Appendix G, "User Exit for Database Recovery" on page 323
- "Configuration Parameters for Database Logging" on page 39

Storage Considerations

When deciding which recovery method to use, consider the storage space required.

The version recovery method requires space to hold the backup copy of the database and the restored database. The rollforward recovery method requires space to hold the backup copy of the database or table spaces, the restored database, and the archived database logs.

If a table contains long field or large object (LOB) columns, you should consider placing this data into a separate table space. This will affect your storage space considerations, as well as affect your plan for recovery. With a separate table space for long field and LOB data, and knowing the time required to back up long field and LOB data, you may decide to use a recovery plan that only occasionally saves a backup of this table space. You may also choose, when creating or altering a table to include LOB columns, not to log changes to those columns. This will reduce the size of the required log space and the corresponding log archive space.

To prevent media failure from destroying a database and your ability to rebuild it, keep the database backup, the database logs, and the database itself on different devices. For this reason, it is highly recommended that you use the *newlogpath* configuration parameter to put database logs on a separate device once the database is created.

The database logs can use up a large amount of storage. If you plan to use the rollforward recovery method, you must decide how to manage the archived logs. Your choices are the following:

- Use a user exit program to copy these logs to another storage device in your environment.
- Manually copy the logs to a storage device or directory other than the database log path directory after they are no longer in the active set of logs.

Keeping Related Data Together

As part of your database design, you will know the relationships that exist between tables. These relationships can be expressed at the application level, when transactions update more than one table, or at the database level, where referential integrity exists between tables, or where triggers on one table affect another table. You should consider these relationships when developing a recovery plan. You will want to back up related sets of data together. Such sets can be established at either the table space or the database level. By keeping related sets of data together, you can recover to a point where all of the data is consistent. This is especially important if you want to be able to perform point-in-time rollforward recovery on table spaces.

Using Different Operating Systems

When working in an environment that has more than one operating system, you must consider that in most cases, the backup and recovery plans cannot be integrated. That is, you cannot usually back up a database on one operating system, and then restore that database on another operating system. In such cases, you should keep the recovery plans for each operating system separate and independent.

There is, however, support for cross-platform backup and restore operations between operating systems with similar architectures such as AIX® and Sun Solaris, and between 32 bit and 64 bit operating systems. When you transfer the backup image between systems, you must transfer it in binary mode. The target system must have the same (or later) version of DB2® as the source system. Restore operations to a down-level system are not supported.

If you must move tables from one operating system to another, and cross-platform backup and restore support is not available in your environment, you can use the **db2move** command, or the export utility followed by the import or the load utility.

Related reference:

- "db2move - Database Movement Tool" in the *Command Reference*
- "EXPORT" in the *Command Reference*
- "IMPORT" in the *Command Reference*
- "LOAD" in the *Command Reference*

Crash Recovery

Transactions (or units of work) against a database can be interrupted unexpectedly. If a failure occurs before all of the changes that are part of the unit of work are completed and committed, the database is left in an inconsistent and unusable state. *Crash recovery* is the process by which the database is moved back to a consistent and usable state. This is done by rolling back incomplete transactions and completing committed transactions that were still in memory when the crash occurred (Figure 2). When a database is in a consistent and usable state, it has attained what is known as a ″point of consistency″.

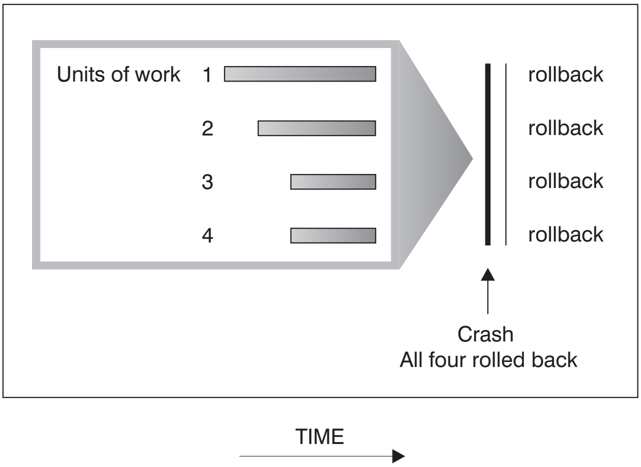


Figure 2. Rolling Back Units of Work (Crash Recovery)

A *transaction failure* results from a severe error or condition that causes the database or the database manager to end abnormally. Partially completed units of work, or UOW that have not been flushed to disk at the time of failure, leave the database in an inconsistent state. Following a transaction failure, the database must be recovered. Conditions that can result in transaction failure include:

- A power failure on the machine, causing the database manager and the database partitions on it to go down
- A serious operating system error that causes DB2® to go down
- A hardware failure such as memory corruption, or disk, CPU, or network failure.

If you want the rollback of incomplete units of work to be done automatically by the database manager, enable the automatic restart (*autorestart*) database configuration parameter by setting it to `ON`. (This is the default value.) If you do not want automatic restart behavior, set the *autorestart* database configuration parameter to OFF. As a result, you will need to issue the RESTART DATABASE command when a database failure occurs. If the database I/O was suspended before the crash occurred, you must specify the WRITE RESUME option of the RESTART DATABASE command in order for the crash recovery to continue. The administration notification log records when the database restart operation begins.

If crash recovery is applied to a database that is enabled for forward recovery (that is, the *logretain* configuration parameter is set to RECOVERY, or the *userexit* configuration parameter is enabled), and an error occurs during crash recovery that is attributable to an individual table space, that table space will be taken offline, and cannot be accessed until it is repaired. Crash recovery continues. At the completion of crash recovery, the other table spaces in the database will be accessible, and connections to the database can be established. However, if the table space that is taken offline is the table space that contains the system catalogs, it must be repaired before any connections will be permitted.

Related reference:

- "Auto Restart Enable configuration parameter - autorestart" in the *Administration Guide: Performance*

Crash Recovery - Details

Recovering Damaged Table Spaces

A damaged table space has one or more containers that cannot be accessed. This is often caused by media problems that are either permanent (for example, a bad disk), or temporary (for example, an offline disk, or an unmounted file system).

If the damaged table space is the system catalog table space, the database cannot be restarted. If the container problems cannot be fixed leaving the original data intact, the only available options are:

- To restore the database
- To restore the catalog table space. (Table space restore is only valid for recoverable databases, because the database must be rolled forward.)

If the damaged table space is *not* the system catalog table space, DB2® attempts to make as much of the database available as possible.

If the damaged table space is the only temporary table space, you should create a new temporary table space as soon as a connection to the database can be made. Once created, the new temporary table space can be used, and normal database operations requiring a temporary table space can resume. You can, if you wish, drop the offline temporary table space. There are special considerations for table reorganization using a system temporary table space:

- If the database or the database manager configuration parameter *indexrec* is set to RESTART, all invalid indexes must be rebuilt during database activation; this includes indexes from a reorganization that crashed during the build phase.
- If there are incomplete reorganization requests in a damaged temporary table space, you may have to set the *indexrec* configuration parameter to ACCESS to avoid restart failures.

Recovering Table Spaces in Recoverable Databases

When crash recovery is necessary, a damaged table space will be taken offline and will not be accessible. It will be placed in roll forward pending state. A restart operation will succeed if there are no additional problems, and the damaged table space can be used again once you:

- Fix the damaged containers without losing the original data, and then complete a table space rollforward operation to the end of the logs. (The rollforward operation will first attempt to bring it from offline to normal state.)
- Perform a table space restore operation after fixing the damaged containers (with or without losing the original data), and then a rollforward operation to the end of the logs or to a point-in-time.

Recovering Table Spaces in Non-recoverable Databases

Since crash recovery is necessary, and logs are not kept indefinitely, the restart operation can only succeed if the user is willing to drop the damaged table spaces. (Successful completion of recovery means that the log records necessary to recover the damaged table spaces to a consistent state will be gone; therefore, the only valid action against such table spaces is to drop them.)

You can do this by invoking an unqualified restart database operation. It will succeed if there are no damaged table spaces. If it fails (SQL0290N), you can look in the administration notification log file for a complete list of table spaces that are currently damaged.

- If you are willing to drop all of these table spaces once the restart database operation is complete, you can initiate another restart database operation, listing all of the damaged table spaces under the DROP PENDING TABLESPACES option. If a damaged table space is included in the DROP PENDING TABLESPACES list, the table space is put into drop pending state, and your only option after recovery is to drop the table space. The restart operation continues without recovering this table space. If a damaged table space is *not* included in the DROP PENDING TABLESPACES list, the restart database operation fails with SQL0290N.

- If you are unwilling to drop (and thus lose the data in) these table spaces, your options are to:
  - Wait and fix the damaged containers (without losing the original data), and then try the restart database operation again
  - Perform a database restore operation.

Note: Putting a table space name into the DROP PENDING TABLESPACES list does not mean that the table space will be in drop pending state. This will occur only if the table space is found to be damaged during the restart operation. Once the restart operation is successful, you should issue DROP TABLESPACE statements to drop each of the table spaces that are in drop pending state (invoke the LIST TABLESPACES command to find out which table spaces are in this state). This way the space can be reclaimed, or the table spaces can be recreated.

Reducing the Impact of Media Failure

To reduce the probability of media failure, and to simplify recovery from this type of failure:

- Mirror or duplicate the disks that hold the data and logs for important databases.
- Use a Redundant Array of Independent Disks (RAID) configuration, such as RAID Level 5.
- In a partitioned database environment, set up a rigorous procedure for handling the data and the logs on the catalog node. Because this node is critical for maintaining the database:
  - Ensure that it resides on a reliable disk
  - Duplicate it
  - Make frequent backups
  - Do not put user data on it.

Protecting Against Disk Failure

If you are concerned about the possibility of damaged data or logs due to a disk crash, consider the use of some form of disk fault tolerance. Generally, this is accomplished through the use of a *disk array*, which is a set of disks.

A disk array is sometimes referred to simply as a RAID (Redundant Array of Independent Disks). Disk arrays can also be provided through software at the operating system or application level. The point of distinction between hardware and software disk arrays is how CPU processing of input/output (I/O) requests is handled. For hardware disk arrays, I/O activity is managed by disk controllers; for software disk arrays, this is done by the operating system or an application.

Hardware Disk Arrays: In a hardware disk array, multiple disks are used and managed by a disk controller, complete with its own CPU. All of the logic required to manage the disks forming this array is contained on the disk controller; therefore, this implementation is operating system-independent.

There are several types of RAID architecture, differing in function and performance, but only RAID level 1 and level 5 are commonly used today.

RAID level 1 is also known as disk mirroring or duplexing. *Disk mirroring* copies data (a complete file) from one disk to a second disk, using a single disk controller. *Disk duplexing* is similar to disk mirroring, except that disks are attached to a second disk controller (like two SCSI adapters). Data protection is good: Either disk can fail, and data is still accessible from the other disk. With disk duplexing, a disk controller can also fail without compromising data protection. Performance is good, but this implementation requires twice the usual number of disks.

RAID level 5 involves data and parity striping by sectors, across all disks. Parity is interleaved with data, rather than being stored on a dedicated drive. Data protection is good: If any disk fails, the data can still be accessed by using information from the other disks, along with the striped parity information. Read performance is good, but write performance is not. A RAID level 5 configuration requires a minimum of three identical disks. The amount of disk space required for overhead varies with the number of disks in the array. In the case of a RAID level 5 configuration with 5 disks, the space overhead is 20 percent.

When using a RAID (but not a RAID level 0) disk array, a failed disk will not prevent you from accessing data on the array. When hot-pluggable or hot-swappable disks are used in the array, a replacement disk can be swapped with the failed disk while the array is in use. With RAID level 5, if two disks fail at the same time, all data is lost (but the probability of simultaneous disk failures is very small).

You might consider using a RAID level 1 hardware disk array or a software disk array for your logs, because this provides recoverability to the point of failure, and offers good write performance, which is important for logs. In cases where reliability is critical (because time cannot be lost recovering data following a disk failure), and write performance is not so critical, consider using a RAID level 5 hardware disk array. Alternatively, if write performance is critical, and the cost of additional disk space is not significant, consider a RAID level 1 hardware disk array for your data, as well as for your logs.

For detailed information about the available RAID levels, visit the following web site: http://www.acnc.com/04_01_00.html

Software Disk Arrays: A software disk array accomplishes much the same as does a hardware disk array, but disk traffic is managed by either the operating system, or by an application program running on the server. Like other programs, the software array must compete for CPU and system resources. This is not a good option for a CPU-constrained system, and it should be remembered that overall disk array performance is dependent on the server's CPU load and capacity.

A typical software disk array provides disk mirroring. Although redundant disks are required, a software disk array is comparatively inexpensive to implement, because costly disk controllers are not required.

CAUTION:
Having the operating system boot drive in the disk array prevents your system from starting if that drive fails. If the drive fails before the disk array is running, the disk array cannot allow access to the drive. A boot drive should be separate from the disk array.

Reducing the Impact of Transaction Failure

To reduce the impact of a transaction failure, try to ensure:

- An uninterrupted power supply
- Adequate disk space for database logs
- Reliable communication links among the database partition servers in a partitioned database environment
- Synchronization of the system clocks in a partitioned database environment.

Related concepts:

- "Synchronizing Clocks in a Partitioned Database System" on page 132

Recovering from Transaction Failures in a Partitioned Database Environment

If a transaction failure occurs in a partitioned database environment, database recovery is usually necessary on both the failed database partition server and any other database partition server that was participating in the transaction:

- Crash recovery occurs on the failed database partition server after the antecedent condition is corrected.
- *Database partition failure recovery* on the other (still active) database partition servers occurs immediately after the failure has been detected.

In a partitioned database environment, the database partition server on which an application is submitted is the coordinator node, and the first agent that works for the application is the coordinator agent. The coordinator agent is responsible for distributing work to other database partition servers, and it keeps track of which ones are involved in the transaction. When the application issues a COMMIT statement for a transaction, the coordinator agent commits the transaction by using the two-phase commit protocol. During the first phase, the coordinator node distributes a PREPARE request to all the other database partition servers that are participating in the transaction. These servers then respond with one of the following:

READ-ONLY	No data change occurred at this server
YES	Data change occurred at this server
NO	Because of an error, the server is not prepared to commit

If one of the servers responds with a NO, the transaction is rolled back. Otherwise, the coordinator node begins the second phase.

During the second phase, the coordinator node writes a COMMIT log record, then distributes a COMMIT request to all the servers that responded with a YES. After all the other database partition servers have committed, they send an acknowledgment of the COMMIT to the coordinator node. The transaction is complete when the coordinator agent has received all COMMIT acknowledgments from all the participating servers. At this point, the coordinator agent writes a FORGET log record.

Transaction Failure Recovery on an Active Database Partition Server

If any database partition server detects that another server is down, all work that is associated with the failed database partition server is stopped:

- If the still active database partition server is the coordinator node for an application, and the application was running on the failed database partition server (and not ready to COMMIT), the coordinator agent is interrupted to do failure recovery. If the coordinator agent is in the second phase of COMMIT processing, SQL0279N is returned to the application, which in turn loses its database connection. Otherwise, the coordinator agent distributes a ROLLBACK request to all other servers participating in the transaction, and SQL1229N is returned to the application.
- If the failed database partition server was the coordinator node for the application, agents that are still working for the application on the active servers are interrupted to do failure recovery. The current transaction is rolled back locally on each server, unless it has been prepared and is waiting for the transaction outcome. In this situation, the transaction is left in doubt on the active database partition servers, and the coordinator node is not aware of this (because it is not available).
- If the application connected to the failed database partition server (before it failed), but neither the local database partition server nor the failed database partition server is the coordinator node, agents working for this application are interrupted. The coordinator node will either send a

ROLLBACK or a disconnect message to the other database partition servers. The transaction will only be indoubt on database partition servers that are still active if the coordinator node returns SQL0279.

Any process (such as an agent or deadlock detector) that attempts to send a request to the failed server is informed that it cannot send the request.

Transaction Failure Recovery on the Failed Database Partition Server

If the transaction failure causes the database manager to end abnormally, you can issue the **db2start** command with the RESTART option to restart the database manager once the database partition has been restarted. If you cannot restart the database partition, you can issue **db2start** to restart the database manager on a different partition.

If the database manager ends abnormally, database partitions on the server may be left in an inconsistent state. To make them usable, crash recovery can be triggered on a database partition server:

- Explicitly, through the RESTART DATABASE command
- Implicitly, through a CONNECT request when the *autorestart* database configuration parameter has been set to `ON`

Crash recovery reapplies the log records in the active log files to ensure that the effects of all complete transactions are in the database. After the changes have been reapplied, all uncommitted transactions are rolled back locally, *except* for indoubt transactions. There are two types of indoubt transaction in a partitioned database environment:

- On a database partition server that is not the coordinator node, a transaction is in doubt if it is prepared but not yet committed.
- On the coordinator node, a transaction is in doubt if it is committed but not yet logged as complete (that is, the FORGET record is not yet written). This situation occurs when the coordinator agent has not received all the COMMIT acknowledgments from all the servers that worked for the application.

Crash recovery attempts to resolve all the indoubt transactions by doing one of the following. The action that is taken depends on whether the database partition server was the coordinator node for an application:

- If the server that restarted is not the coordinator node for the application, it sends a query message to the coordinator agent to discover the outcome of the transaction.
- If the server that restarted *is* the coordinator node for the application, it sends a message to all the other agents (subordinate agents) that the coordinator agent is still waiting for COMMIT acknowledgments.

It is possible that crash recovery may not be able to resolve all the indoubt transactions (for example, some of the database partition servers may not be available). In this situation, the SQL warning message SQL1061W is returned. Because indoubt transactions hold resources, such as locks and active log space, it is possible to get to a point where no changes can be made to the database because the active log space is being held up by indoubt transactions. For this reason, you should determine whether indoubt transactions remain after crash recovery, and recover all database partition servers that are required to resolve the indoubt transactions as quickly as possible.

If one or more servers that are required to resolve an indoubt transaction cannot be recovered in time, and access is required to database partitions on other servers, you can manually resolve the indoubt transaction by making an heuristic decision. You can use the LIST INDOUBT TRANSACTIONS command to query, commit, and roll back the indoubt transaction on the server.

Note: The LIST INDOUBT TRANSACTIONS command is also used in a distributed transaction environment. To distinguish between the two types of indoubt transactions, the *originator* field in the output that is returned by the LIST INDOUBT TRANSACTIONS command displays one of the following:

- DB2 Universal Database Enterprise - Extended Edition, which indicates that the transaction originated in a partitioned database environment.
- XA, which indicates that the transaction originated in a distributed environment.

Identifying the Failed Database Partition Server

When a database partition server fails, the application will typically receive one of the following SQLCODEs. The method for detecting which database manager failed depends on the SQLCODE received:

SQL0279N
This SQLCODE is received when a database partition server involved in a transaction is terminated during COMMIT processing.

SQL1224N
This SQLCODE is received when the database partition server that failed is the coordinator node for the transaction.

SQL1229N
This SQLCODE is received when the database partition server that failed is not the coordinator node for the transaction.

Determining which database partition server failed is a two-step process. The SQLCA associated with SQLCODE SQL1229N contains the node number of the server that detected the error in the sixth array position of the *sqlerrd* field. (The node number that is written for the server corresponds to the node number in the `db2nodes.cfg` file.) On the database partition server that detects the error, a message that indicates the node number of the failed server is written to the administration notification log.

Note: If multiple logical nodes are being used on a processor, the failure of one logical node may cause other logical nodes on the same processor to fail.

Related concepts:

- "Two-phase commit" in the *Administration Guide: Planning*
- "Error recovery during two-phase commit" in the *Administration Guide: Planning*

Related tasks:

- "Manually resolving indoubt transactions" in the *Administration Guide: Planning*

Related reference:

- "db2start - Start DB2" in the *Command Reference*
- "LIST INDOUBT TRANSACTIONS" in the *Command Reference*

Recovering from the Failure of a Database Partition Server

Procedure:

To recover from the failure of a database partition server:

1. Correct the problem that caused the failure.
2. Restart the database manager by issuing the **db2start** command from any database partition server.
3. Restart the database by issuing the RESTART DATABASE command on the failed database partition server or servers.

Related concepts:

- "Recovering from Transaction Failures in a Partitioned Database Environment" on page 16

Related reference:

- "db2start - Start DB2" in the *Command Reference*
- "RESTART DATABASE" in the *Command Reference*

Recovering Indoubt Transactions on the Host when DB2 Connect Has the DB2 Syncpoint Manager Configured

If your application has accessed a host or AS/400 database server during a transaction, there are some differences in how indoubt transactions are recovered.

To access host or AS/400 database servers, DB2 Connect is used. The recovery steps differ if DB2 Connect has the DB2 Syncpoint Manager configured.

Procedures:

The recovery of indoubt transactions at host or AS/400 servers is normally performed automatically by the Transaction Manager (TM) and the DB2 Syncpoint Manager (SPM). An indoubt transaction at a host or AS/400 server does not hold any resources at the local DB2 location, but does hold resources at the host or AS/400 server as long as the transaction is indoubt at that location. If the administrator of the host or AS/400 server determines that a heuristic decision must be made, then the administrator may contact the local DB2 database administrator (for example via telephone) to determine whether to commit or roll back the transaction at the host or AS/400 server. If this occurs, the LIST DRDA INDOUBT TRANSACTIONS command can be used to determine the state of the transaction at the DB2 Connect instance. The following steps can be used as a guideline for most situations involving an SNA communications environment.

1. Connect to the SPM as shown below:

   ```
   db2 => connect to db2spm

     Database Connection Information

     Database product       = SPM0500
     SQL authorization ID   = CRUS
     Local database alias   = DB2SPM
   ```

2. Issue the LIST DRDA INDOUBT TRANSACTIONS command to display the indoubt transactions known to the SPM. The example below shows one indoubt transaction known to the SPM. The db_name is the local alias for the host or AS/400 server. The partner_lu is the fully qualified luname of the host or AS/400 server. This provides the best identification of the host or AS/400 server, and should be provided by the caller from the host or AS/400 server. The luwid provides a unique identifier for a transaction and is available at all hosts and AS/400 servers. If the transaction in question is displayed, then the `uow_status` field can be used to determine the outcome of the transaction if the value is C (commit) or R (rollback). If you issue the LIST DRDA INDOUBT TRANSACTIONS command with the WITH PROMPTING parameter, you can commit, roll back, or forget the transaction interactively.

```
db2 => list drda indoubt transactions
 DRDA Indoubt Transactions:
 1.db_name: DBAS3    db_alias: DBAS3    role: AR
   uow_status: C  partner_status: I  partner_lu: USIBMSY.SY12DQA
 corr_tok: USIBMST.STB3327L
 luwid: USIBMST.STB3327.305DFDA5DC00.0001
 xid: 53514C2000000017 00000000544D4442 0000000000305DFD A63055E962000000
      00035F
```

3. If an indoubt transaction for the partner_lu and for the luwid is not displayed, or if the LIST DRDA INDOUBT TRANSACTIONS command returns as follows:

   ```
   db2 => list drda indoubt transactions
   SQL1251W  No data returned for heuristic query.
   ```

   then the transaction was rolled back.

   There is another unlikely but possible situation that may occur. If an indoubt transaction with the proper luwid for the partner_lu is displayed, but the uow_status is ″I″, the SPM doesn’t know whether the transaction is to be committed or rolled back. In this situation, you should use the WITH PROMPTING parameter to either commit or roll back the transaction on the DB2 Connect workstation. Then allow DB2 Connect to resynchronize with the host or AS/400 server based on the heuristic decision.

Related tasks:

- “Recovering Indoubt Transactions on the Host when DB2 Connect Does Not Use the DB2 Syncpoint Manager” on page 22

Related reference:

- “db2start - Start DB2” in the *Command Reference*
- “LIST INDOUBT TRANSACTIONS” in the *Command Reference*
- “RESTART DATABASE” in the *Command Reference*

Recovering Indoubt Transactions on the Host when DB2 Connect Does Not Use the DB2 Syncpoint Manager

If your application has accessed a host or AS/400 database server during a transaction, there are some differences in how indoubt transactions are recovered.

To access host or AS/400 database servers, DB2 Connect is used. The recovery steps differ if DB2 Connect has the DB2 Syncpoint Manager configured.

Procedure:

Use the information in this section when TCP/IP connectivity is used to update DB2 for OS/390 in a multisite update from either DB2 Connect Personal Edition or DB2 Connect Enterprise Server Edition, and the DB2 Syncpoint Manager is not used. The recovery of indoubt transactions in this situation differs from that for indoubt transactions involving the DB2 Syncpoint Manager. When an indoubt transaction occurs in this environment, an alert entry is generated at the client, at the database server, and (or) at the Transaction Manager (TM) database, depending on who detected the problem. The alert entry is placed in the `db2alert.log` file.

The resynchronization of any indoubt transactions occurs automatically as soon as the TM and the participating databases and their connections are all available again. You should allow automatic resynchronization to occur rather than heuristically force a decision at the database server. If, however, you must do this then use the following steps as a guideline.

Note: Because the DB2 Syncpoint Manager is not involved, you cannot use the LIST DRDA INDOUBT TRANSACTIONS command.

1. On the OS/390 host, issue the command DISPLAY THREAD TYPE(INDOUBT).

   From this list identify the transaction that you want to heuristically complete. For details about the DISPLAY command, see the *DB2 for OS/390 Command Reference*. The LUWID displayed can be matched to the same luwid at the Transaction Manager Database.
2. Issue the RECOVER THREAD(<LUWID>) ACTION(ABORT|COMMIT) command, depending on what you want to do.

   For details about the RECOVER command, see the *DB2 for OS/390 Command Reference*.

Related tasks:

- "Recovering Indoubt Transactions on the Host when DB2 Connect Has the DB2 Syncpoint Manager Configured" on page 21

Related reference:

- "LIST INDOUBT TRANSACTIONS" in the *Command Reference*

Disaster Recovery

The term *disaster recovery* is used to describe the activities that need to be done to restore the database in the event of a fire, earthquake, vandalism, or other catastrophic events. A plan for disaster recovery can include one or more of the following:

- A site to be used in the event of an emergency

- A different machine on which to recover the database
- Off-site storage of database backups and archived logs.

If your plan for disaster recovery is to recover the entire database on another machine, you require at least one full database backup and all the archived logs for the database. You may choose to keep a standby database up to date by applying the logs to it as they are archived. Or, you may choose to keep the database backup and log archives in the standby site, and perform restore and rollforward operations only after a disaster has occurred. (In this case, a recent database backup is clearly desirable.) With a disaster, however, it is generally not possible to recover all of the transactions up to the time of the disaster.

The usefulness of a table space backup for disaster recovery depends on the scope of the failure. Typically, disaster recovery requires that you restore the entire database; therefore, a full database backup should be kept at a standby site. Even if you have a separate backup image of every table space, you cannot use them to recover the database. If the disaster is a damaged disk, a table space backup of each table space on that disk can be used to recover. If you have lost access to a container because of a disk failure (or for any other reason), you can restore the container to a different location.

Both table space backups and full database backups can have a role to play in any disaster recovery plan. The DB2® facilities available for backing up, restoring, and rolling data forward provide a foundation for a disaster recovery plan. You should ensure that you have tested recovery procedures in place to protect your business.

Related concepts:

- "Redefining Table Space Containers During a Restore Operation (Redirected Restore)" on page 93

Version Recovery

Version recovery is the restoration of a previous version of the database, using an image that was created during a backup operation. You use this recovery method with non-recoverable databases (that is, databases for which you do not have archived logs). You can also use this method with recoverable databases by using the WITHOUT ROLLING FORWARD option on the RESTORE DATABASE command. A database restore operation will rebuild the entire database using a backup image created earlier. A database backup allows you to restore a database to a state identical to the one at the time that the backup was made. However, every unit of work from the time of the backup to the time of the failure is lost (see Figure 3 on page 25).

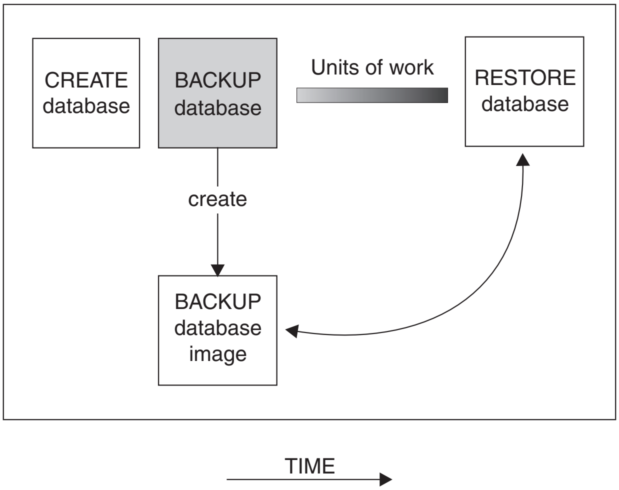


Figure 3. Version Recovery. The database is restored from the latest backup image, but all units of work processed between the time of backup and failure are lost.

Using the version recovery method, you must schedule and perform full backups of the database on a regular basis.

In a partitioned database environment, the database is located across many database partition servers (or nodes). You must restore all partitions, and the backup images that you use for the restore database operation must all have been taken at the same time. (Each database partition is backed up and restored separately.) A backup of each database partition taken at the same time is known as a *version backup*.

Rollforward Recovery

To use the *rollforward recovery* method, you must have taken a backup of the database, and archived the logs (by enabling either the *logretain* or the *userexit* database configuration parameters, or both). Restoring the database and specifying the WITHOUT ROLLING FORWARD option is equivalent to using the version recovery method. The database is restored to a state identical to the one at the time that the offline backup image was made. If you restore the database and do *not* specify the WITHOUT ROLLING FORWARD option for the restore database operation, the database will be in rollforward pending state at the end of the restore operation. This allows rollforward recovery to take place.

Note: The WITHOUT ROLLING FORWARD option cannot be used if the database backup was taken online.

The two types of rollforward recovery to consider are:

- *Database rollforward recovery*. In this type of rollforward recovery, transactions recorded in database logs are applied following the database restore operation (see Figure 4). The database logs record all changes made to the database. This method completes the recovery of the database to its state at a particular point in time, or to its state immediately before the failure (that is, to the end of the active logs.)

  In a partitioned database environment, the database is located across many database partitions. If you are performing point-in-time rollforward recovery, all database partitions must be rolled forward to ensure that all partitions are at the same level. If you need to restore a single database partition, you can perform rollforward recovery to the end of the logs to bring it up to the same level as the other partitions in the database. Only recovery to the end of the logs can be used if one database partition is being rolled forward. Point-in-time recovery applies to *all* database partitions.

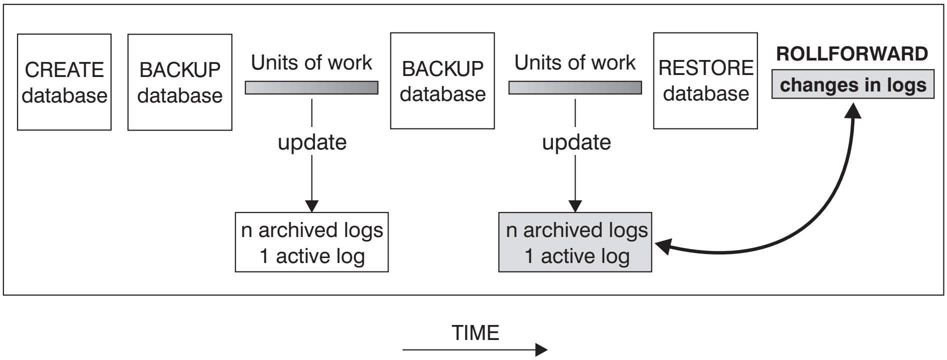


Figure 4. Database Rollforward Recovery. There can be more than one active log in the case of a long-running transaction.

- *Table space rollforward recovery*. If the database is enabled for forward recovery, it is also possible to back up, restore, and roll table spaces forward (see Figure 5 on page 27). To perform a table space restore and rollforward operation, you need a backup image of either the entire database (that is, all of the table spaces), or one or more individual table spaces. You also need the log records that affect the table spaces that are to be recovered. You can roll forward through the logs to one of two points:
  - The end of the logs; or,
  - A particular point in time (called *point-in-time* recovery).

Table space rollforward recovery can be used in the following two situations:

- After a table space restore operation, the table space is always in rollforward pending state, and it must be rolled forward. Invoke the

ROLLFORWARD DATABASE command to apply the logs against the table spaces to either a point in time, or to the end of the logs.

- If one or more table spaces are in *rollforward pending* state after crash recovery, first correct the table space problem. In some cases, correcting the table space problem does not involve a restore database operation. For example, a power loss could leave the table space in rollforward pending state. A restore database operation is not required in this case. Once the problem with the table space is corrected, you can use the ROLLFORWARD DATABASE command to apply the logs against the table spaces to the end of the logs. If the problem is corrected before crash recovery, crash recovery may be sufficient to take the database to a consistent, usable state.

  Note: If the table space in error contains the system catalog tables, you will not be able to start the database. You must restore the SYSCATSPACE table space, then perform rollforward recovery to the end of the logs.

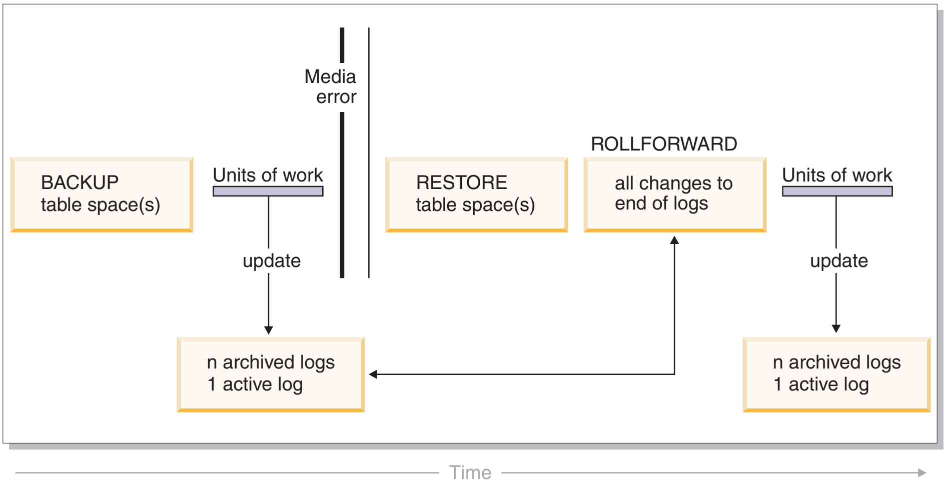


Figure 5. Table Space Rollforward Recovery. There can be more than one active log in the case of a long-running transaction.

In a partitioned database environment, if you are rolling a table space forward *to a point in time*, you do not have to supply the list of nodes (database partitions) on which the table space resides. DB2® submits the rollforward request to all partitions. This means the table space must be restored on all database partitions on which the table space resides.

In a partitioned database environment, if you are rolling a table space forward *to the end of the logs*, you must supply the list of database partitions if you do *not* want to roll the table space forward on all partitions. If you want to roll all table spaces (on all partitions) that are in rollforward pending state forward to the end of the logs, you do not have to supply the list of database partitions. By default, the database rollforward request is sent to all partitions.

Related concepts:

- "Understanding Recovery Logs" on page 34

Related reference:

- "ROLLFORWARD DATABASE" on page 134

Related samples:

- "dbrecov.out -- HOW TO RECOVER A DATABASE (C)"
- "dbrecov.sqc -- How to recover a database (C)"
- "dbrecov.out -- HOW TO RECOVER A DATABASE (C++)"
- "dbrecov.sqC -- How to recover a database (C++)"

Incremental Backup and Recovery

As the size of databases, and particularly warehouses, continues to expand into the terabyte and petabyte range, the time and hardware resources required to back up and recover these databases is also growing substantially. Full database and table space backups are not always the best approach when dealing with large databases, because the storage requirements for multiple copies of such databases are enormous. Consider the following issues:

- When a small percentage of the data in a warehouse changes, it should not be necessary to back up the entire database.
- Appending table spaces to existing databases and then taking only table space backups is risky, because there is no guarantee that nothing outside of the backed up table spaces has changed between table space backups.

DB2® now supports incremental backup and recovery (but not of long field or large object data). An *incremental backup* is a backup image that contains only pages that have been updated since the previous backup was taken. In addition to updated data and index pages, each incremental backup image also contains all of the initial database meta-data (such as database configuration, table space definitions, database history, and so on) that is normally stored in full backup images.

Two types of incremental backup are supported:

- *Incremental*. An incremental backup image is a copy of all database data that has changed since the most recent, successful, full backup operation. This is also known as a cumulative backup image, because a series of incremental backups taken over time will each have the contents of the previous incremental backup image. The predecessor of an incremental backup image is always the most recent successful full backup of the same object.

- *Delta*. A delta, or incremental delta, backup image is a copy of all database data that has changed since the last successful backup (full, incremental, or delta) of the table space in question. This is also known as a differential, or non-cumulative, backup image. The predecessor of a delta backup image is the most recent successful backup containing a copy of each of the table spaces in the delta backup image.

The key difference between incremental and delta backup images is their behavior when successive backups are taken of an object that is continually changing over time. Each successive incremental image contains the entire contents of the previous incremental image, plus any data that has changed, or is new, since the previous full backup was produced. Delta backup images contain only the pages that have changed since the previous image of any type was produced.

Combinations of database and table space incremental backups are permitted, in both online and offline modes of operation. Be careful when planning your backup strategy, because combining database and table space incremental backups implies that the predecessor of a database backup (or a table space backup of multiple table spaces) is not necessarily a single image, but could be a unique set of previous database and table space backups taken at different times.

To rebuild the database or the table space to a consistent state, the recovery process must begin with a consistent image of the entire object (database or table space) to be restored, and must then apply each of the appropriate incremental backup images in the order described below.

To enable the tracking of database updates, DB2 supports a new database configuration parameter, *trackmod*, which can have one of two accepted values:

- NO. Incremental backup is not permitted with this configuration. Database page updates are not tracked or recorded in any way. This is the default value.
- YES. Incremental backup is permitted with this configuration. When update tracking is enabled, the change becomes effective at the first successful connection to the database. Before an incremental backup can be taken on a particular table space, a full backup of that table space is necessary.

For SMS and DMS table spaces, the granularity of this tracking is at the table space level. In table space level tracking, a flag for each table space indicates whether or not there are pages in that table space that need to be backed up. If no pages in a table space need to be backed up, the backup operation can skip that table space altogether.

Although minimal, the tracking of updates to the database can have an impact on the runtime performance of transactions that update or insert data.

Related tasks:

- "Restoring from Incremental Backup Images" on page 30

Incremental Backup and Recovery - Details

Restoring from Incremental Backup Images

Procedure:

A restore operation from incremental backup images always consists of the following steps:

1. Identifying the incremental target image.

   Determine the final image to be restored, and request an incremental restore operation from the DB2 restore utility. This image is known as the target image of the incremental restore, because it will be the last image to be restored. The incremental target image is specified using the TAKEN AT parameter in the RESTORE DATABASE command.
2. Restoring the most recent full database or table space image to establish a baseline against which each of the subsequent incremental backup images can be applied.
3. Restoring each of the required full or table space incremental backup images, in the order in which they were produced, on top of the baseline image restored in Step 2.
4. Repeating Step 3 until the target image from Step 1 is read a second time. The target image is accessed twice during a complete incremental restore operation. During the first access, only initial data is read from the image; none of the user data is read. The complete image is read and processed only during the second access.

   The target image of the incremental restore operation must be accessed twice to ensure that the database is initially configured with the correct history, database configuration, and table space definitions for the database that will be created during the restore operation. In cases where a table space has been dropped since the initial full database backup image was taken, the table space data for that image will be read from the backup images but ignored during incremental restore processing.

There are two ways to restore incremental backup images.

- For a manual incremental restore, the RESTORE command must be issued once for each backup image that needs to be restored (as outlined in the steps above).
- For an automatic incremental restore, the RESTORE command is issued only once specifying the target image to be used. DB2 then uses the database history to determine the remaining required backup images and restores them.

Manual Incremental Restore Example

To restore a set of incremental backup images, using manual incremental restore, specify the target image using the TAKEN AT *timestamp* option of the RESTORE DATABASE command and follow the steps outlined above. For example:

```
1. db2 restore database sample incremental taken at <ts>

   where:
     <ts> points to the last incremental backup image (the target image)
      to be restored

2. db2 restore database sample incremental taken at <ts1>

   where:
     <ts1> points to the initial full database (or table space) image

3. db2 restore database sample incremental taken at <tsX>

   where:
     <tsX> points to each incremental backup image in creation sequence

4. Repeat Step 3, restoring each incremental backup image up to and
including image <ts>
```

If you are using manual incremental restore for a database restore operation, and table space backup images have been produced, the table space images must be restored in the chronological order of their backup time stamps.

If you want to use manual incremental restore, the **db2ckrst** utility can be used to query the database history and generate a list of backup image time stamps needed for an incremental restore. A simplified restore syntax for a manual incremental restore is also generated. It is recommended that you keep a complete record of backups, and only use this utility as a guide.

Automatic Incremental Restore Example

To restore a set of incremental backup images using automatic incremental restore, specify the TAKEN AT timestamp option on the RESTORE DATABASE command. Use the time stamp for the last image that you want to restore. For example:

```
   db2 restore db sample incremental automatic taken at 20001228152133
```

This will result in the DB2 restore utility performing each of the steps described at the beginning of this section automatically. During the initial phase of processing, the backup image with time stamp 20001228152133 is read, and the restore utility verifies that the database, its history, and the table space definitions exist and are valid.

During the second phase of processing, the database history is queried to build a chain of backup images required to perform the requested restore operation. If, for some reason this is not possible, and DB2 is unable to build a complete chain of required images, the restore operation terminates, and an error message is returned. In this case, an automatic incremental restore will not be possible, and you will have issue the RESTORE DATABASE command with the INCREMENTAL ABORT option. This will cleanup any remaining resources so that you can proceed with a manual incremental restore.

Note: It is highly recommended that you not use the FORCE option of the PRUNE HISTORY command. The default operation of this command prevents you from deleting history entries that may be required for recovery from the most recent, full database backup image, but with the FORCE option, it is possible to delete entries that are required for an automatic restore operation.

During the third phase of processing, DB2 will restore each of the remaining backup images in the generated chain. If an error occurs during this phase, you will have to issue the RESTORE DATABASE command with the INCREMENTAL ABORT option to cleanup any remaining resources. You will then have to determine if the error can be resolved before you re-issue the RESTORE command or attempt the manual incremental restore again.

Related concepts:

- "Incremental Backup and Recovery" on page 28

Related reference:

- "RESTORE DATABASE" on page 95
- "db2ckrst - Check Incremental Restore Image Sequence" on page 216

Limitations to Automatic Incremental Restore

1. If a table space name has been changed since the backup operation you want to restore from, and you use the new name when you issue a table

level restore operation, the required chain of backup images from the database history will not be generated correctly and an error will occur (SQL2571N).

Example:

```
db2 backup db sample -> <ts1>
db2 backup db sample incremental -> <ts2>
db2 rename tablespace from userspace1 to t1
db2 restore db sample tablespace ('t1') incremental automatic taken
at <ts2>

SQL2571N Automatic incremental restore is unable to proceed.
Reason code: "3".
```

Suggested workaround: Use manual incremental restore.

2. If you drop a database, the database history will be deleted. If you restore the dropped database, the database history will be restored to its state at the time of the restored backup and all history entries after that time will be lost. If you then attempt to perform an automatic incremental restore that would need to use any of these lost history entries, the RESTORE utility will attempt to restore an incorrect chain of backups and will return an ″out of sequence″ error (SQL2572N).

   Example:

   ```
   db2 backup db sample -> <ts1>
   db2 backup db sample incremental -> <ts2>
   db2 backup db sample incremental delta -> <ts3>
   db2 backup db sample incremental delta -> <ts4>
   db2 drop db sample
   db2 restore db sample incremental automatic taken at <ts2>
   db2 restore db sample incremental automatic taken at <ts4>
   ```

   Suggested workarounds:

   - Use manual incremental restore.
   - Restore the history file first from image <ts4> before issuing an automatic incremental restore.

3. If you restore a backup image from one database into another database and then do an incremental (delta) backup, you can no longer use automatic incremental restore to restore this backup image.

   Example:

   ```
   db2 create db a
   db2 create db b

   db2 update db cfg for a using trackmod on

   db2 backup db a -> ts1
   db2 restore db a taken at ts1 into b

   db2 backup db b incremental -> ts2
   ```

```
db2 restore db b incremental automatic taken at ts2

SQL2542N No match for a database image file was found based on the source
database alias "B" and timestamp "ts1" provided.
```

Suggested workaround:

- Use manual incremental restore as follows:

  ```
  db2 restore db b incremental taken at ts2
  db2 restore db a incremental taken at ts1 into b
  db2 restore db b incremental taken at ts2
  ```

- After the manual restore operation into database B, issue a full database backup to start a new incremental chain

Related concepts:

- "Incremental Backup and Recovery" on page 28

Related tasks:

- "Restoring from Incremental Backup Images" on page 30

Related reference:

- "RESTORE DATABASE" on page 95

Understanding Recovery Logs

All databases have logs associated with them. These logs keep records of database changes. If a database needs to be restored to a point beyond the last full, offline backup, logs are required to roll the data forward to the point of failure.

There are two types of DB2® logging: *circular*, and *archive*, each provides a different level of recovery capability:

- *Circular* logging is the default behavior when a new database is created. (The *logretain* and *userexit database configuration parameters are set to* NO.) With this type of logging, only full, offline backups of the database are allowed. The database must be offline (inaccessible to users) when a full backup is taken. As the name suggests, circular logging uses a "ring" of online logs to provide recovery from transaction failures and system crashes. The logs are used and retained only to the point of ensuring the integrity of current transactions. Circular logging does not allow you to roll a database forward through transactions performed after the last full backup operation. All changes occurring since the last backup operation are lost. Since this type of restore operation recovers your data to the specific point in time at which a full backup was taken, it is called *version recovery*.

Figure 6 shows that the active log uses a ring of log files when circular logging is active.

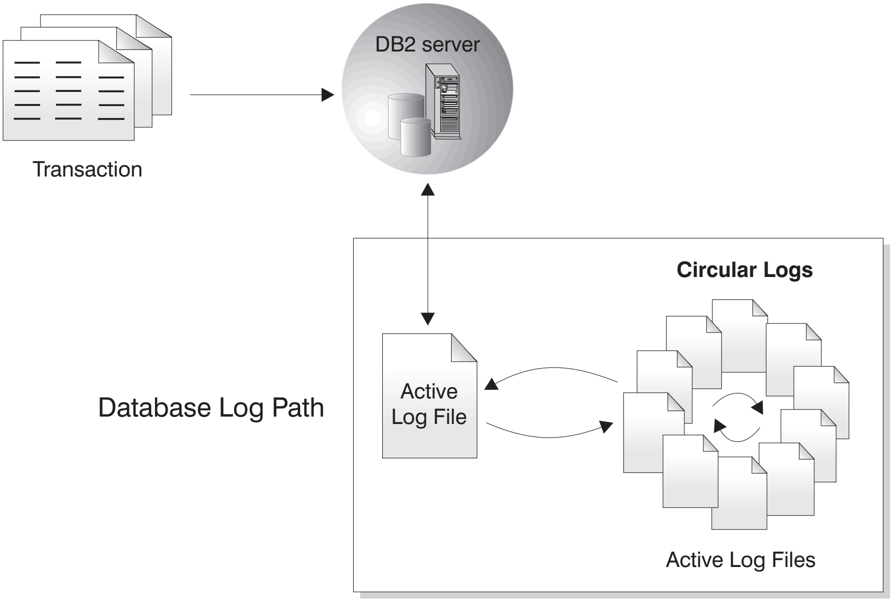


Figure 6. Circular Logging

Active logs are used during crash recovery to prevent a failure (system power or application error) from leaving a database in an inconsistent state. The RESTART DATABASE command uses the active logs, if needed, to move the database to a consistent and usable state. During crash recovery, yet uncommitted changes recorded in these logs are rolled back. Changes that were committed but not yet written from memory (the buffer pool) to disk (database containers) are redone. These actions ensure the integrity of the database. Active logs are located in the database log path directory.

- *Archive* logging is used specifically for rollforward recovery. Enabling the *logretain* and/or the *userexit* database configuration parameter will result in archive logging. To archive logs, you can choose to have DB2 leave the log files in the active path and then manually archive them, or you can install a user exit program to automate the archiving. Archived logs are logs that were active but are no longer required for crash recovery.

The advantage of choosing archive logging is that rollforward recovery can use both archived logs and active logs to rebuild a database either to the end of the logs, or to a specific point in time. The archived log files can be used to recover changes made after the backup was taken. This is different from circular logging where you can only recover to the time of the backup, and all changes made after that are lost.

Taking online backups is only supported if the database is configured for archive logging. During an online backup operation, all activities against the database are logged. When an online backup image is restored, the logs must be rolled forward at least to the point in time at which the backup operation completed. For this to happen, the logs must have been archived and made available when the database is restored. After an online backup is complete, DB2 forces the currently active log to be closed, and as a result, it will be archived. This ensures that your online backup has a complete set of archived logs available for recovery.

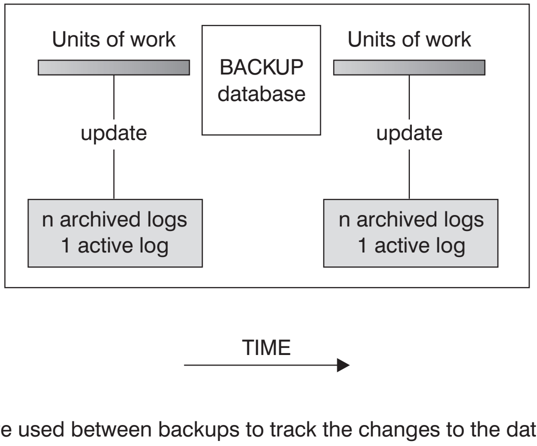

Units of work
BACKUP database
Units of work
update
update
n archived logs 1 active log
n archived logs 1 active log
TIME
Logs are used between backups to track the changes to the databases.

Figure 7. Active and Archived Database Logs in Rollforward Recovery. There can be more than one active log in the case of a long-running transaction.

Two database configuration parameters allow you to change where archived logs are stored: The *newlogpath* parameter, and the *userexit* parameter. Changing the *newlogpath* parameter also affects where active logs are stored.

To determine which log *extents* in the database log path directory are archived logs, check the value of the *loghead* database configuration parameter. This parameter indicates the lowest numbered log that is active. Those logs with sequence numbers less than *loghead* are archived logs and can be moved. You can check the value of this parameter by using the Control Center; or, by using the command line processor and the GET DATABASE CONFIGURATION command to view the "First active log file". For more information about this configuration parameter, see the *Administration Guide: Performance* book.

Related concepts:

- “Log Mirroring” on page 37

Related reference:

- Appendix G, "User Exit for Database Recovery" on page 323
- "First Active Log File configuration parameter - loghead" in the *Administration Guide: Performance*

Recovery Log Details

Log Mirroring

DB2® supports log mirroring at the database level. Mirroring log files helps protect a database from:

- Accidental deletion of an active log
- Data corruption caused by hardware failure

If you are concerned that your active logs may be damaged (as a result of a disk crash), you should consider using a new DB2 configuration parameter, MIRRORLOGPATH, to specify a secondary path for the database to manage copies of the active log, mirroring the volumes on which the logs are stored.

The MIRRORLOGPATH configuration parameter allows the database to write an identical second copy of log files to a different path. It is recommended that you place the secondary log path on a physically separate disk (preferably one that is also on a different disk controller). That way, the disk controller cannot be a single point of failure.

When MIRRORLOGPATH is first enabled, it will not actually be used until the next database startup. This is similar to the NEWLOGPATH configuration parameter.

If there is an error writing to either the active log path or the mirror log path, the database will mark the failing path as "bad", write a message to the administration notification log, and write subsequent log records to the remaining "good" log path only. DB2 will not attempt to use the "bad" path again until the current log file is completed. When DB2 needs to open the next log file, it will verify that this path is valid, and if so, will begin to use it. If not, DB2 will not attempt to use the path again until the next log file is accessed for the first time. There is no attempt to synchronize the log paths, but DB2 keeps information about access errors that occur, so that the correct paths are used when log files are archived. If a failure occurs while writing to the remaining "good" path, the database shuts down.

Related reference:

- "Mirror Log Path configuration parameter - mirrorlogpath" in the *Administration Guide: Performance*

Reducing Logging with the NOT LOGGED INITIALLY Parameter

If your application creates and populates work tables from master tables, and you are not concerned about the recoverability of these work tables because they can be easily recreated from the master tables, you may want to create the work tables specifying the NOT LOGGED INITIALLY parameter on the CREATE TABLE statement. The advantage of using the NOT LOGGED INITIALLY parameter is that any changes made on the table (including insert, delete, update, or create index operations) in the same unit of work that creates the table will not be logged. This not only reduces the logging that is done, but may also increase the performance of your application. You can achieve the same result for existing tables by using the ALTER TABLE statement with the NOT LOGGED INITIALLY parameter.

Notes:

1. You can create more than one table with the NOT LOGGED INITIALLY parameter in the same unit of work.
2. Changes to the catalog tables and other user tables are still logged.

Because changes to the table are not logged, you should consider the following when deciding to use the NOT LOGGED INITIALLY table attribute:

- *All* changes to the table will be flushed out to disk at commit time. This means that the commit may take longer.
- If the NOT LOGGED INITIALLY attribute is activated and an activity occurs that is not logged, the entire unit of work will be rolled back if a statement fails or a ROLLBACK TO SAVEPOINT is executed (SQL1476N).
- You cannot recover these tables when rolling forward. If the rollforward operation encounters a table that was created or altered with the NOT LOGGED INITIALLY option, the table is marked as unavailable. After the database is recovered, any attempt to access the table returns SQL1477N.

  Note: When a table is created, row locks are held on the catalog tables until a COMMIT is done. To take advantage of the no logging behavior, you must populate the table in the same unit of work in which it is created. This has implications for concurrency.

Reducing Logging with Declared Temporary Tables

If you plan to use declared temporary tables as work tables, note the following:

- Declared temporary tables are not created in the catalogs; therefore locks are not held.
- Logging is not performed against declared temporary tables, even after the first COMMIT.
- Use the ON COMMIT PRESERVE option to keep the rows in the table after a COMMIT; otherwise, all rows will be deleted.

- Only the application that creates the declared temporary table can access that instance of the table.
- The table is implicitly dropped when the application connection to the database is dropped.
- Errors in operation during a unit of work using a declared temporary table do not cause the unit of work to be completely rolled back. However, an error in operation in a statement changing the contents of a declared temporary table will delete all the rows in that table. A rollback of the unit of work (or a savepoint) will delete all rows in declared temporary tables that were modified in that unit of work (or savepoint).

Related concepts:

- "Application processes, concurrency, and recovery" in the *SQL Reference, Volume 1*

Related tasks:

- "Creating a table space" in the *Administration Guide: Implementation*

Related reference:

- "DECLARE GLOBAL TEMPORARY TABLE statement" in the *SQL Reference, Volume 2*

Configuration Parameters for Database Logging

Primary logs (logprimary)

This parameter specifies the number of primary logs of size *logfilsz* that will be created.

A primary log, whether empty or full, requires the same amount of disk space. Thus, if you configure more logs than you need, you use disk space unnecessarily. If you configure too few logs, you can encounter a log-full condition. As you select the number of logs to configure, you must consider the size you make each log and whether your application can handle a log-full condition. The total log file size limit on active log space is 256 GB.

If you are enabling an existing database for rollforward recovery, change the number of primary logs to the sum of the number of primary and secondary logs, plus 1. Additional information is logged for LONG VARCHAR and LOB fields in a database enabled for rollforward recovery.

Secondary logs (logsecond)

This parameter specifies the number of secondary log files that are created and used for recovery, if needed.

If the primary log files become full, secondary log files (of size *logfilsiz*) are allocated, one at a time as needed, up to the maximum number specified by this parameter. If this parameter is set to -1, the database is configured with infinite active log space. There is no limit on the size or number of in-flight transactions running on the database.

Notes:

1. The *userexit* database configuration parameter must be enabled in order to set *logsecond* parameter to -1.
2. If this parameter is set to -1, crash recovery time may be increased since DB2 may need to retrieve archived log files.

Log file size (logfilsiz)

This parameter specifies the size of each configured log, in number of 4-KB pages.

There is a 256-GB logical limit on the total active log space that you can configure. This limit is the result of the upper limit on *logfilsiz*, which is 262144, and the upper limit on (*logprimary* + *logsecond*), which is 256.

The size of the log file has a direct bearing on performance. There is a performance cost for switching from one log to another. So, from a pure performance perspective, the larger the log file size the better. This parameter also indicates the log file size for archiving. In this case, a larger log file is size it not necessarily better, since a larger log file size may increase the chance of failure or cause a delay in log shipping scenarios. When considering active log space, it may be better to have a larger number of smaller log files. For example, if there are 2 very large log files and a transaction starts close to the end of one log file, only half of the log space remains available.

Every time a database is deactivated (all connections to the database are terminated), the log file that is currently being written is truncated. So, if a database is frequently being deactivated, it is better not to choose a large log file size because DB2 will create a large file only to have it truncated. You can use the ACTIVATE DATABASE command to avoid this cost, and having the buffer pool primed will also help with performance.

Assuming that you have an application that keeps the database open to minimize processing time when opening the database, the log file size should be determined by the amount of time it takes to make offline archived log copies.

Minimizing log file loss is also an important consideration when setting the log size. Archiving takes an entire log. If you use a single large log, you increase the time between archiving. If the medium containing the log fails, some transaction information will probably be lost. Decreasing the log size increases the frequency of archiving but can reduce the amount of information loss in case of a media failure since the smaller logs before the one lost can be used.

Log Buffer (logbufsz)

This parameter allows you to specify the amount of memory to use as a buffer for log records before writing these records to disk. The log records are written to disk when any one of the following events occurs:

- A transaction commits
- The log buffer becomes full
- Some other internal database manager event occurs.

Increasing the log buffer size results in more efficient input/output (I/O) activity associated with logging, because the log records are written to disk less frequently, and more records are written each time.

Number of Commits to Group (mincommit)

This parameter allows you to delay the writing of log records to disk until a minimum number of commits have been performed. This delay can help reduce the database manager overhead associated with writing log records and, as a result, improve performance when you have multiple applications running against a database, and many commits are requested by the applications within a very short period of time.

The grouping of commits occurs only if the value of this parameter is greater than 1, and if the number of applications connected to the database is greater than the value of this parameter. When commit grouping is in effect, application commit requests are held until either one second has elapsed, or the number of commit requests equals the value of this parameter.

New log path (newlogpath)

The database logs are initially created in SQLOGDIR, which is a subdirectory of the database directory. You can change the location in which active logs and future archived logs are placed by changing the value of this configuration parameter to point to a different directory or to a device. Active logs that are currently stored in the database log path directory are not moved to the new location if the database is configured for rollforward recovery.

Because you can change the log path location, the logs needed for rollforward recovery may exist in different directories or on different devices. You can change the value of this configuration parameter during a rollforward operation to allow you to access logs in multiple locations.

You must keep track of the location of the logs.

Changes are not applied until the database is in a consistent state. The configuration parameter *database_consistent* returns the status of the database.

Mirror log path (mirrorlogpath)

To protect the logs on the primary log path from disk failure or accidental deletion, you can specify that an identical set of logs be maintained on a secondary (mirror) log path. To do this, change the value of this configuration parameter to point to a different directory. Active logs that are currently stored in the mirrored log path directory are not moved to the new location if the database is configured for rollforward recovery.

Because you can change the log path location, the logs needed for rollforward recovery may exist in different directories. You can change the value of this configuration parameter during a rollforward operation to allow you to access logs in multiple locations.

You must keep track of the location of the logs.

Changes are not applied until the database is in a consistent state. The configuration parameter *database_consistent* returns the status of the database.

To turn this configuration parameter off, set its value to DEFAULT.

Notes:

1. This configuration parameter is not supported if the primary log path is a raw device.
2. The value specified for this parameter cannot be a raw device.

Log retain (logretain)

If logretain is set to `RECOVERY`, archived logs are kept in the database log path directory, and the database is considered to be recoverable, meaning that rollforward recovery is enabled.

User exit (userexit)

This parameter causes the database manager to call a user exit program for archiving and retrieving logs. The log files are archived in a location that is different from the active log path. If *userexit* is set to `ON`, rollforward recovery is enabled.

The data transfer speed of the device you use to store offline archived logs, and the software used to make the copies, must at a minimum match the average rate at which the database manager writes data in the logs. If the transfer speed cannot keep up with new log data being generated, you may run out of disk space if logging activity continues for a sufficiently long period of time. The amount of time it takes to run out of disk space is determined by the amount of free disk space. If this happens, database processing stops.

The data transfer speed is most significant when using tape or an optical medium. Some tape devices require the same amount of time to copy a file, regardless of its size. You must determine the capabilities of your archiving device.

Tape devices have other considerations. The frequency of the archiving request is important. For example, if the time taken to complete any copy operation is five minutes, the log should be large enough to hold five minutes of log data during your peak work load. The tape device may have design limits that restrict the number of operations per day. These factors must be considered when you determine the log size.

Note: This value must be set to `ON` to enable infinite active log space.

Note: The default values for the *logretain* and *userexit* database configuration parameters do not support rollforward recovery, and must be changed if you are going to use them.

Overflow log path (overflowlogpath)

This parameter can be used for several functions, depending on your logging requirements. You can specify a location for DB2 to find log files that are needed for a rollforward operation. It is similar to the OVERFLOW LOG PATH option of the ROLLFORWARD command; however, instead of specifying the OVERFLOW LOG PATH option for every ROLLFORWARD command issued, you can set this configuration parameter once. If both are used, the OVERFLOW LOG PATH option will overwrite the *overflowlogpath* configuration parameter for that rollforward operation.

If *logsecond* is set to -1, you can specify a directory for DB2 to store active log files retrieved from the archive. (Active log files must be retrieved for rollback operations if they are no longer in the active log path).

If *overflowlogpath* is not specified, DB2 will retrieve the log files into the active log path. By specifying this parameter you can provide additional resource for DB2 to store the retrieved log files. The benefit includes spreading the I/O cost to different disks, and allowing more log files to be stored in the active log path.

For example, if you are using the **db2ReadLog** API for replication, you can use *overflowlogpath* to specify a location for DB2 to search for log files that are needed for this API. If the log file is not found (in either the active log path or the overflow log path) and the database is configured with *userexit* enabled, DB2 will retrieve the log file. You can also use this parameter to specify a directory for DB2 to store the retrieved log files. The benefit comes from reducing the I/O cost on the active log path and allowing more log files to be stored in the active log path.

If you have configured a raw device for the active log path, *overflowlogpath* must be configured if you want to set *logsecond* to -1, or if you want to use the **db2ReadLog** API.

To set *overflowlogpath*, specify a string of up to 242 bytes. The string must point to a path name, and it must be a fully qualified path name, not a relative path name. The path name must be a directory, not a raw device.

Note: In a partitioned database environment, the node number is automatically appended to the path. This is done to maintain the uniqueness of the path in multiple logical node configurations.

Block on log disk full (blk_log_dsk_ful)

This configuration parameter can be set to prevent disk full errors from being generated when DB2 cannot create a new log file in the active log path. Instead, DB2 will attempt to create the log file every five minutes until it succeeds. After each attempt, DB2 will write a message to the administration notification log. The only way to confirm that your application is hanging because of a log disk full condition is to monitor the administration notification log. Until the log file is successfully created, any user application that attempts to update table data will not be able to commit transactions. Read-only queries may not be directly affected; however, if a query needs to access data that is locked by an update request or a data page that is fixed in the buffer pool by the updating application, read-only queries will also appear to hang.

Setting *blk_log_dsk_ful* to `YES` causes applications to hang when DB2 encounters a log disk full error. You are then able to resolve the error and the transaction can continue. A disk full situation can be resolved by moving old log files to another file system, or by increasing the size of the file system so that hanging applications can complete.

If blk_log_dsk_ful is set to `NO`, a transaction that receives a log disk full error will fail and be rolled back. In some cases, the database will come down if a transaction causes a log disk full error.

Related concepts:

- “Managing Log Files” on page 45
- “Enhancing Recovery Performance” on page 60

Related reference:

- Appendix G, "User Exit for Database Recovery" on page 323
- "Number of Secondary Log Files configuration parameter - logsecond" in the *Administration Guide: Performance*
- "Change the Database Log Path configuration parameter - newlogpath" in the *Administration Guide: Performance*

Managing Log Files

Consider the following when managing database logs:

- The numbering scheme for archived logs starts with S0000000.LOG, and continues through S9999999.LOG, accommodating a potential maximum of 10 million log files. The database manager resets to S0000000.LOG if:
  - A database configuration file is changed to enable rollforward recovery
  - A database configuration file is changed to *disable* rollforward recovery
  - S9999999.LOG has been used.

  DB2® reuses log names after restoring a database (with or without rollforward recovery). The database manager ensures that an incorrect log is not applied during rollforward recovery, but it cannot detect the location of the required log. You must ensure that the correct logs are available for rollforward recovery.

  When a rollforward operation completes successfully, the last log that was used is truncated, and logging begins with the next sequential log. Any log in the log path directory with a sequence number greater than the last log used for rollforward recovery is re-used. Any entries in the truncated log following the truncation point are overwritten with zeros. Ensure that you make a copy of the logs before invoking the rollforward utility. (You can invoke a user exit program to copy the logs to another location.)
- If a database has not been activated (by way of the ACTIVATE DATABASE command), DB2 truncates the current log file when all applications have disconnected from the database. The next time an application connects to the database, DB2 starts logging to a new log file. If many small log files are being produced on your system, you may want to consider using the ACTIVATE DATABASE command. This not only saves the overhead of having to initialize the database when applications connect, it also saves the overhead of having to allocate a large log file, truncate it, and then allocate a new large log file.
- An archived log may be associated with two or more different *log sequences* for a database, because log file names are reused (see Figure 8 on page 46). For example, if you want to recover Backup 2, there are two possible log sequences that could be used. If, during full database recovery, you roll forward to a point in time and stop before reaching the end of the logs, you

have created a new log sequence. The two log sequences cannot be combined. If you have an online backup image that spans the first log sequence, you must use this log sequence to complete rollforward recovery.

If you have created a new log sequence after recovery, any table space backup images on the old log sequence are invalid. This is usually recognized at restore time, but the restore utility fails to recognize a table space backup image on an old log sequence if a database restore operation is immediately followed by the table space restore operation. Until the database is actually rolled forward, the log sequence that is to be used is unknown. If the table space is on an old log sequence, it must be “caught” by the table space rollforward operation. A restore operation using an invalid backup image may complete successfully, but the table space rollforward operation for that table space will fail, and the table space will be left in restore pending state.

For example, suppose that a table space-level backup operation, Backup 3, completes between `S0000013.LOG` and `S0000014.LOG` in the top log sequence (see Figure 8). If you want to restore and roll forward using the database-level backup image, Backup 2, you will need to roll forward through `S0000012.LOG`. After this, you could continue to roll forward through either the top log sequence or the (newer) bottom log sequence. If you roll forward through the bottom log sequence, you will not be able to use the table space-level backup image, Backup 3, to perform table space restore and rollforward recovery.

To complete a table space rollforward operation to the end of the logs using the table space-level backup image, Backup 3, you will have to restore the database-level backup image, Backup 2, and then roll forward using the top log sequence. Once the table space-level backup image, Backup 3, has been restored, you can initiate a rollforward operation to the end of the logs.

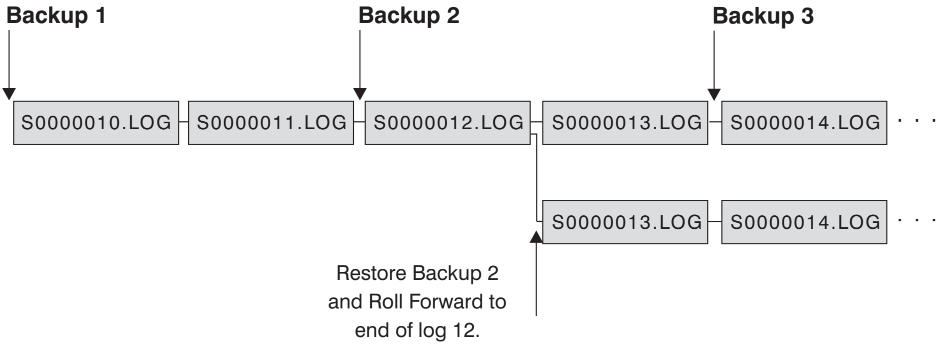


Figure 8. Re-using Log File Names

Related reference:

- Appendix G, “User Exit for Database Recovery” on page 323

Managing Log Files with a User Exit Program

The following considerations apply to calling a user exit program for archiving and retrieving log files:

- The database configuration file parameter *userexit* specifies whether the database manager invokes a user exit program to archive files or to retrieve log files during rollforward recovery of databases. A request to retrieve a log file is made when the rollforward utility needs a log file that is not found in the log path directory.

  Note: On Windows® operating systems, you cannot use a REXX user exit to archive logs.
- When archiving, a log file is passed to the user exit when it is full, even if the log file is still active and is needed for normal processing. This allows copies of the data to be moved away from volatile media as quickly as possible. The log file passed to the user exit is retained in the log path directory until it is no longer needed for normal processing. At this point, the disk space is reused.
- DB2® opens a log file in read mode when it starts a user exit program to archive the file. On some platforms, this prevents the user exit program from being able to delete the log file. Other platforms, like AIX, allow processes, including the user exit program, to delete log files. A user exit program should never delete a log file after it is archived, because the file could still be active and needed for crash recovery. DB2 manages disk space reuse when log files are archived.
- When a log file has been archived and is inactive, DB2 does not delete the file but renames it as the next log file when such a file is needed. This results in a performance gain, because creating a new log file (instead of renaming the file) causes all pages to be written out to guarantee the disk space. It is more efficient to reuse than to free up and then reacquire the necessary pages on disk.
- DB2 will *not* invoke the user exit program to retrieve the log file during crash recovery or rollback unless the *logsecond* database configuration parameter is set to -1.
- A user exit program does not guarantee rollforward recovery to the point of failure, but only attempts to make the failure window smaller. As log files fill, they are queued for the user exit routine. Should the disk containing the log fail before a log file is filled, the data in that log file is lost. Also, since the files are queued for archiving, the disk can fail before all the files are copied, causing any log files in the queue to be lost.
- The configured size of each individual log file has a direct bearing on the user exit. If each log file is very large, a large amount of data can be lost if a disk fails. A database configured with small log files causes the data to be passed to the user exit routine more often.

However, if you are moving the data to a slower device such as tape, you might want to have larger log files to prevent the queue from building up. If the queue becomes full, archive and retrieve requests will not be processed. Processing will resume when there is room on the queue. Unprocessed requests will not be automatically requeued.

- An archive request to the user exit program occurs only if *userexit* is configured, and each time an active log file is filled. It is possible that an active log file is not full when the last disconnection from the database occurs and the user exit program is also called for a partially filled active log file.

  Note: To free unused log space, the log file is truncated before it is archived.
- A copy of the log should be made to another physical device so that the offline log file can be used by rollforward recovery if the device containing the log file experiences a media failure. This should not be the same device containing database data files.
- If you have enabled user exit programs and are using a tape drive as a storage device for logs and backup images, you need to ensure that the destination for the backup images and the archived logs is not the same tape drive. Since some log archiving may take place while a backup operation is in progress, an error may occur when the two processes are trying to write to the same tape drive at the same time.
- In some cases, if a database is closed before a positive response has been received from a user exit program for an archive request, the database manager will send another request when the database is opened. Thus, a log file may be archived more than once.
- If a user exit program receives a request to archive a file that does not exist (because there were multiple requests to archive and the file was deleted after the first successful archiving operation), or to retrieve a file that does not exist (because it is located in another directory or the end of the logs has been reached), it should ignore this request and pass a successful return code.
- The user exit program should allow for the existence of different log files with the same name after a point in time recovery; it should be written to preserve both log files and to associate those log files with the correct recovery path.
- If a user exit program is enabled for two or more databases that are using the same tape device to archive log files, and a rollforward operation is taking place on one of the databases, the other database(s) should not be active. If another database tries to archive a log file while the rollforward operation is in progress, the logs required for the rollforward operation may not be found or the new log file archived to the tape device might overwrite the log files previously stored on that tape device.

To prevent either situation from occurring, you can ensure that no other databases on the node that calls the user exit program are open during the rollforward operation, or write a user exit program to handle this situation.

Related concepts:

- "Managing Log Files" on page 45

Log File Allocation and Removal

Log files in the database log directory are never removed if they may be required for crash recovery. When the *userexit* database configuration parameter is enabled, a full log file becomes a candidate for removal only after it is no longer required for crash recovery. A log file which is required for crash recovery is called an active log. A log file which is not required for crash recovery is called an archived log.

The process of allocating new log files and removing old log files is dependent on the settings of *userexit* and *logretain* database configuration parameters:

Both *logretain* and *userexit* are set to `OFF`

Circular logging will be used. Rollforward recovery is not supported with circular logging, while crash recovery is.

During circular logging, new log files, other than secondary logs, are not generated and old log files are not deleted. Log files are handled in a circular fashion. That is, when the last log file is full, DB2® begins writing to the first log file.

A log full situation can occur if all of the log files are active and the circular logging process cannot wrap to the first log file. Secondary log files are created when all the primary log files are active and full. Once a secondary log is created, it is not deleted until the database is restarted.

***Logretain* is set to `ON` and *userexit* is set to `OFF`**

Both rollforward recovery and crash recovery are enabled. The database is known to be recoverable. When *userexit* is set to `OFF`, DB2 does not delete log files from the database log directory. Each time a log file becomes full, DB2 begins writing records to another log file, and (if the maximum number of primary and secondary logs has not been reached) creates a new log file.

***Userexit* is set to `ON`**

When both *logretain* and *userexit* are set to `on`, both rollforward recovery and crash recovery are enabled. When a log file becomes full, it is automatically archived using the user supplied user exit program.

Log files are usually not deleted. Instead, when a new log file is required and one is not available, an archived log file is renamed and used again. An archived log file, is not deleted or renamed once it has been closed and copied to the log archive directory. DB2 waits until a new log file is needed and then renames the oldest archived log. A log file that has been moved to the database directory during recovery is removed during the recovery process when it is no longer needed. Until DB2 runs out of log space, you will see old log files in the database directory.

If an error is encountered while archiving a log file, archiving of log files will be suspended for five minutes before being attempted again. DB2 will then continue archiving log files as they become full. Log files that became full during the five minute waiting period will not be archived immediately after the delay, DB2 will spread the archive of these files over time.

The easiest way to remove old log files is to restart the database. Once the database is restarted, only new log files and log files that the user exit program failed to archive will be found in the database directory.

When a database is restarted, the minimum number of logs in the database log directory will equal the number of primary logs which can be configured using the *logprimary* database configuration parameter. It is possible for more than the number of primary logs to be found in the log directory. This can occur if the number of empty logs in the log directory at the time the database was shut down, is greater than the value of the *logprimary* configuration parameter at the time the database is restarted. This will happen if the value of the *logprimary* configuration parameter is changed between the database being shut down and restarted, or if secondary logs are allocated and never used.

When a database is restarted, if the number of empty logs is less than the number of primary logs specified by the *logprimary* configuration parameter, additional log files will be allocated to make up the difference. If there are more empty logs than primary logs available in the database directory, the database can be restarted with as many available empty logs as are found in the database directory. After database shutdown, secondary log files that have been created will remain in the active log path when the database is restarted.

Blocking Transactions When the Log Directory File is Full

The *blk_log_dsk_ful* database configuration parameter can be set to prevent ″disk full″ errors from being generated when DB2® cannot create a new log file in the active log path.

Instead, DB2 attempts to create the log file every five minutes until it succeeds. If the database is configured with the *userexit* parameter set to `ON`, DB2 also checks for the completion of log file archiving. If an archived log file is archived successfully, DB2 can rename the inactive log file to the new log file name and continue. After each attempt, DB2 writes a message to the administration notification log. The only way that you can confirm that your application is hanging because of a log disk full condition is to monitor the administration notification log.

Until the log file is successfully created, any user application that attempts to update table data will not able to commit transactions. Read-only queries may not be directly affected; however, if a query needs to access data that is locked by an update request, or a data page that is fixed in the buffer pool by the updating application, read-only queries will also appear to hang.

Related concepts:

- "Understanding Recovery Logs" on page 34
- "Managing Log Files with a User Exit Program" on page 47

On Demand Log Archive

DB2® now supports the closing (and, if the user exit option is enabled, the archiving) of the active log for a recoverable database at any time. This allows you to collect a complete set of log files up to a known point, and then to use these log files to update a standby database.

You can initiate on demand log archiving by invoking the ARCHIVE LOG command, or by calling the **db2ArchiveLog** API.

Related reference:

- "ARCHIVE LOG" on page 225
- "db2ArchiveLog - Archive Active Log" in the *Administrative API Reference*

Using Raw Logs

You can use a raw device for your database log. There are both advantages and disadvantages in doing so.

- The advantages are:
  - You can attach more than 26 physical drives to a system.
  - The file I/O path length is shorter. This may improve performance on your system. You should conduct benchmarks to evaluate if there are measurable benefits for your work load.
- The disadvantages are:
  - The device cannot be shared by other applications; the entire device *must* be assigned to DB2.

  - The device cannot be operated upon by any operating system utility or third-party tool which would backup or copy from the device.
  - You can easily wipe out the file system on an existing drive if you specify the wrong physical drive number.

You can configure a raw log with the *newlogpath* database configuration parameter. Before doing so, however, consider the advantages and disadvantages listed above, and the additional considerations listed below:

- Only one device is allowed. You can define the device over multiple disks at the operating system level. DB2® will make an operating system call to determine the size of the device in 4-KB pages.

  If you use multiple disks, this will provide a larger device, and the striping that results can improve performance by faster I/O throughput.
- DB2 will attempt to write to the last 4-KB page of the device. If the device size is greater than 2 GB, the attempt to write to the last page will fail on operating systems that do not provide support for devices larger than 2 GB. In this situation, DB2 will attempt to use all pages, up to the supported limit.

  Information about the size of the device is used to indicate the size of the device (in 4-KB pages) available to DB2 under the support of the operating system. The amount of disk space that DB2 can write to is referred to as the *device-size-available*.

  The first 4-KB page of the device is not used by DB2 (this space is generally used by operating system for other purposes.) This means that the total space available to DB2 is *device-size = device-size-available - 1*.
- The *logsecond* parameter is not used. DB2 will not allocate secondary logs. The size of active log space is the number of 4-KB pages that result from *logprimary* x *logfilsiz*.
- Log records are still grouped into log extents, each with a log file size (*logfilsiz*) of 4-KB pages. Log extents are placed in the raw device, one after another. Each extent also consists of an extra two pages for the extent header. This means that the *number of available log extents* the device can support is *device-size* / (*logfilsiz* + 2)
- The device must be large enough to support the active log space. That is, the *number of available log extents* must be greater than (or equal to) the value specified for the *logprimary* configuration parameter. If the *userexit* configuration parameter is enabled, ensure that the raw device can contain more logs than the value specified for the *logprimary* configuration parameter. This will compensate for the delay incurred when the user exit program is archiving a log file.
- If you are using circular logging, the *logprimary* configuration parameter will determine the number of log extents that are written to the device. This may result in unused space on the device.

- If you are using log retention (*logretain*) without a user exit program, after the *number of available log extents* are all used up, all operations that result in an update will receive a log full error. At this time, you must shut down the database and take an offline backup of it to ensure recoverability. After the database backup operation, the log records written to the device are lost. This means that you cannot use an earlier database backup image to restore the database, then roll it forward. If you take a database backup before the *number of available log extents* are all used up, you can restore and roll the database forward.
- If you are using log retention (*logretain*) with a user exit program, the user exit program is called for each log extent as it is filled with log records. The user exit program must be able to read the device, and to store the archived log as a file. DB2 will not call a user exit program to retrieve log files to a raw device. Instead, during rollforward recovery, DB2 will read the extent headers to determine if the raw device contains the required log file. If the required log file is not found in the raw device, DB2 will search the overflow log path. If the log file is still not found, DB2 will call the user exit program to retrieve the log file into the overflow log path. If you do not specify an overflow log path for the rollforward operation, DB2 will not call the user exit program to retrieve the log file.
- If you have configured a raw device for logging, and are using DataPropagator™ (DPROP), or another application that calls the **db2ReadLog** API, the *overflowlogpath* database configuration parameter must be configured. DB2 may call a user exit program to retrieve the log file and return the log data requested by the **db2ReadLog** API. The retrieved log file will be placed in the path specified by the *overflowlogpath* database configuration parameter.

Related tasks:

- "Specifying raw I/O" in the *Administration Guide: Implementation*

Related reference:

- "db2ReadLog - Asynchronous Read Log" on page 263
- Appendix F, "Tivoli Storage Manager" on page 319

How to Prevent Losing Log Files

In situations where you need to drop a database or perform a point-in-time rollforward recovery, it is possible to lose log files that may be required for future recovery operations. In these cases, it is important to make copies of all the logs in the current database log path directory. Consider the following scenarios:

- If you plan to drop a database prior to a restore operation, you need to save the log files in the active log path before issuing the DROP DATABASE command. After the database has been restored, these log files may be required for rollforward recovery because some of them may not have been archived before the database was dropped. Normally, you are not required to drop a database prior to issuing the RESTORE command. However, you may have to drop the database (or drop the database on one partition by specifying the AT NODE option of DROP DATABASE command), because it has been damaged to the extent that the RESTORE command fails. You may also decide to drop a database prior to the restore operation to give yourself a fresh start.

- If you are rolling a database forward to a specific point in time, log data after the time stamp you specify will be overwritten. If, after you have completed the point-in-time rollforward operation and reconnected to the database, you determine that you actually needed to roll the database forward to a later point in time, you will not be able to because the logs will already have been overwritten. It is possible that the original set of log files may have been archived; however, DB2® may be calling a user exit program to automatically archive the newly generated log files. Depending on how the user exit program is written, this could cause the original set of log files in the archive log directory to be overwritten. Even if both the original and new set of log files exist in the archive log directory (as different versions of the same files), you may have to determine which set of logs should be used for future recovery operations.

Related concepts:

- "Understanding Recovery Logs" on page 34

Understanding the Recovery History File

A recovery history file is created with each database and is automatically updated whenever:

- A database or table spaces are backed up
- A database or table spaces are restored
- A database or table spaces are rolled forward
- A table space is created
- A table space is altered
- A table space is quiesced
- A table space is renamed
- A table space is dropped
- A table is loaded
- A table is dropped
- A table is reorganized

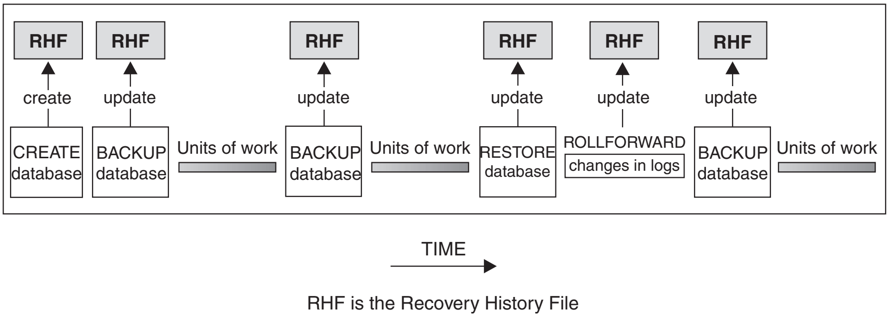


Figure 9. Creating and Updating the Recovery History File

You can use the summarized backup information in this file to recover all or part of a database to a given point in time. The information in the file includes:

- An identification (ID) field to uniquely identify each entry
- The part of the database that was copied and how
- The time the copy was made
- The location of the copy (stating both the device information and the logical way to access the copy)
- The last time a restore operation was done
- The time at which a table space was renamed, showing the previous and the current name of the table space
- The status of a backup operation: active, inactive, expired, or deleted
- The last log sequence number saved by the database backup or processed during a rollforward recovery operation.

To see the entries in the recovery history file, use the LIST HISTORY command.

Every backup operation (database, table space, or incremental) includes a copy of the recovery history file. The recovery history file is linked to the database. Dropping a database deletes the recovery history file. Restoring a database to a new location restores the recovery history file. Restoring does not overwrite the existing history recovery file unless the file that exists on disk has no entries. If that is the case, the database history will be restored from the backup image.

If the current database is unusable or not available, and the associated recovery history file is damaged or deleted, an option on the RESTORE command allows only the recovery history file to be restored. The recovery history file can then be reviewed to provide information on which backup to use to restore the database.

The size of the file is controlled by the *rec_his_retentn* configuration parameter that specifies a retention period (in days) for the entries in the file. Even if the number for this parameter is set to zero (0), the most recent full database backup (plus its restore set) is kept. (The only way to remove this copy is to use the PRUNE with FORCE option.) The retention period has a default value of 366 days. The period can be set to an indefinite number of days by using -1. In this case, explicit pruning of the file is required.

Related reference:

- "Recovery History Retention Period configuration parameter - rec_his_retentn" in the *Administration Guide: Performance*
- "LIST HISTORY" on page 228

Recovery History File - Garbage Collection

Garbage Collection

Although you can use the PRUNE HISTORY command at any time to remove entries from the history file, it is recommended that such pruning be left to DB2. The number of DB2® database backups recorded in the recovery history file is monitored automatically by DB2 *garbage collection*. DB2 garbage collection is invoked:

- After a full, non-incremental database backup operation completes successfully.
- After a database restore operation, where a rollforward operation is not required, completes successfully.
- After a successful database rollforward operation completes successfully.

The configuration parameter *num_db_backups* defines how many active full (non-incremental) database backup images are kept. The value of this parameter is used to scan the history file, starting with the last entry.

After every full (non-incremental) database backup operation, the *rec_his_retentn* configuration parameter is used to prune expired entries from the history file.

An *active database backup* is one that can be restored and rolled forward using the current logs to recover the current state of the database. An *inactive database backup* is one that, if restored, moves the database back to a previous state.

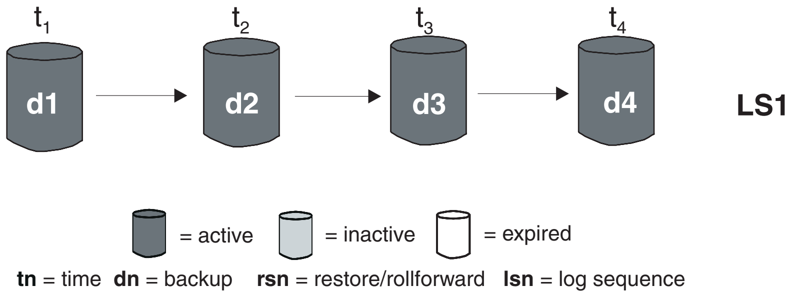

t_1
t_2
t_3
t_4
d1
d2
d3
d4
LS1
= active
= inactive
= expired
tn = time dn = backup rsn = restore/rollforward lsn = log sequence

Figure 10. Active Database Backups. The value of *num_db_backups* has been set to four.

All active database backup images that are no longer needed are marked as “expired”. These images are considered to be unnecessary, because more recent backup images are available. All table space backup images and load backup copies that were taken before the database backup image expired are also marked as “expired”.

All database backup images that are marked as “inactive” and that were taken prior to the point at which an expired database backup was taken are also marked as “expired”. All associated inactive table space backup images and load backup copies are also marked as “expired”.

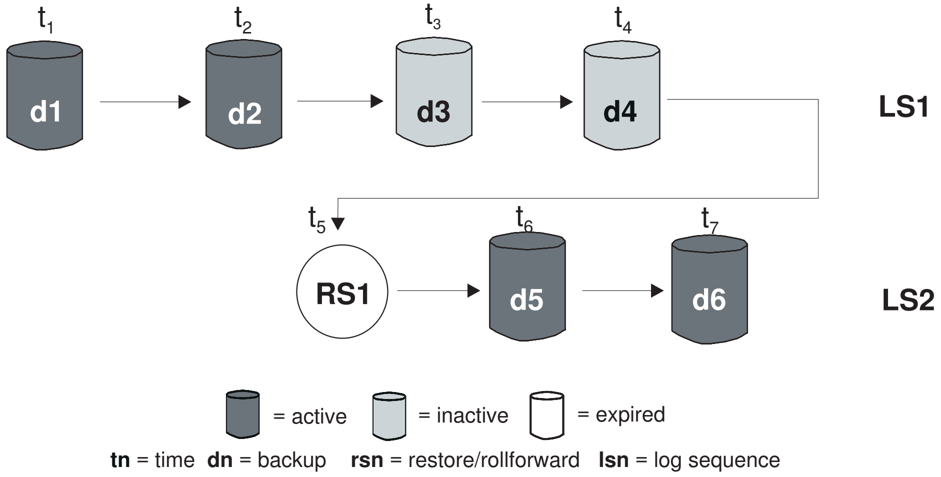


Figure 11. Inactive Database Backups

If an active database backup image is restored, but it is not the most recent database backup recorded in the history file, any subsequent database backup images belonging to the same log sequence are marked as “inactive”.

If an inactive database backup image is restored, any inactive database backups belonging to the current log sequence are marked as “active” again. All active database backup images that are no longer in the current log sequence are marked as “inactive”.

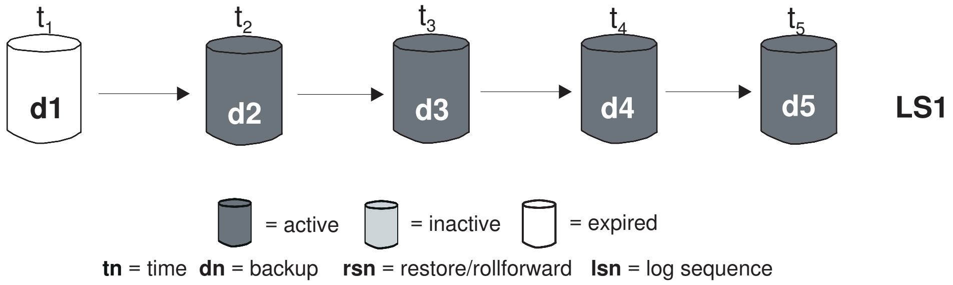

t1
t2
t3
t4
t5
d1
d2
d3
d4
d5
LS1
= active
= inactive
= expired
tn = time dn = backup rsn = restore/rollforward lsn = log sequence

Figure 12. Expired Database Backups

DB2 garbage collection is also responsible for marking the history file entries for a DB2 database or table space backup image as “inactive”, if that backup does not correspond to the current *log sequence*, also called the current *log chain*. The current log sequence is determined by the DB2 database backup image that has been restored, and the log files that have been processed. Once a database backup image is restored, all subsequent database backup images become “inactive”, because the restored image begins a new log chain. (This is true if the backup image was restored without rolling forward. If a rollforward operation has occurred, all database backups that were taken after the break in the log chain are marked as “inactive”. It is conceivable that an older database backup image will have to be restored because the rollforward utility has gone through the log sequence containing a damaged current backup image.)

A table space-level backup image becomes “inactive” if, after it is restored, the current state of the database cannot be reached by applying the current log sequence.

If a backup image contains DATALINK columns, all Data Links servers running the DB2 Data Links Manager are contacted to request garbage collection. DB2 garbage collection then deletes backups of the associated Data Links server files that were contained in the expired backup, but that were unlinked before the next database backup operation.

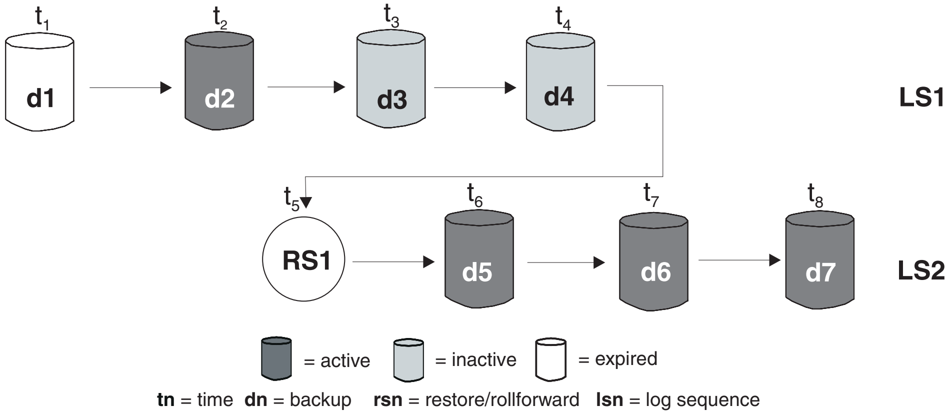

t1
t2
t3
t4
d1
d2
d3
d4
LS1
t5
t6
t7
t8
RS1
d5
d6
d7
LS2
= active
= inactive
= expired
tn = time dn = backup rsn = restore/rollforward lsn = log sequence

Figure 13. Mixed Active, Inactive, and Expired Database Backups

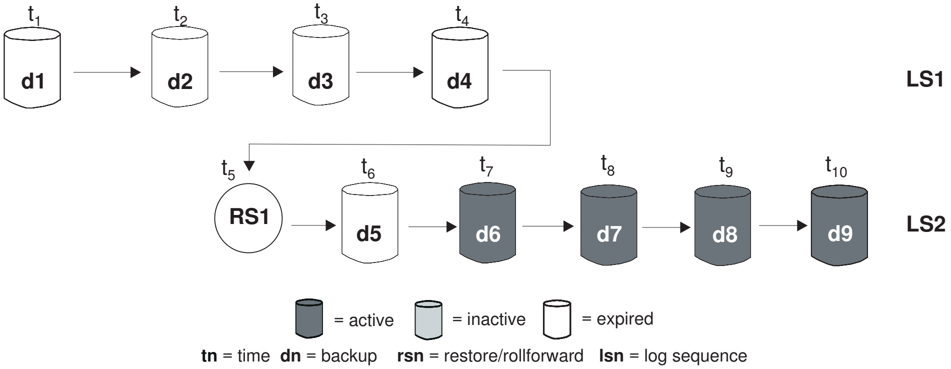


Figure 14. Expired Log Sequence

Related concepts:

- "Understanding the Recovery History File" on page 54

Related reference:

- "PRUNE HISTORY/LOGFILE" on page 231

Understanding Table Space States

The current status of a table space is reflected by its *state*. The table space states most commonly associated with recovery are:

- *Rollforward pending*. A table space is put in this state after it is restored, or following an input/output (I/O) error. After it is restored, the table space can be rolled forward to the end of the logs or to a point in time. Following an I/O error, the table space must be rolled forward to the end of the logs.

- *Rollforward-in-progress*. A table space is put in this state when a rollforward operation on that table space is in progress. Once the rollforward operation completes successfully, the table space is no longer in rollforward-in-progress state. The table space can also be taken out of this state if the rollforward operation is cancelled.
- *Restore pending*. A table space is put in this state if a rollforward operation on that table space is cancelled, or if a rollforward operation on that table space encounters an unrecoverable error, in which case the table space must be restored and rolled forward again.
- *Backup pending*. A table space is put in this state after a point-in-time rollforward operation, or after a load operation with the no copy option. The table space must be backed up before it can be used. (If it is not backed up, the table space cannot be updated, but read only operations are allowed.)

Enhancing Recovery Performance

The following should be considered when thinking about recovery performance:

- You can improve performance for databases that are frequently updated by placing the logs on a separate device. In the case of an online transaction processing (OLTP) environment, often more I/O is needed to write data to the logs than to store a row of data. Placing the logs on a separate device will minimize the disk arm movement that is required to move between a log and the database files.

  You should also consider what other files are on the disk. For example, moving the logs to the disk used for system paging in a system that has insufficient real memory will defeat your tuning efforts.
- To reduce the amount of time required to complete a restore operation:
  - Adjust the restore buffer size. The buffer size must be a multiple of the buffer size that was used during the backup operation.
  - Increase the number of buffers.

    If you use multiple buffers and I/O media devices, you should use at least twice as many buffers as media devices to ensure that they do not have to wait for data. The size of the buffers used will also contribute to the performance of the restore operation. The ideal restore buffer size should be a multiple of the extent size for the table spaces.

    If you have multiple table spaces with different extent sizes, specify a value that is a multiple of the largest extent size.

The *minimum* recommended number of buffers is the number of media devices plus the number specified for the PARALLELISM option.

  - Use multiple source devices.
  - Set the PARALLELISM option for the restore operation to be at least one (1) greater than the number of source devices.

- If a table contains large amounts of long field and LOB data, restoring it could be very time consuming. If the database is enabled for rollforward recovery, the RESTORE command provides the capability to restore selected table spaces. If the long field and LOB data is critical to your business, restoring these table spaces should be considered against the time required to complete the backup task for these table spaces. By storing long field and LOB data in separate table spaces, the time required to complete the restore operation can be reduced by choosing not to restore the table spaces containing the long field and LOB data. If the LOB data can be reproduced from a separate source, choose the NOT LOGGED option when creating or altering a table to include LOB columns. If you choose not to restore the table spaces that contain long field and LOB data, but you need to restore the table spaces that contain the table, you must roll forward to the end of the logs so that all table spaces that contain table data are consistent.

  Note: If you back up a table space that contains table data without the associated long or LOB fields, you cannot perform point-in-time rollforward recovery on that table space. All the table spaces for a table must be rolled forward simultaneously to the same point in time.

- The following apply for both backup and restore operations:
  - Multiple I/O buffers and devices should be used.
  - Allocate at least twice as many buffers as devices being used.
  - Do not overload the I/O device controller bandwidth.
  - Use more buffers of smaller size rather than a few large buffers.
  - Tune the number and the size of the buffers according to the system resources.
  - Use of the PARALLELISM option

Enhancing Recovery Performance - Parallel Recovery

Parallel Recovery

DB2® uses multiple agents to perform both crash recovery and database rollforward recovery. You can expect better performance during these operations, particularly on symmetric multi-processor (SMP) machines; using multiple agents during database recovery takes advantage of the extra CPUs that are available on SMP machines.

The agent type introduced by parallel recovery is db2agnsc. DB2 chooses the number of agents to be used for database recovery based on the number of CPUs on the machine.

DB2 distributes log records to these agents so that they can be reapplied concurrently, where appropriate. For example, the processing of log records associated with insert, delete, update, add key, and delete key operations can be parallelized in this way. Because the log records are parallelized at the page level (log records on the same data page are processed by the same agent), performance is enhanced, even if all the work was done on one table.

Related concepts:

- "Enhancing Recovery Performance" on page 60

Chapter 2. Database Backup

This section describes the DB2 UDB backup utility, which is used to create backup copies of a database or table spaces.

The following topics are covered:

- "Backup Overview"
- "Privileges, Authorities, and Authorization Required to Use Backup" on page 66
- "Using Backup" on page 67
- "Backing Up to Tape" on page 69
- "Backing Up to Named Pipes" on page 71
- "BACKUP DATABASE" on page 72
- "db2Backup - Backup database" on page 77
- "Backup Sessions - CLP Examples" on page 84
- "Optimizing Backup Performance" on page 85

Backup Overview

The simplest form of the DB2® BACKUP DATABASE command requires only that you specify the alias name of the database that you want to back up. For example:

```
   db2 backup db sample
```

If the command completes successfully, you will have acquired a new backup image that is located in the path or the directory from which the command was issued. It is located in this directory because the command in this example does not explicitly specify a target location for the backup image. On Windows® operating systems, for example, this command (when issued from the root directory) creates an image that appears in a directory listing as follows:

```
 Directory of D:\SAMPLE.0\DB2\NODE0000\CATN0000\20010320

03/20/2001  12:26p      <DIR>          .
03/20/2001  12:26p      <DIR>          ..
03/20/2001  12:27p          12,615,680 122644.001
```

Note: If the DB2 client and server are not located on the same system, the default target directory for the backup image is the current working directory on the client system where the command was issued. This target directory or device must exist on the server system.

Backup images are created at the target location that you have the option to specify when you invoke the backup utility. This location can be:

- A directory (for backups to disk or diskette)
- A device (for backups to tape)
- A Tivoli® Storage Manager (TSM) server
- Another vendor's server

The recovery history file is updated automatically with summary information whenever you invoke a database backup operation. This file is created in the same directory as the database configuration file.

On UNIX® based systems, file names for backup images created on disk consist of a concatenation of several elements, separated by periods:

```
DB_alias.Type.Inst_name.NODEnnnn.CATNnnnn.timestamp.Seq_num
```

For example:

```
STAFF.0.DB201.NODE0000.CATN0000.19950922120112.001
```

On Windows operating systems, a four-level subdirectory tree is used:

```
DB_alias.Type\Inst_name\NODEnnnn\CATNnnnn\yyyymmdd\hhmmss.Seq_num
```

For example:

```
SAMPLE.0\DB2\NODE0000\CATN0000\20010320\122644.001
```

Database alias
A 1- to 8-character database alias name that was specified when the backup utility was invoked.

Type
Type of backup operation, where: 0 represents a full database-level backup, 3 represents a table space-level backup, and 4 represents a backup image generated by the LOAD...COPY TO command.

Instance name
A 1- to 8-character name of the current instance that is taken from the **DB2INSTANCE** environment variable.

Node number
The node number. In non-partitioned database systems, this is always NODE0000. In partitioned database systems, it is NODE*xxxx*, where *xxxx* is the number assigned to the node in the db2nodes.cfg file.

Catalog node number
The node number of the catalog node for the database. In non-partitioned database systems, this is always `CATN0000`. In partitioned database systems, it is `CATN`*xxxx*, where *xxxx* is the number assigned to the node in the `db2nodes.cfg` file.

Time stamp
A 14-character representation of the date and time at which the backup operation was performed. The time stamp is in the form *yyyymmddhhnnss*, where:

- *yyyy* represents the year (1995 to 9999)
- *mm* represents the month (01 to 12)
- *dd* represents the day of the month (01 to 31)
- *hh* represents the hour (00 to 23)
- *nn* represents the minutes (00 to 59)
- *ss* represents the seconds (00 to 59)

Sequence number
A 3-digit number used as a file extension.

When a backup image is written to tape:

- File names are not created, but the information described above is stored in the backup header for verification purposes.
- A tape device must be available through the standard operating system interface. On a large partitioned database system, however, it may not be practical to have a tape device dedicated to each database partition server. You can connect the tape devices to one or more TSM servers, so that access to these tape devices is provided to each database partition server.
- On a partitioned database system, you can also use products that provide virtual tape device functions, such as REELlibrarian 4.2 or CLIO/S. You can use these products to access the tape device connected to other nodes (database partition servers) through a pseudo tape device. Access to the remote tape device is provided transparently, and the pseudo tape device can be accessed through the standard operating system interface.

You cannot back up a database that is in an unusable state, except when that database is in backup pending state. If any table space is in an abnormal state, you cannot back up the database or that table space, unless it is in backup pending state.

If a database or a table space is in a partially restored state because a system crash occurred during the restore operation, you must successfully restore the database or the table space before you can back it up.

A backup operation will fail if a list of the table spaces to be backed up contains the name of a temporary table space.

The backup utility provides concurrency control for multiple processes that are making backup copies of different databases. This concurrency control keeps the backup target devices open until all the backup operations have ended. If an error occurs during a backup operation, and an open container cannot be closed, other backup operations targeting the same drive may receive access errors. To correct such access errors, you must terminate the backup operation that caused the error and disconnect from the target device. If you are using the backup utility for concurrent backup operations to tape, ensure that the processes do not target the same tape.

Displaying Backup Information

You can use **db2ckbkp** to display information about existing backup images. This utility allows you to:

- Test the integrity of a backup image and determine whether or not it can be restored.
- Display information that is stored in the backup header.
- Display information about the objects and the log file header in the backup image.

Related concepts:

- "Understanding the Recovery History File" on page 54

Related reference:

- "db2ckbkp - Check Backup" on page 213
- Appendix F, "Tivoli Storage Manager" on page 319

Privileges, Authorities, and Authorization Required to Use Backup

Privileges enable users to create or access database resources. Authority levels provide a method of grouping privileges and higher-level database manager maintenance and utility operations. Together, these act to control access to the database manager and its database objects. Users can access only those objects for which they have the appropriate authorization; that is, the required privilege or authority.

You must have SYSADM, SYSCTRL, or SYSMAINT authority to use the backup utility.

Using Backup

Prerequisites:

You should not be connected to the database that is to be backed up: the backup utility automatically establishes a connection to the specified database, and this connection is terminated at the completion of the backup operation.

The database can be local or remote. The backup image remains on the database server, unless you are using a storage management product such as Tivoli Storage Manager (TSM).

On a partitioned database system, database partitions are backed up individually. The operation is local to the database partition server on which you invoke the utility. You can, however, issue **db2_all** from one of the database partition servers in the instance to invoke the backup utility on a list of servers, which you identify by node number. (Use the LIST NODES command to identify the nodes, or database partition servers, that have user tables on them.) If you do this, you must back up the catalog node first, then back up the other database partitions. You can also use the Command Center to back up database partitions. Because this approach does not support rollforward recovery, back up the database residing on these nodes regularly. You should also keep a copy of the `db2nodes.cfg` file with any backup copies you take, as protection against possible damage to this file.

On a distributed request system, backup operations apply to the distributed request database and the metadata stored in the database catalog (wrappers, servers, nicknames, and so on). Data source objects (tables and views) are not backed up, unless they are stored in the distributed request database.

If a database was created with a previous release of the database manager, and the database has not been migrated, you must migrate the database before you can back it up.

Restrictions:

The following restrictions apply to the backup utility:

- A table space backup operation and a table space restore operation cannot be run at the same time, even if different table spaces are involved.
- If you want to be able to do rollforward recovery in a partitioned database environment, you must regularly back up the database on the list of nodes, and you must have at least one backup image of the rest of the nodes in the system (even those that do not contain user data for that database). Two situations require the backed-up image of a database partition at a database partition server that does not contain user data for the database:

- You added a database partition server to the database system after taking the last backup, and you need to do forward recovery on this database partition server.
- Point-in-time recovery is used, which requires that all database partitions in the system are in rollforward pending state.

Procedure:

The backup utility can be invoked through the command line processor (CLP), the Backup Database notebook or Wizard in the Control Center, or the **db2Backup** application programming interface (API).

Following is an example of the BACKUP DATABASE command issued through the CLP:

```
db2 backup database sample to c:\DB2Backups
```

To open the Backup Database notebook or wizard:

1. From the Control Center, expand the object tree until you find the Databases folder.
2. Click on the Databases folder. Any existing databases are displayed in the pane on the right side of the window (the contents pane).
3. Click the right mouse button on the database you want in the contents pane, and select Backup Database or Backup Database Using Wizard from the pop-up menu. The Backup Database notebook or the Backup Database wizard opens.

Detailed information is provided through the online help facility within the Control Center.

Related concepts:

- "Administrative APIs in Embedded SQL or DB2 CLI Programs" in the *Application Development Guide: Programming Client Applications*
- "Introducing the plug-in architecture for the Control Center" in the *Administration Guide: Implementation*

Related tasks:

- "Migrating databases" in the *Quick Beginnings for DB2 Servers*

Related reference:

- "LIST DBPARTITIONNUMS" in the *Command Reference*
- "db2Backup - Backup database" on page 77

Backing Up to Tape

When you back up your database or table space, you must correctly set your block size and your buffer size. This is particularly true if you are using a variable block size (on AIX, for example, if the block size has been set to zero).

There is a restriction on the number of fixed block sizes that can be used when backing up. This restriction exists because DB2® writes out the backup image header as a 4-KB block. The only fixed block sizes DB2 supports are 512, 1024, 2048, and 4096 bytes. If you are using a fixed block size, you can specify any backup buffer size. However, you may find that your backup operation will not complete successfully if the fixed block size is not one of the sizes that DB2 supports.

If your database is large, using a fixed block size means that your backup operations will take a long time. You may want to consider using a variable block size.

Note: Use of a variable block size is currently *not* supported. If you must use this option, ensure that you have well tested procedures in place that enable you to recover successfully, using backup images that were created with a variable block size.

When using a variable block size, you must specify a backup buffer size that is less than or equal to the maximum limit for the tape devices that you are using. For optimal performance, the buffer size must be equal to the maximum block size limit of the device being used.

Before a tape device can be used on a Windows® operating system, the following command must be issued:

```
db2 initialize tape on <device> using <blksize>
```

Where:

<device>
is a valid tape device name. The default on Windows operating systems is `\\.\TAPE0`.

<blksize>
is the blocking factor for the tape. It must be a factor or multiple of 4096. The default value is the default block size for the device.

Restoring from a backup image with variable block size may return an error. If this happens, you may need to rewrite the image using an appropriate block size. Following is an example on AIX:

```
tcl -b 0 -Bn -f /dev/rmt0 read > backup_filename.file
dd if=backup_filename.file of=/dev/rmt0/ obs=4096 conv=sync
```

The backup image is dumped to a file called `backup_filenam.file`. The **dd** command dumps the image back onto tape, using a block size of 4096.

There is a problem with this approach if the image is too large to dump to a file. One possible solution is to use the **dd** command to dump the image from one tape device to another. This will work as long as the image does not span more than one tape. When using two tape devices, the **dd** command is:

```
dd if=/dev/rmt1 of=/dev/rmt0 obs=4096
```

If using two tape devices is not possible, you may be able to dump the image to a raw device using the **dd** command, and then to dump the image from the raw device to tape. The problem with this approach is that the **dd** command *must* keep track of the number of blocks dumped to the raw device. This number must be specified when the image is moved back to tape. If the **dd** command is used to dump the image from the raw device to tape, the command dumps the entire contents of the raw device to tape. The **dd** utility cannot determine how much of the raw device is used to hold the image.

When using the backup utility, you will need to know the maximum block size limit for your tape devices. Here are some examples:

Device	Attachment	Block Size Limit	DB2 Buffer Size Limit (in 4-KB pages)
8 mm	scsi	131,072	32
3420	s370	65,536	16
3480	s370	65,536	16
3490	s370	65,536	16
3490E	s370	65,536	16
7332 (4 mm)[1]	scsi	262,144	64
3490e	scsi	262,144	64
3590[2]	scsi	2,097,152	512
3570 (magstar MP)		262,144	64

Notes:

1. The 7332 does not implement a block size limit. 256 KB is simply a suggested value. Block size limit is imposed by the parent adapter.
2. While the 3590 does support a 2-MB block size, you could experiment with lower values (like 256 KB), provided the performance is adequate for your needs.

3. For information about your device limit, check your device documentation or consult with the device vendor.

Backing Up to Named Pipes

Support is now available for database backup to (and database restore from) local named pipes on UNIX based systems.

Prerequisites:

Both the writer and the reader of the named pipe must be on the same machine. The pipe must exist and be located on a local file system. Because the named pipe is treated as a local device, there is no need to specify that the target is a named pipe.

Procedure:

Following is an AIX example:

1. Create a named pipe:
   ```
   mkfifo /u/dmcinnis/mypipe
   ```
2. Use this pipe as the target for a database backup operation:
   ```
   db2 backup db sample to /u/dmcinnis/mypipe
   ```
3. If this backup image is going to be used by the restore utility, the restore operation must be invoked *before* the backup operation, so that it will not miss any data:
   ```
   db2 restore db sample into mynewdb from /u/dmcinnis/mypipe
   ```

Related tasks:

- "Using Backup" on page 67

Related reference:

- "BACKUP DATABASE" on page 72
- "RESTORE DATABASE" on page 95

BACKUP DATABASE

Creates a backup copy of a database or a table space.

Scope:

This command only affects the database partition on which it is executed.

Authorization:

One of the following:

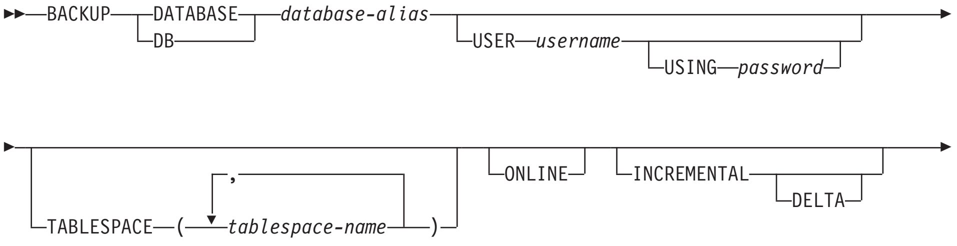
- *sysadm*
- *sysctrl*
- *sysmaint*

Required connection:

Database. This command automatically establishes a connection to the specified database.

Note: If a connection to the specified database already exists, that connection will be terminated and a new connection established specifically for the backup operation. The connection is terminated at the completion of the backup operation.

Command syntax:



```
►─┬──────────────────────────────────────────────────────┬──────────────────►
  ├─USE──┬─TSM──┬──┬──────────────────────────────┬──────┤
  │      └─XBSA─┘  └─OPEN──num-sessions──SESSIONS─┘      │
  │         ┌─,─────────┐                                 │
  │         ▼           │                                 │
  ├─TO────────┬─dir─┬───┴─────────────────────────────────┤
  │           └─dev─┘                                     │
  └─LOAD──library-name──┬───────────────────────────────┬┘
                        └─OPEN──num-sessions──SESSIONS──┘

►─┬───────────────────────────────┬─┬─────────────────────────┬─┬────────────────────┬──►
  └─WITH──num-buffers──BUFFERS────┘ └─BUFFER──buffer-size─────┘ └─PARALLELISM──n─────┘

►─┬─────────────────────┬───────────────────────────────────────────────────────────►◄
  └─WITHOUT PROMPTING───┘
```

Command parameters:

DATABASE database-alias
Specifies the alias of the database to back up.

USER username
Identifies the user name under which to back up the database.

USING password
The password used to authenticate the user name. If the password is omitted, the user is prompted to enter it.

TABLESPACE tablespace-name
A list of names used to specify the table spaces to be backed up.

ONLINE
Specifies online backup. The default is offline backup. Online backups are only available for databases configured with *logretain* or *userexit* enabled.

Note: An online backup operation may time out if there is an IX lock on `sysibm.systables`, because the DB2 backup utility requires an S lock on objects containing LOBs.

INCREMENTAL
Specifies a cumulative (incremental) backup image. An incremental backup image is a copy of all database data that has changed since the most recent successful, full backup operation.

DELTA
Specifies a non-cumulative (delta) backup image. A delta backup image is a copy of all database data that has changed since the most recent successful backup operation of any type.

USE TSM

Specifies that the backup is to use Tivoli Storage Manager (formerly ADSM) output.

OPEN num-sessions SESSIONS

The number of I/O sessions to be created between DB2 and TSM or another backup vendor product.

Note: This parameter has no effect when backing up to tape, disk, or other local device.

USE XBSA

Specifies that the XBSA interface is to be used. Backup Services APIs (XBSA) are an open application programming interface for applications or facilities needing data storage management for backup or archiving purposes. Legato NetWorker is a storage manager that currently supports the XBSA interface.

TO dir/dev

A list of directory or tape device names. The full path on which the directory resides must be specified. If USE TSM, TO, and LOAD are omitted, the default target directory for the backup image is the current working directory of the client computer. This target directory or device must exist on the database server. This parameter may be repeated to specify the target directories and devices that the backup image will span. If more than one target is specified (target1, target2, and target3, for example), target1 will be opened first. The media header and special files (including the configuration file, table space table, and history file) are placed in target1. All remaining targets are opened, and are then used in parallel during the backup operation. Because there is no general tape support on Windows operating systems, each type of tape device requires a unique device driver. To back up to the FAT file system on Windows operating systems, users must conform to the 8.3 naming restriction.

Use of tape devices or floppy disks may generate messages and prompts for user input. Valid response options are:

c Continue. Continue using the device that generated the warning message (for example, when a new tape has been mounted)

d Device terminate. Stop using *only* the device that generated the warning message (for example, when there are no more tapes)

t Terminate. Abort the backup operation.

If the tape system does not support the ability to uniquely reference a backup image, it is recommended that multiple backup copies of the same database not be kept on the same tape.

LOAD library-name

The name of the shared library (DLL on Windows operating systems) containing the vendor backup and restore I/O functions to be used. It can contain the full path. If the full path is not given, it will default to the path on which the user exit program resides.

WITH num-buffers BUFFERS

The number of buffers to be used. The default is 2. However, when creating a backup to multiple locations, a larger number of buffers may be used to improve performance.

BUFFER buffer-size

The size, in 4-KB pages, of the buffer used when building the backup image. The minimum value for this parameter is 8 pages; the default value is 1024 pages. If a buffer size of zero is specified, the value of the database manager configuration parameter *backbufsz* will be used as the buffer allocation size.

If using tape with variable block size, reduce the buffer size to within the range that the tape device supports.. Otherwise, the backup operation may succeed, but the resulting image may not be recoverable.

When using tape devices on SCO UnixWare 7, specify a buffer size of 16.

With most versions of Linux, using DB2's default buffer size for backup operations to a SCSI tape device results in error SQL2025N, reason code 75. To prevent the overflow of Linux internal SCSI buffers, use this formula:

```
bufferpages <= ST_MAX_BUFFERS * ST_BUFFER_BLOCKS / 4
```

where *bufferpages* is the value of either *backbufsz* or *restbufsz*, and `ST_MAX_BUFFERS` and `ST_BUFFER_BLOCKS` are defined in the Linux kernel under the `drivers/scsi` directory.

PARALLELISM n

Determines the number of table spaces which can be read in parallel by the backup utility. The default value is 1.

WITHOUT PROMPTING

Specifies that the backup will run unattended, and that any actions which normally require user intervention will return an error message.

Examples:

In the following example, the database WSDB is defined on all 4 partitions, numbered 0 through 3. The path /dev3/backup is accessible from all partitions. Partition 0 is the catalog partition, and needs to be backed-up separately since this is an offline backup. To perform an offline backup of all the WSDB database partitions to /dev3/backup, issue the following commands from one of the database partitions:

```
    db2_all '<<+0< db2 BACKUP DATABASE wsdb TO /dev3/backup'
    db2_all '|<<-0< db2 BACKUP DATABASE wsdb TO /dev3/backup'
```

In the second command, the db2_all utility will issue the same backup command to each database partition in turn (except partition 0). All four database partition backup images will be stored in the /dev3/backup directory.

In the following example database SAMPLE is backed up to a TSM server using two concurrent TSM client sessions. The backup utility will use four buffers which are the default buffer size (1024 x 4K pages).

```
   db2 backup database sample use tsm open 2 sessions with 4 buffers
```

In the next example, a tablespace level backup of tablespaces (syscatspace, userspace1) of database `payroll` is done to tapes.

```
   db2 backup database payroll tablespace (syscatspace, userspace1) to
      /dev/rmt0, /dev/rmt1 with 8 buffers without prompting
```

Following is a sample weekly incremental backup strategy for a recoverable database. It includes a weekly full database backup operation, a daily non-cumulative (delta) backup operation, and a mid-week cumulative (incremental) backup operation:

```
   (Sun) db2 backup db sample use tsm
   (Mon) db2 backup db sample online incremental delta use tsm
   (Tue) db2 backup db sample online incremental delta use tsm
   (Wed) db2 backup db sample online incremental use tsm
   (Thu) db2 backup db sample online incremental delta use tsm
   (Fri) db2 backup db sample online incremental delta use tsm
   (Sat) db2 backup db sample online incremental use tsm
```

Related reference:

- "RESTORE DATABASE" on page 95
- "ROLLFORWARD DATABASE" on page 134

db2Backup - Backup database

Creates a backup copy of a database or a table space.

Scope:

This API only affects the database partition on which it is executed.

Authorization:

One of the following:

- *sysadm*
- *sysctrl*
- *sysmaint*

Required connection:

Database. This API automatically establishes a connection to the specified database.

The connection will be terminated upon the completion of the backup.

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2Backup */
/* ... */
SQL_API_RC SQL_API_FN
db2Backup (
  db2Uint32     versionNumber,
  void         *pDB2BackupStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2BackupStruct
{
  char                         *piDBAlias;
  char                         oApplicationId[SQLU_APPLID_LEN+1];
  char                         oTimestamp[SQLU_TIME_STAMP_LEN+1];
  struct db2TablespaceStruct  *piTablespaceList;
  struct db2MediaListStruct   *piMediaList;
  char                         *piUsername;
  char                         *piPassword;
  void                         *piVendorOptions;
  db2Uint32                    iVendorOptionsSize;
  db2Uint32                    oBackupSize;
  db2Uint32                    iCallerAction;
  db2Uint32                    iBufferSize;
  db2Uint32                    iNumBuffers;
  db2Uint32                    iParallelism;
  db2Uint32                    iOptions;
} db2BackupStruct;

typedef SQL_STRUCTURE db2TablespaceStruct
{
  char                         **tablespaces;
  db2Uint32                    numTablespaces;
} db2TablespaceStruct;

typedef SQL_STRUCTURE db2MediaListStruct
{
  char                         **locations;
  db2Uint32                    numLocations;
  char                         locationType;
} db2MediaListStruct;
/* ... */
```

Generic API syntax:

```
/* File: db2ApiDf.h */
/* API: db2Backup */
/* ... */
SQL_API_RC SQL_API_FN
db2gBackup (
  db2Uint32     versionNumber,
  void         *pDB2gBackupStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2gBackupStruct
{
  char                         *piDBAlias;
  db2Uint32                    iDBAliasLen;
  char                         *poApplicationId;
  db2Uint32                    iApplicationIdLen;
  char                         *poTimestamp;
  db2Uint32                    iTimestampLen;
  struct db2gTablespaceStruct *piTablespaceList;
  struct db2gMediaListStruct  *piMediaList;
  char                         *piUsername;
  db2Uint32                    iUsernameLen;
  char                         *piPassword;
  db2Uint32                    iPasswordLen;
  void                         *piVendorOptions;
  db2Uint32                    iVendorOptionsSize;
  db2Uint32                    oBackupSize;
  db2Uint32                    iCallerAction;
  db2Uint32                    iBufferSize;
  db2Uint32                    iNumBuffers;
  db2Uint32                    iParallelism;
  db2Uint32                    iOptions;
} db2gBackupStruct;

typedef SQL_STRUCTURE db2gTablespaceStruct
{
  struct db2Char               *tablespaces;
  db2Uint32                    numTablespaces;
} db2gTablespaceStruct;

typedef SQL_STRUCTURE db2gMediaListStruct
{
  struct db2Char               *locations;
  db2Uint32                    numLocations;
  char                         locationType;
} db2gMediaListStruct;

typedef SQL_STRUCTURE db2Char
{
   char            *pioData;
   db2Uint32       iLength;
   db2Uint32       oLength;
} db2Char;
/* ... */
```

API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter *pDB2BackupStruct*.

pDB2BackupStruct
Input. A pointer to the *db2BackupStruct* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

piDBAlias
Input. A string containing the database alias (as cataloged in the system database directory) of the database to back up.

iDBAliasLen
Input. A 4-byte unsigned integer representing the length in bytes of the database alias.

oApplicationId
Output. The API will return a string identifying the agent servicing the application. Can be used to obtain information about the progress of the backup operation using the database monitor.

poApplicationId
Output. Supply a buffer of length SQLU_APPLID_LEN+1 (defined in `sqlutil`). The API will return a string identifying the agent servicing the application. Can be used to obtain information about the progress of the backup operation using the database monitor.

iApplicationIdLen
Input. A 4-byte unsigned integer representing the length in bytes of the *poApplicationId* buffer. Should be equal to SQLU_APPLID_LEN+1 (defined in `sqlutil`).

oTimestamp
Output. The API will return the time stamp of the backup image

poTimestamp
Output. Supply a buffer of length SQLU_TIME_STAMP_LEN+1 (defined in `sqlutil`). The API will return the time stamp of the backup image.

iTimestampLen
Input. A 4-byte unsigned integer representing the length in bytes of the *poTimestamp* buffer. Should be equal to SQLU_TIME_STAMP_LEN+1 (defined in `sqlutil`).

piTablespaceList
Input. List of table spaces to be backed up. Required for table space level backup only. Must be NULL for a database level backup. See structure DB2TablespaceStruct.

piMediaList

Input. This structure allows the caller to specify the destination for the backup operation. The information provided depends on the value of the locationType field. The valid values for locationType (defined in `sqlutil.h`) are:

SQLU_LOCAL_MEDIA

Local devices (a combination of tapes, disks, or diskettes).

SQLU_TSM_MEDIA

TSM. If the locations pointer is set to NULL, the TSM shared library provided with DB2 is used. If a different version of the TSM shared library is desired, use SQLU_OTHER_MEDIA and provide the shared library name.

SQLU_OTHER_MEDIA

Vendor product. Provide the shared library name in the locations field.

SQLU_USER_EXIT

User exit. No additional input is required (only available when server is on OS/2).

For more information, see structure DB2MediaListStruct.

piUsername

Input. A string containing the user name to be used when attempting a connection. Can be NULL.

iUsernameLen

Input. A 4-byte unsigned integer representing the length in bytes of the user name. Set to zero if no user name is provided.

piPassword

Input. A string containing the password to be used with the user name. Can be NULL.

iPasswordLen

Input. A 4-byte unsigned integer representing the length in bytes of the password. Set to zero if no password is provided.

piVendorOptions

Input. Used to pass information from the application to the vendor functions. This data structure must be flat; that is, no level of indirection is supported. Note that byte-reversal is not done, and code page is not checked for this data.

iVendorOptionsSize

Input. The length of the *piVendorOptions* field, which cannot exceed 65535 bytes.

oBackupSize
Output. Size of the backup image (in MB).

iCallerAction
Input. Specifies action to be taken. Valid values (defined in `db2ApiDf.h`) are:

DB2BACKUP_BACKUP
Start the backup.

DB2BACKUP_NOINTERRUPT
Start the backup. Specifies that the backup will run unattended, and that scenarios which normally require user intervention will either be attempted without first returning to the caller, or will generate an error. Use this caller action, for example, if it is known that all of the media required for the backup have been mounted, and utility prompts are not desired.

DB2BACKUP_CONTINUE
Continue the backup after the user has performed some action requested by the utility (mount a new tape, for example).

DB2BACKUP_TERMINATE
Terminate the backup after the user has failed to perform some action requested by the utility.

DB2BACKUP_DEVICE_TERMINATE
Remove a particular device from the list of devices used by backup. When a particular medium is full, backup will return a warning to the caller (while continuing to process using the remaining devices). Call backup again with this caller action to remove the device which generated the warning from the list of devices being used.

DB2BACKUP_PARM_CHK
Used to validate parameters without performing a backup. This option does not terminate the database connection after the call returns. After successful return of this call, it is expected that the user will issue a call with SQLUB_CONTINUE to proceed with the action.

DB2BACKUP_PARM_CHK_ONLY
Used to validate parameters without performing a backup. Before this call returns, the database connection established by this call is terminated, and no subsequent call is required.

iBufferSize
Input. Backup buffer size in 4KB allocation units (pages). Minimum is 8 units. The default is 1024 units.

iNumBuffers
Input. Specifies number of backup buffers to be used. Minimum is 2. Maximum is limited by memory. Can specify 0 for the default value of 2.

iParallelism
Input. Degree of parallelism (number of buffer manipulators). Minimum is 1. Maximum is 1024. The default is 1.

iOptions
Input. A bitmap of backup properties. The options are to be combined using the bitwise OR operator to produce a value for *iOptions*. Valid values (defined in `db2ApiDf.h`) are:

DB2BACKUP_OFFLINE
Offline gives an exclusive connection to the database.

DB2BACKUP_ONLINE
Online allows database access by other applications while the backup operation occurs.

Note: An online backup operation may appear to hang if users are holding locks on SMS LOB data.

DB2BACKUP_DB
Full database backup.

DB2BACKUP_TABLESPACE
Table space level backup. For a table space level backup, provide a list of table spaces in the *piTablespaceList* parameter.

DB2BACKUP_INCREMENTAL
Specifies a cumulative (incremental) backup image. An incremental backup image is a copy of all database data that has changed since the most recent successful, full backup operation.

DB2BACKUP_DELTA
Specifies a noncumulative (delta) backup image. A delta backup image is a copy of all database data that has changed since the most recent successful backup operation of any type.

tablespaces
A pointer to the list of table spaces to be backed up. For C, the list is null-terminated strings. In the generic case, it is a list of *db2Char* structures.

numTablespaces
Number of entries in the *tablespaces* parameter.

locations
A pointer to the list of media locations. For C, the list is null-terminated strings. In the generic case, it is a list of db2Char structures.

numLocations
The number of entries in the *locations* parameter.

locationType
A character indicated the media type. Valid values (defined in `sqlutil.h`.) are:

SQLU_LOCAL_MEDIA
Local devices (tapes, disks, diskettes, or named pipes).

SQLU_TSM_MEDIA
Tivoli Storage Manager.

SQLU_OTHER_MEDIA
Vendor library.

SQLU_USER_EXIT
User exit (only available when the server is on OS/2).

pioData
A pointer to the character data buffer.

iLength
Input. The size of the *pioData* buffer.

oLength
Output. Reserved for future use.

Related samples:

- "dbrecov.sqc -- How to recover a database (C)"
- "dbrecov.sqC -- How to recover a database (C++)"

Backup Sessions - CLP Examples

Example 1

In the following example database SAMPLE is backed up to a TSM server using 2 concurrent TSM client sessions. The backup utility will use 4 buffers which are the default buffer size (1024 x 4K pages).

```
db2 backup database sample use tsm open 2 sessions with 4 buffers

db2 backup database payroll tablespace (syscatspace, userspace1) to
   /dev/rmt0, /dev/rmt1 with 8 buffers without prompting
```

Example 2

Following is a sample weekly incremental backup strategy for a recoverable database. It includes a weekly full database backup operation, a daily non-cumulative (delta) backup operation, and a mid-week cumulative (incremental) backup operation:

```
(Sun) db2 backup db kdr use tsm
(Mon) db2 backup db kdr online incremental delta use tsm
(Tue) db2 backup db kdr online incremental delta use tsm
(Wed) db2 backup db kdr online incremental use tsm
(Thu) db2 backup db kdr online incremental delta use tsm
(Fri) db2 backup db kdr online incremental delta use tsm
(Sat) db2 backup db kdr online incremental use tsm
```

Example 3

To initiate a backup operation to a tape device in a Windows environment, issue:

```
db2 backup database sample to \\.\tape0
```

Related tasks:

- "Using Backup" on page 67

Optimizing Backup Performance

To reduce the amount of time required to complete a backup operation:

- Specify table space backup.

  You can back up (and subsequently recover) part of a database by using the TABLESPACE option on the BACKUP DATABASE command. This facilitates the management of table data, indexes, and long field or large object (LOB) data in separate table spaces.
- Increase the value of the PARALLELISM parameter on the BACKUP DATABASE command so that it reflects the number of table spaces being backed up.

  The PARALLELISM parameter defines the number of processes or threads that are started when reading data from the database. Each process or thread is assigned to a specific table space. When it finishes backing up this table space, it requests another. Note, however, that each process or thread requires both memory and CPU overhead: on a heavily loaded system, keep the PARALLELISM parameter at its default value of 1.
- Increase the backup buffer size.

  The ideal backup buffer size is a multiple of the table space extent size. If you have multiple table spaces with different extent sizes, specify a value that is a multiple of the largest extent size.
- Increase the number of buffers.

If you use multiple buffers and I/O channels, you should use at least twice as many buffers as channels to ensure that the channels do not have to wait for data.

- Use multiple target devices.

Related concepts:

- "Backup Overview" on page 63

Related tasks:

- "Using Backup" on page 67

Chapter 3. Database Restore

This section describes the DB2 UDB restore utility, which is used to rebuild damaged or corrupted databases or table spaces that were previously backed up.

The following topics are covered:

- "Restore Overview"
- "Privileges, Authorities, and Authorization Required to Use Restore" on page 88
- "Using Restore" on page 89
- "Using Incremental Restore in a Test and Production Environment" on page 90
- "Redefining Table Space Containers During a Restore Operation (Redirected Restore)" on page 93
- "Restoring to an Existing Database" on page 94
- "Restoring to a New Database" on page 95
- "RESTORE DATABASE" on page 95
- "db2Restore - Restore database" on page 104
- "Restore Sessions - CLP Examples" on page 115

Restore Overview

The simplest form of the DB2® RESTORE DATABASE command requires only that you specify the alias name of the database that you want to restore. For example:

```
   db2 restore db sample
```

In this example, because the SAMPLE database exists, the following message is returned:

```
SQL2539W  Warning!  Restoring to an existing database that is the same as
the backup image database.  The database files will be deleted.
Do you want to continue ? (y/n)
```

If you specify `y`, and a backup image for the SAMPLE database exists, the restore operation should complete successfully.

A database restore operation requires an exclusive connection: that is, no applications can be running against the database when the operation starts, and the restore utility prevents other applications from accessing the database until the restore operation completes successfully. A table space restore operation, however, can be done online.

A table space is not usable until the restore operation (followed by rollforward recovery) completes successfully.

If you have tables that span more than one table space, you should back up and restore the set of table spaces together.

When doing a partial or subset restore operation, you can use either a table space-level backup image, or a full database-level backup image and choose one or more table spaces from that image. All the log files associated with these table spaces from the time that the backup image was created must exist.

Optimizing Restore Performance

To reduce the amount of time required to complete a restore operation:

- Increase the restore buffer size.

  The restore buffer size must be a positive integer multiple of the backup buffer size specified during the backup operation. If an incorrect buffer size is specified, the buffers allocated will be the smallest acceptable size.
- Increase the number of buffers.

  The value you specify must be a multiple of the number of pages that you specified for the backup buffer. The minimum number of pages is 16.
- Increase the value of the PARALLELISM option.

  This will increase the number of buffer manipulators (BM) that will be used to write to the database during the restore operation. The default value is 1.

Privileges, Authorities, and Authorization Required to Use Restore

Privileges enable users to create or access database resources. Authority levels provide a method of grouping privileges and higher-level database manager maintenance and utility operations. Together, these act to control access to the database manager and its database objects. Users can access only those objects for which they have the appropriate authorization; that is, the required privilege or authority.

You must have SYSADM, SYSCTRL, or SYSMAINT authority to restore to an *existing* database from a full database backup. To restore to a *new* database, you must have SYSADM or SYSCTRL authority.

Using Restore

Prerequisites:

When restoring to an *existing* database, you should not be connected to the database that is to be restored: the restore utility automatically establishes a connection to the specified database, and this connection is terminated at the completion of the restore operation. When restoring to a *new* database, an instance attachment is required to create the database. When restoring to a *new remote* database, you must first attach to the instance where the new database will reside. Then, create the new database, specifying the code page and the territory of the server.

The database can be local or remote.

Restrictions:

The following restrictions apply to the restore utility:

- You can only use the restore utility if the database has been previously backed up using the DB2 backup utility.
- A database restore operation cannot be started while the rollforward process is running.
- You can restore a table space only if the table space currently exists, and if it is the same table space; "same" means that the table space was not dropped and then recreated between the backup and the restore operation.)
- You cannot restore a table space-level backup to a new database.
- You cannot perform an online table space-level restore operation involving the system catalog tables.

Procedure:

The restore utility can be invoked through the command line processor (CLP), the Restore Database notebook or wizard in the Control Center, or the **db2Restore** application programming interface (API).

Following is an example of the RESTORE DATABASE command issued through the CLP:

```
db2 restore db sample from D:\DB2Backups taken at 20010320122644
```

To open the Restore Database notebook or wizard:

1. From the Control Center, expand the object tree until you find the Databases folder.
2. Click on the Databases folder. Any existing databases are displayed in the pane on the right side of the window (the contents pane).

3. Click the right mouse button on the database you want in the contents pane, and select Restore Database or Restore Database Using Wizard from the pop-up menu. The Restore Database notebook or the Restore Database wizard opens.

Detailed information is provided through the online help facility within the Control Center.

Related concepts:

- "Administrative APIs in Embedded SQL or DB2 CLI Programs" in the *Application Development Guide: Programming Client Applications*
- "Introducing the plug-in architecture for the Control Center" in the *Administration Guide: Implementation*

Related reference:

- "db2Restore - Restore database" on page 104

Using Incremental Restore in a Test and Production Environment

Once a production database is enabled for incremental backup and recovery, you can use an incremental or delta backup image to create or refresh a test database. You can do this by using either manual or automatic incremental restore. To restore the backup image from the production database to the test database, use the INTO *target-database-alias* option on the RESTORE DATABASE command. For example, in a production database with the following backup images:

```
backup db prod
Backup successful. The timestamp for this backup image is : <ts1>

backup db prod incremental
Backup successful. The timestamp for this backup image is : <ts2>
```

an example of a manual incremental restore would be:

```
restore db prod incremental taken at ts2 into test without
prompting
DB20000I  The RESTORE DATABASE command completed successfully.

restore db prod incremental taken at ts1 into test without
prompting
DB20000I  The RESTORE DATABASE command completed successfully.

restore db prod incremental taken at ts2 into test without
prompting
DB20000I  The RESTORE DATABASE command completed successfully.
```

If the database TEST already exists, the restore operation will overwrite any data that is already there. If the database TEST does not exist, the restore utility will create it and then populate it with the data from the backup images.

Since automatic incremental restore operations are dependent on the database history, the restore steps change slightly based on whether or not the test database exists. To perform an automatic incremental restore to the database TEST, its history must contain the backup image history for database PROD. The database history for the backup image will replace any database history that already exists for database TEST if:

- the database TEST does not exist when the RESTORE DATABASE command is issued, or
- the database TEST exists when the RESTORE DATABASE command is issued, and the database TEST history contains no records.

The following example shows an automatic incremental restore to database TEST which does not exist:

```
restore db prod incremental automatic taken at ts2 into test without
prompting
DB20000I  The RESTORE DATABASE command completed successfully.
```

The restore utility will create the TEST database and populate it.

If the database TEST does exist and the database history is not empty, you must drop the database before the automatic incremental restore operation as follows:

```
drop db test
DB20000I  The DROP DATABASE command completed successfully.

restore db prod incremental automatic taken at ts2 into test without
prompting
DB20000I  The RESTORE DATABASE command completed successfully.
```

If you do not want to drop the database, you can issue the PRUNE HISTORY command using a timestamp far into the future and the WITH FORCE OPTION parameter before issuing the RESTORE DATABASE command:

```
connect to test
Database Connection Information

Database server       = <server id>
SQL authorization ID       = <id>
Local database alias       = TEST

prune history 9999 with force option
DB20000I  The PRUNE command completed successfully.

connect reset
```

```
DB20000I  The SQL command completed successfully.
restore db prod incremental automatic taken at ts2 into test without
prompting
SQL2540W  Restore is successful, however a warning "2539" was
encountered during Database Restore while processing in No
Interrupt mode.
```

In this case, the RESTORE DATABASE COMMAND will act in the same manner as when the database TEST did not exist.

If the database TEST does exist and the database history is empty, you do not have to drop the database TEST before the automatic incremental restore operation:

```
restore db prod incremental automatic taken at ts2 into test without
prompting
SQL2540W  Restore is successful, however a warning "2539" was
encountered during Database Restore while processing in No
Interrupt mode.
```

You can continue taking incremental or delta backups of the test database without first taking a full database backup. However, if you ever need to restore one of the incremental or delta images you will have to perform a manual incremental restore. This is because automatic incremental restore operations require that each of the backup images restored during an automatic incremental restore be created from the same database alias.

If you make a full database backup of the test database after you complete the restore operation using the production backup image, you can take incremental or delta backups and can restore them using either manual or automatic mode.

Related concepts:

- “Incremental Backup and Recovery” on page 28

Related reference:

- “BACKUP DATABASE” on page 72
- “RESTORE DATABASE” on page 95
- “LIST HISTORY” on page 228

Redefining Table Space Containers During a Restore Operation (Redirected Restore)

During a database backup operation, a record is kept of all the table space containers associated with the table spaces that are being backed up. During a restore operation, all containers listed in the backup image are checked to determine if they exist and if they are accessible. If one or more of these containers is inaccessible because of media failure (or for any other reason), the restore operation will fail. A successful restore operation in this case requires redirection to different containers. DB2® supports adding, changing, or removing table space containers.

You can redefine table space containers by invoking the RESTORE DATABASE command and specifying the REDIRECT parameter, or by using the Containers page of the Restore Database notebook in the Control Center. The process for invoking a redirected restore of an incremental backup image is similar to the process for a non-incremental backup image: Call the RESTORE DATABASE command with the REDIRECT parameter and specify the backup image from which the database should be incrementally restored.

During a redirected restore operation, directory and file containers are automatically created if they do not already exist. The database manager does not automatically create device containers.

Container redirection provides considerable flexibility for managing table space containers. For example, even though adding containers to SMS table spaces is not supported, you could accomplish this by specifying an additional container when invoking a redirected restore operation.

Related reference:

- “RESTORE DATABASE” on page 95
- “Restore Sessions - CLP Examples” on page 115

Related samples:

- “dbrecov.out -- HOW TO RECOVER A DATABASE (C)”
- “dbrecov.sqc -- How to recover a database (C)”
- “dbrecov.out -- HOW TO RECOVER A DATABASE (C++)”
- “dbrecov.sqC -- How to recover a database (C++)”

Restoring to an Existing Database

You can restore a full database backup image to an existing database. The backup image may differ from the existing database in its alias name, its database name, or its database seed.

A database *seed* is a unique identifier for a database that does not change during the life of the database. The seed is assigned by the database manager when the database is created. DB2® always uses the seed from the backup image.

When restoring to an existing database, the restore utility:

- Deletes table, index, and long field data from the existing database, and replaces it with data from the backup image.
- Replaces table entries for each table space being restored.
- Retains the recovery history file, unless it is damaged or has no entries. If the recovery history file is damaged, the database manager copies the file from the backup image.
- Retains the authentication type for the existing database.
- Retains the database directories for the existing database. The directories define where the database resides, and how it is cataloged.
- Compares the database seeds. If the seeds are different:
  - Deletes the logs associated with the existing database.
  - Copies the database configuration file from the backup image.
  - Sets NEWLOGPATH to the value of the *logpath* database configuration parameter if NEWLOGPATH was specified on the RESTORE DATABASE command.

  If the database seeds are the same:
  - Deletes the logs if the image is for a non-recoverable database.
  - Retains the current database configuration file, unless the file has been corrupted, in which case the file is copied from the backup image.
  - Sets NEWLOGPATH to the value of the *logpath* database configuration parameter if NEWLOGPATH was specified on the RESTORE DATABASE command; otherwise, copies the current log path to the database configuration file. Validates the log path: If the path cannot be used by the database, changes the database configuration to use the default log path.

Restoring to a New Database

You can create a new database and then restore a full database backup image to it. If you do not create a new database, the restore utility will create one.

When restoring to a new database, the restore utility:

- Creates a new database, using the database alias name that was specified through the target database alias parameter. (If a target database alias was not specified, the restore utility creates the database with an alias that is the same as that specified through the source database alias parameter.)
- Restores the database configuration file from the backup image.
- Sets NEWLOGPATH to the value of the *logpath* database configuration parameter if NEWLOGPATH was specified on the RESTORE DATABASE command. Validates the log path: If the path cannot be used by the database, changes the database configuration to use the default log path.
- Restores the authentication type from the backup image.
- Restores the comments from the database directories in the backup image.
- Restores the recovery history file for the database.

RESTORE DATABASE

Rebuilds a damaged or corrupted database that has been backed up using the DB2 backup utility. The restored database is in the same state it was in when the backup copy was made. This utility can also restore to a database with a name different from the database name in the backup image (in addition to being able to restore to a new database).

This utility can also be used to restore backup images that were produced by the previous two versions of DB2. If a migration is required, it will be invoked automatically at the end of the restore operation.

If, at the time of the backup operation, the database was enabled for rollforward recovery, the database can be brought to the state it was in prior to the occurrence of the damage or corruption by invoking the rollforward utility after successful completion of a restore operation.

This utility can also restore from a table space level backup.

To restore a database that was backed up on a different workstation platform, use the db2move utility. You cannot restore a database made on one platform directly to another platform. Supported versions of Microsoft Windows are considered equivalent.

Scope:

This command only affects the node on which it is executed.

Authorization:

To restore to an existing database, one of the following:

- *sysadm*
- *sysctrl*
- *sysmaint*

To restore to a new database, one of the following:

- *sysadm*
- *sysctrl*

Required connection:

Database, to restore to an existing database. This command automatically establishes a connection to the specified database.

Instance and database, to restore to a new database. The instance attachment is required to create the database.

To restore to a new database at an instance different from the current instance (as defined by the value of the **DB2INSTANCE** environment variable), it is necessary to first attach to the instance where the new database will reside.

To restore to a new *remote* database, it is necessary to attach to the instance where the new database will reside.

Command syntax:

```
►►──RESTORE──┬─DATABASE─┬──source-database-alias──┬─┤ restore-options ├─┬──────────────►◄
             └─DB───────┘                          ├─CONTINUE────────────┤
                                                   └─ABORT───────────────┘
```

restore-options:

```
├──┬──────────────────────────────────────────┬──────────────────────────────────────►
   └─USER──username──┬──────────────────────┬─┘
                     └─USING──password──────┘
```

```
>--+---------------------------------------------------+-------------->
   +-TABLESPACE--+--------+----------------------------+
   |             '-ONLINE-'                            |
   |                  .-,---------------.              |
   |                  V                 |              |
   +-TABLESPACE--(----tablespace-name---+--)--+--------+
   |                                          '-ONLINE-'|
   '-HISTORY FILE--+--------+--------------------------'
                   '-ONLINE-'

>--+-----------------------------+------------------------------------>
   '-INCREMENTAL--+-----------+--'
                  +-AUTO------+
                  +-AUTOMATIC-+
                  '-ABORT-----'

>--+-----------------------------------------------------------+----->
   +-USE--+-TSM--+--+---------------------------------+--------+
   |      '-XBSA-'  '-OPEN--num-sessions--SESSIONS-'           |
   |          .-,-----------.                                  |
   |          V             |                                  |
   +-FROM------+-directory-+-+---------------------------------+
   |           '-device----'                                   |
   '-LOAD--shared-library--+---------------------------------+-'
                           '-OPEN--num-sessions--SESSIONS-'

>--+-------------------------+--+--------------------------+--------->
   '-TAKEN AT--date-time-----'  '-TO--target-directory-----'

>--+------------------------------------+--+--------------------------+-->
   '-INTO--target-database-alias--------'  '-NEWLOGPATH--directory----'

>--+--------------------------------+--+----------------------+--+-----------------------+-->
   '-WITH--num-buffers--BUFFERS-----'  '-BUFFER--buffer-size--'  '-DLREPORT--filename----'

>--+------------------+--+----------+--+------------------+---------->
   '-REPLACE EXISTING-'  '-REDIRECT-'  '-PARALLELISM--n---'

>--+--------------------------+--+-------------------+--+---------------------+---><
   '-WITHOUT ROLLING FORWARD--'  '-WITHOUT DATALINK--'  '-WITHOUT PROMPTING---'
```

Command parameters:

DATABASE source-database-alias

Alias of the source database from which the backup was taken.

CONTINUE
Specifies that the containers have been redefined, and that the final step in a redirected restore operation should be performed.

ABORT
This parameter:

- Stops a redirected restore operation. This is useful when an error has occurred that requires one or more steps to be repeated. After RESTORE DATABASE with the ABORT option has been issued, each step of a redirected restore operation must be repeated, including RESTORE DATABASE with the REDIRECT option.
- Terminates an incremental restore operation before completion.

USER username
Identifies the user name under which the database is to be restored.

USING password
The password used to authenticate the user name. If the password is omitted, the user is prompted to enter it.

TABLESPACE tablespace-name
A list of names used to specify the table spaces that are to be restored.

ONLINE
This keyword, applicable only when performing a table space-level restore operation, is specified to allow a backup image to be restored online. This means that other agents can connect to the database while the backup image is being restored, and that the data in other table spaces will be available while the specified table spaces are being restored.

HISTORY FILE
This keyword is specified to restore only the history file from the backup image.

INCREMENTAL
Without additional parameters, INCREMENTAL specifies a manual cumulative restore operation. During manual restore the user must issue each restore command manually for each image involved in the restore. Do so according to the following order: last, first, second, third and so on up to and including the last image.

INCREMENTAL AUTOMATIC/AUTO
Specifies an automatic cumulative restore operation.

INCREMENTAL ABORT
Specifies abortion of an in-progress manual cumulative restore operation.

USE TSM
Specifies that the database is to be restored from TSM-managed output.

OPEN num-sessions SESSIONS
Specifies the number of I/O sessions that are to be used with TSM or the vendor product.

USE XBSA
Specifies that the XBSA interface is to be used. Backup Services APIs (XBSA) are an open application programming interface for applications or facilities needing data storage management for backup or archiving purposes. Legato NetWorker is a storage manager that currently supports the XBSA interface.

FROM directory/device
The directory or device on which the backup images reside. If USE TSM, FROM, and LOAD are omitted, the default value is the current directory.

On Windows operating systems, the specified directory must not be a DB2-generated directory. For example, given the following commands:

```
db2 backup database sample to c:\backup
db2 restore database sample from c:\backup
```

DB2 generates subdirectories under the `c:\backup` directory that should be ignored. To specify precisely which backup image to restore, use the TAKEN AT parameter. There may be several backup images stored on the same path.

If several items are specified, and the last item is a tape device, the user is prompted for another tape. Valid response options are:

c Continue. Continue using the device that generated the warning message (for example, continue when a new tape has been mounted).

d Device terminate. Stop using *only* the device that generated the warning message (for example, terminate when there are no more tapes).

t Terminate. Abort the restore operation after the user has failed to perform some action requested by the utility.

LOAD shared-library
The name of the shared library (DLL on Windows operating systems) containing the vendor backup and restore I/O functions to be used. The name can contain a full path. If the full path is not given, the value defaults to the path on which the user exit program resides.

TAKEN AT date-time

The time stamp of the database backup image. The time stamp is displayed after successful completion of a backup operation, and is part of the path name for the backup image. It is specified in the form *yyyymmddhhmmss*. A partial time stamp can also be specified. For example, if two different backup images with time stamps 19971001010101 and 19971002010101 exist, specifying `19971002` causes the image with time stamp `19971002010101` to be used. If a value for this parameter is not specified, there must be only one backup image on the source media.

TO target-directory

The target database directory. This parameter is ignored if the utility is restoring to an existing database. The drive and directory that you specify must be local.

Note: On Windows operating systems, when using this parameter specify the drive letter. For example, you might specify `x:\`*`path_name`* to restore to a specific path, or `x:` if you do not need to specify a path. If the path name is too long, an error is returned.

INTO target-database-alias

The target database alias. If the target database does not exist, it is created.

When you restore a database backup to an existing database, the restored database inherits the alias and database name of the existing database. When you restore a database backup to a nonexistent database, the new database is created with the alias and database name that you specify. This new database name must be unique on the system where you restore it.

NEWLOGPATH directory

The absolute pathname of a directory that will be used for active log files after the restore operation. This parameter has the same function as the *newlogpath* database configuration parameter, except that its effect is limited to the restore operation in which it is specified. The parameter can be used when the log path in the backup image is not suitable for use after the restore operation; for example, when the path is no longer valid, or is being used by a different database.

WITH num-buffers BUFFERS

The number of buffers to be used. The default value is 2. However, a larger number of buffers can be used to improve performance when multiple sources are being read from, or if the value of PARALLELISM has been increased.

BUFFER buffer-size

The size, in pages, of the buffer used for the restore operation. The minimum value for this parameter is 8 pages; the default value is 1024 pages. If a buffer size of zero is specified, the value of the database manager configuration parameter *restbufsz* will be used as the buffer allocation size.

The restore buffer size must be a positive integer multiple of the backup buffer size specified during the backup operation. If an incorrect buffer size is specified, the buffers are allocated to be of the smallest acceptable size.

When using tape devices on SCO UnixWare 7, specify a buffer size of 16.

DLREPORT filename

The file name, if specified, must be specified as an absolute path. Reports the files that become unlinked, as a result of a fast reconcile, during a restore operation. This option is only to be used if the table being restored has a DATALINK column type and linked files.

REPLACE EXISTING

If a database with the same alias as the target database alias already exists, this parameter specifies that the restore utility is to replace the existing database with the restored database. This is useful for scripts that invoke the restore utility, because the command line processor will not prompt the user to verify deletion of an existing database. If the WITHOUT PROMPTING parameter is specified, it is not necessary to specify REPLACE EXISTING, but in this case, the operation will fail if events occur that normally require user intervention.

REDIRECT

Specifies a redirected restore operation. To complete a redirected restore operation, this command should be followed by one or more SET TABLESPACE CONTAINERS commands, and then by a RESTORE DATABASE command with the CONTINUE option.

Note: All commands associated with a single redirected restore operation must be invoked from the same window or CLP session.

WITHOUT ROLLING FORWARD

Specifies that the database is not to be put in rollforward pending state after it has been successfully restored.

If, following a successful restore operation, the database is in rollforward pending state, the ROLLFORWARD command must be invoked before the database can be used again.

WITHOUT DATALINK
Specifies that any tables with DATALINK columns are to be put in DataLink_Reconcile_Pending (DRP) state, and that no reconciliation of linked files is to be performed.

PARALLELISM n
Specifies the number of buffer manipulators that are to be spawned during the restore operation. The default value is 1.

WITHOUT PROMPTING
Specifies that the restore operation is to run unattended. Actions that normally require user intervention will return an error message. When using a removable media device, such as tape or diskette, the user is prompted when the device ends, even if this option is specified.

Examples:

In the following example, the database WSDB is defined on all 4 partitions, numbered 0 through 3. The path /dev3/backup is accessible from all partitions. The following offline backup images are available from /dev3/backup:

```
wsdb.0.db2inst1.NODE0000.CATN0000.20020331234149.001
wsdb.0.db2inst1.NODE0001.CATN0000.20020331234427.001
wsdb.0.db2inst1.NODE0002.CATN0000.20020331234828.001
wsdb.0.db2inst1.NODE0003.CATN0000.20020331235235.001
```

To restore the catalog partition first, then all other database partitions of the WSDB database from the /dev3/backup directory, issue the following commands from one of the database partitions:

```
db2_all '<<+0< db2 RESTORE DATABASE wsdb FROM /dev3/backup
TAKEN AT 20020331234149
  INTO wsdb REPLACE EXISTING'
db2_all '<<+1< db2 RESTORE DATABASE wsdb FROM /dev3/backup
TAKEN AT 20020331234427
  INTO wsdb REPLACE EXISTING'
db2_all '<<+2< db2 RESTORE DATABASE wsdb FROM /dev3/backup
TAKEN AT 20020331234828
  INTO wsdb REPLACE EXISTING'
db2_all '<<+3< db2 RESTORE DATABASE wsdb FROM /dev3/backup
TAKEN AT 20020331235235
  INTO wsdb REPLACE EXISTING'
```

The db2_all utility issues the restore command to each specified database partition.

Following is a typical redirected restore scenario for a database whose alias is MYDB:

1. Issue a RESTORE DATABASE command with the REDIRECT option.

   ```
   db2 restore db mydb replace existing redirect
   ```

After successful completion of step 1, and before completing step 3, the restore operation can be aborted by issuing:

```
db2 restore db mydb abort
```

2. Issue a SET TABLESPACE CONTAINERS command for each table space whose containers must be redefined. For example:

```
db2 set tablespace containers for 5 using
   (file 'f:\ts3con1' 20000, file 'f:\ts3con2' 20000)
```

To verify that the containers of the restored database are the ones specified in this step, issue the LIST TABLESPACE CONTAINERS command.

3. After successful completion of steps 1 and 2, issue:

```
db2 restore db mydb continue
```

This is the final step of the redirected restore operation.

4. If step 3 fails, or if the restore operation has been aborted, the redirected restore can be restarted, beginning at step 1.

Following is a sample weekly incremental backup strategy for a recoverable database. It includes a weekly full database backup operation, a daily non-cumulative (delta) backup operation, and a mid-week cumulative (incremental) backup operation:

```
(Sun) backup db mydb use tsm
(Mon) backup db mydb online incremental delta use tsm
(Tue) backup db mydb online incremental delta use tsm
(Wed) backup db mydb online incremental use tsm
(Thu) backup db mydb online incremental delta use tsm
(Fri) backup db mydb online incremental delta use tsm
(Sat) backup db mydb online incremental use tsm
```

For an automatic database restore of the images created on Friday morning, issue:

```
restore db mydb incremental automatic taken at (Fri)
```

For a manual database restore of the images created on Friday morning, issue:

```
restore db mydb incremental taken at (Fri)
restore db mydb incremental taken at (Sun)
restore db mydb incremental taken at (Wed)
restore db mydb incremental taken at (Thu)
restore db mydb incremental taken at (Fri)
```

Usage notes:

Any RESTORE DATABASE command of the form `db2 restore db <name>` will perform a full database restore, regardless of whether the image being restored is a database image or a table space image. Any RESTORE DATABASE command of the form `db2 restore db <name> tablespace` will perform a table space restore of the table spaces found in the image. Any RESTORE DATABASE command in which a list of table spaces is provided will perform a restore of whatever table spaces are explicitly listed.

Related reference:

- "BACKUP DATABASE" on page 72
- "ROLLFORWARD DATABASE" on page 134
- "db2move - Database Movement Tool" in the *Command Reference*

db2Restore - Restore database

Rebuilds a damaged or corrupted database that has been backed up using db2Backup - Backup Database. The restored database is in the same state it was in when the backup copy was made. This utility can also restore to a database with a name different from the database name in the backup image (in addition to being able to restore to a new database).

This utility can also be used to restore DB2 databases created in the two previous releases.

This utility can also restore from a table space level backup.

Scope:

This API only affects the database partition from which it is called.

Authorization:

To restore to an existing database, one of the following:

- *sysadm*
- *sysctrl*
- *sysmaint*

To restore to a new database, one of the following:

- *sysadm*
- *sysctrl*

Required connection:

Database, to restore to an existing database. This API automatically establishes a connection to the specified database and will release the connection when the restore operation finishes.

Instance and database, to restore to a new database. The instance attachment is required to create the database.

To restore to a new database at an instance different from the current instance (as defined by the value of the DB2INSTANCE environment variable), it is necessary to first attach to the instance where the new database will reside.

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2Restore */
/* ... */
SQL_API_RC SQL_API_FN
db2Restore (
  db2Uint32      versionNumber,
  void         *pDB2RestoreStruct,
  struct sqlca *pSqlca);
/* ... */

typedef SQL_STRUCTURE db2RestoreStruct
{
  char                        *piSourceDBAlias;
  char                        *piTargetDBAlias;
  char                        oApplicationId[SQLU_APPLID_LEN+1];
  char                        *piTimestamp;
  char                        *piTargetDBPath;
  char                        *piReportFile;
  struct db2TablespaceStruct  *piTablespaceList;
  struct db2MediaListStruct   *piMediaList;
  char                        *piUsername;
  char                        *piPassword;
  char                        *piNewLogPath;
  void                        *piVendorOptions;
  db2Uint32                   iVendorOptionsSize;
  db2Uint32                   iParallelism;
  db2Uint32                   iBufferSize;
  db2Uint32                   iNumBuffers;
  db2Uint32                   iCallerAction;
  db2Uint32                   iOptions;
} db2BackupStruct;

typedef SQL_STRUCTURE db2TablespaceStruct
{
  char                        **tablespaces;
  db2Uint32                   numTablespaces;
} db2TablespaceStruct;

typedef SQL_STRUCTURE db2MediaListStruct
{
  char                        **locations;
  db2Uint32                   numLocations;
  char                        locationType;
} db2MediaListStruct;
/* ... */
```

Generic API syntax:

```
/* File: db2ApiDf.h */
/* API: db2gRestore */
/* ... */
SQL_API_RC SQL_API_FN
db2gRestore (
  db2Uint32      versionNumber,
  void         *pDB2gRestoreStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2gRestoreStruct
{
  char                        *piSourceDBAlias;
  db2Uint32                   iSourceDBAliasLen;
  char                        *piTargetDBAlias;
  db2Uint32                   iTargetDBAliasLen;
  char                        *poApplicationId;
  db2Uint32                   iApplicationIdLen;
  char                        *piTimestamp;
  db2Uint32                   iTimestampLen;
  char                        *piTargetDBPath;
  db2Uint32                   iTargetDBPathLen;
  char                        *piReportFile;
  db2Uint32                   iReportFileLen;
  struct db2gTablespaceStruct *piTablespaceList;
  struct db2gMediaListStruct  *piMediaList;
  char                        *piUsername;
  db2Uint32                   iUsernameLen;
  char                        *piPassword;
  db2Uint32                   iPasswordLen;
  char                        *piNewLogPath;
  db2Uint32                   iNewLogPathLen;
  void                        *piVendorOptions;
  db2Uint32                   iVendorOptionsSize;
  db2Uint32                   iParallelism;
  db2Uint32                   iBufferSize;
  db2Uint32                   iNumBuffers;
  db2Uint32                   iCallerAction;
  db2Uint32                   iOptions;
} db2gBackupStruct;

typedef SQL_STRUCTURE db2gTablespaceStruct
{
  struct db2Char              *tablespaces;
  db2Uint32                   numTablespaces;
} db2gTablespaceStruct;

typedef SQL_STRUCTURE db2gMediaListStruct
{
  struct db2Char              *locations;
  db2Uint32                   numLocations;
  char                        locationType;
} db2gMediaListStruct;

typedef SQL_STRUCTURE db2Char
{
   char            *pioData;
   db2Uint32       iLength;
   db2Uint32       oLength;
} db2Char;
/* ... */
```

API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter pParamStruct.

pDB2RestoreStruct
Input. A pointer to the *db2RestoreStruct* structure

pSqlca
Output. A pointer to the *sqlca* structure.

piSourceDBAlias
Input. A string containing the database alias of the source database backup image.

iSourceDBAliasLen
Input. A 4-byte unsigned integer representing the length in bytes of the source database alias.

piTargetDBAlias
Input. A string containing the target database alias. If this parameter is null, the *piSourceDBAlias* will be used.

iTargetDBAliasLen
Input. A 4-byte unsigned integer representing the length in bytes of the target database alias.

oApplicationId
Output. The API will return a string identifying the agent servicing the application. Can be used to obtain information about the progress of the backup operation using the database monitor.

poApplicationId
Output. Supply a buffer of length SQLU_APPLID_LEN+1 (defined in `sqlutil`). The API will return a string identifying the agent servicing the application. Can be used to obtain information about the progress of the backup operation using the database monitor.

iApplicationIdLen
Input. A 4-byte unsigned integer representing the length in bytes of the poApplicationId buffer. Should be equal to SQLU_APPLID_LEN+1 (defined in `sqlutil`).

piTimestamp
Input. A string representing the timestamp of the backup image. This field is optional if there is only one backup image in the source specified.

iTimestampLen
Input. A 4-byte unsigned integer representing the length in bytes of the piTimestamp buffer.

piTargetDBPath

Input. A string containing the relative or fully qualified name of the target database directory on the server. Used if a new database is to be created for the restored backup; otherwise not used.

piReportFile

Input. The file name, if specified, must be fully qualified. The datalinks files that become unlinked during restore (as a result of a fast reconcile) will be reported.

iReportFileLen

Input. A 4-byte unsigned integer representing the length in bytes of the piReportFile buffer.

piTablespaceList

Input. List of table spaces to be restored. Used when restoring a subset of table spaces from a database or table space backup image. See the *DB2TablespaceStruct* structure . The following restrictions apply:

- The database must be recoverable; that is, log retain or user exits must be enabled.
- The database being restored to must be the same database that was used to create the backup image. That is, table spaces can not be added to a database through the table space restore function.
- The rollforward utility will ensure that table spaces restored in a partitioned database environment are synchronized with any other database partition containing the same table spaces. If a table space restore operation is requested and the *piTablespaceList* is NULL, the restore utility will attempt to restore all of the table spaces in the backup image.

When restoring a table space that has been renamed since it was backed up, the new table space name must be used in the restore command. If the old table space name is used, it will not be found.

piMediaList

Input. Source media for the backup image. The information provided depends on the value of the locationType field. The valid values for locationType (defined in `sqlutil`) are:

SQLU_LOCAL_MEDIA

Local devices (a combination of tapes, disks, or diskettes).

SQLU_TSM_MEDIA

TSM. If the locations pointer is set to NULL, the TSM shared library provided with DB2 is used. If a different version of the TSM shared library is desired, use SQLU_OTHER_MEDIA and provide the shared library name.

SQLU_OTHER_MEDIA
Vendor product. Provide the shared library name in the locations field.

SQLU_USER_EXIT
User exit. No additional input is required (only available when server is on OS/2).

piUsername
Input. A string containing the user name to be used when attempting a connection. Can be NULL.

iUsernameLen
Input. A 4-byte unsigned integer representing the length in bytes of piUsername. Set to zero if no user name is provided.

piPassword
Input. A string containing the password to be used with the user name. Can be NULL.

iPasswordLen
Input. A 4-byte unsigned integer representing the length in bytes of piPassword. Set to zero if no password is provided.

piNewLogPath
Input. A string representing the path to be used for logging after the restore has completed. If this field is null the default log path will be used.

iNewLogPathLen
Input. A 4-byte unsigned integer representing the length in bytes of *piNewLogPath*.

piVendorOptions
Input. Used to pass information from the application to the vendor functions. This data structure must be flat; that is, no level of indirection is supported. Note that byte-reversal is not done, and the code page is not checked for this data.

iVendorOptionsSize
Input. The length of the *piVendorOptions*, which cannot exceed 65535 bytes.

iParallelism
Input. Degree of parallelism (number of buffer manipulators). Minimum is 1. Maximum is 1024. The default is 1.

iBufferSize
Input. Backup buffer size in 4KB allocation units (pages). Minimum is

8 units. The default is 1024 units. The size entered for a restore must be equal to or an integer multiple of the buffer size used to produce the backup image.

iNumBuffers
Input. Specifies number of restore buffers to be used.

iCallerAction
Input. Specifies action to be taken. Valid values (defined in `db2ApiDf`) are:

DB2RESTORE_RESTORE
Start the restore operation.

DB2RESTORE_NOINTERRUPT
Start the restore. Specifies that the restore will run unattended, and that scenarios which normally require user intervention will either be attempted without first returning to the caller, or will generate an error. Use this caller action, for example, if it is known that all of the media required for the restore have been mounted, and utility prompts are not desired.

DB2RESTORE_CONTINUE
Continue the restore after the user has performed some action requested by the utility (mount a new tape, for example).

DB2RESTORE_TERMINATE
Terminate the restore after the user has failed to perform some action requested by the utility.

DB2RESTORE_DEVICE_TERMINATE
Remove a particular device from the list of devices used by restore. When a particular device has exhausted its input, restore will return a warning to the caller. Call restore again with this caller action to remove the device which generated the warning from the list of devices being used.

DB2RESTORE_PARM_CHK
Used to validate parameters without performing a restore. This option does not terminate the database connection after the call returns. After successful return of this call, it is expected that the user will issue a call with DB2RESTORE_CONTINUE to proceed with the action.

DB2RESTORE_PARM_CHK_ONLY
Used to validate parameters without performing a restore. Before this call returns, the database connection established by this call is terminated, and no subsequent call is required.

DB2RESTORE_TERMINATE_INCRE
Terminate an incremental restore operation before completion.

DB2RESTORE_RESTORE_STORDEF
Initial call. Table space container redefinition requested.

DB2RESTORE_STORDEF_NOINTERRUPT
Initial call. The restore will run uninterrupted. Table space container redefinition requested.

iOptions
Input. A bitmap of restore properties. The options are to be combined using the bitwise OR operator to produce a value for *iOptions*. Valid values (defined in `db2ApiDf`) are:

DB2RESTORE_OFFLINE
Perform an offline restore operation.

DB2RESTORE_ONLINE
Perform an online restore operation.

DB2RESTORE_DB
Restore all table spaces in the database. This must be run offline

DB2RESTORE_TABLESPACE
Restore only the table spaces listed in the *piTablespaceList* parameter from the backup image. This can be online or offline.

DB2RESTORE_HISTORY
Restore only the history file.

DB2RESTORE_INCREMENTAL
Perform a manual cumulative restore operation.

DB2RESTORE_AUTOMATIC
Perform an automatic cumulative (incremental) restore operation. Must be specified with DB2RESTORE_INCREMENTAL.

DB2RESTORE_DATALINK
Perform reconciliation operations. Tables with a defined DATALINK column must have RECOVERY YES option specified.

DB2RESTORE_NODATALINK
Do not perform reconciliation operations. Tables with DATALINK columns are placed into DataLink_Roconcile_pending (DRP) state. Tables with a defined DATALINK column must have the RECOVERY YES option specified.

DB2RESTORE_ROLLFWD
Place the database in rollforward pending state after it has been successfully restored.

DB2RESTORE_NOROLLFWD
Do not place the database in rollforward pending state after it has been successfully restored. This cannot be specified for backups taken online or for table space level restores. If, following a successful restore, the database is in roll-forward pending state, db2Rollforward - Rollforward Database must be executed before the database can be used.

tablespaces
A pointer to the list of table spaces to be backed up. For C, the list is null-terminated strings. In the generic case, it is a list of *db2Char* structures.

numTablespaces
Number of entries in the *tablespaces* parameter.

locations
A pointer to the list of media locations. For C, the list is null-terminated strings. In the generic case, it is a list of db2Char structures.

numLocations
The number of entries in the *locations* parameter.

locationType
A character indicated the media type. Valid values (defined in `sqlutil`) are:

SQLU_LOCAL_MEDIA
Local devices(tapes, disks, diskettes, or named pipes).

SQLU_TSM_MEDIA
Tivoli Storage Manager.

SQLU_OTHER_MEDIA
Vendor library.

SQLU_USER_EXIT
User exit (only available when the server is on OS/2).

pioData
A pointer to the character data buffer.

iLength
Input. The size of the *pioData* buffer

oLength
Output. Reserverd for future use.

Usage notes:

For offline restore, this utility connects to the database in exclusive mode. The utility fails if any application, including the calling application, is already connected to the database that is being restored. In addition, the request will fail if the restore utility is being used to perform the restore, and any application, including the calling application, is already connected to any database on the same workstation. If the connect is successful, the API locks out other applications until the restore is completed.

The current database configuration file will not be replaced by the backup copy unless it is unusable. If the file is replaced, a warning message is returned.

The database or table space must have been backed up using db2Backup - Backup Database.

If the caller action is DB2RESTORE_NOINTERRUPT, the restore continues without prompting the application. If the caller action is DB2RESTORE_RESTORE, and the utility is restoring to an existing database, the utility returns control to the application with a message requesting some user interaction. After handling the user interaction, the application calls RESTORE DATABASE again, with the caller action set to indicate whether processing is to continue (DB2RESTORE_CONTINUE) or terminate (DB2RESTORE_TERMINATE) on the subsequent call. The utility finishes processing, and returns an SQLCODE in the *sqlca*.

To close a device when finished, set the caller action to DB2RESTORE_DEVICE_TERMINATE. If, for example, a user is restoring from 3 tape volumes using 2 tape devices, and one of the tapes has been restored, the application obtains control from the API with an SQLCODE indicating end of tape. The application can prompt the user to mount another tape, and if the user indicates ″no more″, return to the API with caller action SQLUD_DEVICE_TERMINATE to signal end of the media device. The device driver will be terminated, but the rest of the devices involved in the restore will continue to have their input processed until all segments of the restore set have been restored (the number of segments in the restore set is placed on the last media device during the backup process). This caller action can be used with devices other than tape (vendor supported devices).

To perform a parameter check before returning to the application, set caller action to DB2RESTORE_PARM_CHK.

Set caller action to DB2RESTORE_RESTORE_STORDEF when performing a redirected restore; used in conjunction with sqlbstsc - Set Tablespace Containers.

If a system failure occurs during a critical stage of restoring a database, the user will not be able to successfully connect to the database until a successful restore is performed. This condition will be detected when the connection is attempted, and an error message is returned. If the backed-up database is not configured for roll-forward recovery, and there is a usable current configuration file with either of these parameters enabled, following the restore, the user will be required to either take a new backup of the database, or disable the log retain and user exit parameters before connecting to the database.

Although the restored database will not be dropped (unless restoring to a nonexistent database), if the restore fails, it will not be usable.

If the restore type specifies that the history file on the backup is to be restored, it will be restored over the existing history file for the database, effectively erasing any changes made to the history file after the backup that is being restored. If this is undesirable, restore the history file to a new or test database so that its contents can be viewed without destroying any updates that have taken place.

If, at the time of the backup operation, the database was enabled for roll forward recovery, the database can be brought to the state it was in prior to the occurrence of the damage or corruption by issuing db2Rollforward after successful execution of db2Restore. If the database is recoverable, it will default to roll forward pending state after the completion of the restore.

If the database backup image is taken offline, and the caller does not want to roll forward the database after the restore, the DB2RESTORE_NOROLLFWD option can be used for the restore. This results in the database being useable immediately after the restore. If the backup image is taken online, the caller must roll forward through the corresponding log records at the completion of the restore.

Related samples:

- "dbrecov.sqc -- How to recover a database (C)"
- "dbrecov.sqC -- How to recover a database (C++)"

Restore Sessions - CLP Examples

Example 1

Following is a typical non-incremental redirected restore scenario for a database whose alias is MYDB:

1. Issue a RESTORE DATABASE command with the REDIRECT option.

   ```
   db2 restore db mydb replace existing redirect
   ```

2. Issue a SET TABLESPACE CONTAINERS command for each table space whose containers you want to redefine. For example, in a Windows environment:

   ```
   db2 set tablespace containers for 5 using
          (file 'f:\ts3con1'20000, file 'f:\ts3con2'20000)
   ```

   To verify that the containers of the restored database are the ones specified in this step, issue the LIST TABLESPACE CONTAINERS command for every table space whose container locations are being redefined.
3. After successful completion of steps 1 and 2, issue:

   ```
   db2 restore db mydb continue
   ```

   This is the final step of the redirected restore operation.
4. If step 3 fails, or if the restore operation has been aborted, the redirected restore can be restarted, beginning at step 1.

Notes:

1. After successful completion of step 1, and before completing step 3, the restore operation can be aborted by issuing:

   ```
   db2 restore db mydb abort
   ```
2. If step 3 fails, or if the restore operation has been aborted, the redirected restore can be restarted, beginning at step 1.

Example 2

Following is a typical manual incremental redirected restore scenario for a database whose alias is MYDB and has the following backup images:

```
backup db mydb
Backup successful. The timestamp for this backup image is : <ts1>

backup db mydb incremental
Backup successful. The timestamp for this backup image is : <ts2>
```

1. Issue a RESTORE DATABASE command with the INCREMENTAL and REDIRECT options.

   ```
   db2 restore db mydb incremental taken at <ts2> replace existing redirect
   ```
2. Issue a SET TABLESPACE CONTAINERS command for each table space whose containers must be redefined. For example, in a Windows environment:

   ```
   db2 set tablespace containers for 5 using
          (file 'f:\ts3con1'20000, file 'f:\ts3con2'20000)
   ```

   To verify that the containers of the restored database are the ones specified in this step, issue the LIST TABLESPACE CONTAINERS command.
3. After successful completion of steps 1 and 2, issue:

   ```
   db2 restore db mydb continue
   ```

4. The remaining incremental restore commands can now be issued as follows:

   ```
   db2 restore db mydb incremental taken at <ts1>
   db2 restore db mydb incremental taken at <ts2>
   ```

   This is the final step of the redirected restore operation.

Notes:

1. After successful completion of step 1, and before completing step 3, the restore operation can be aborted by issuing:

   ```
   db2 restore db mydb abort
   ```

2. After successful completion of step 3, and before issuing all the required commands in step 4, the restore operation can be aborted by issuing:

   ```
   db2 restore db mydb incremental abort
   ```

3. If step 3 fails, or if the restore operation has been aborted, the redirected restore can be restarted, beginning at step 1.
4. If either restore command fails in step 4, the failing command can be reissued to continue the restore process.

Example 3

Following is a typical automatic incremental redirected restore scenario for the same database:

1. Issue a RESTORE DATABASE command with the INCREMENTAL AUTOMATIC and REDIRECT options.

   ```
   db2 restore db mydb incremental automatic taken at <ts2>
          replace existing redirect
   ```

2. Issue a SET TABLESPACE CONTAINERS command for each table space whose containers must be redefined. For example, in a Windows environment:

   ```
   db2 set tablespace containers for 5 using
          (file 'f:\ts3con1'20000, file 'f:\ts3con2'20000)
   ```

   To verify that the containers of the restored database are the ones specified in this step, issue the LIST TABLESPACE CONTAINERS command.

3. After successful completion of steps 1 and 2, issue:

   ```
   db2 restore db mydb continue
   ```

   This is the final step of the redirected restore operation.

Notes:

1. After successful completion of step 1, and before completing step 3, the restore operation can be aborted by issuing:

   ```
   db2 restore db mydb abort
   ```

2. If step 3 fails, or if the restore operation has been aborted, the redirected restore can be restarted, beginning at step 1 after issuing:

   ```
   db2 restore db mydb incremental abort
   ```

Related reference:

- "RESTORE DATABASE" on page 95
- "LIST TABLESPACE CONTAINERS" in the *Command Reference*
- "SET TABLESPACE CONTAINERS" in the *Command Reference*

Chapter 4. Rollforward Recovery

This section describes the DB2 UDB rollforward utility, which is used to recover a database by applying transactions that were recorded in the database recovery log files.

The following topics are covered:

- "Rollforward Overview"
- "Privileges, Authorities, and Authorization Required to Use Rollforward" on page 121
- "Using Rollforward" on page 121
- "Rolling Forward Changes in a Table Space" on page 123
- "Recovering a Dropped Table" on page 128
- "Using the Load Copy Location File" on page 130
- "Synchronizing Clocks in a Partitioned Database System" on page 132
- "Client/Server Timestamp Conversion" on page 134
- "ROLLFORWARD DATABASE" on page 134
- "db2Rollforward - Rollforward Database" on page 145
- "Rollforward Sessions - CLP Examples" on page 157

Rollforward Overview

The simplest form of the DB2® ROLLFORWARD DATABASE command requires only that you specify the alias name of the database that you want to rollforward recover. For example:

```
   db2 rollforward db sample to end of logs and stop
```

In this example, the command returns:

```
                              Rollforward Status

 Input database alias                   = sample
 Number of nodes have returned status   = 1

 Node number                            = 0
 Rollforward status                     = not pending
 Next log file to be read               =
 Log files processed                    =  -
 Last committed transaction             = 2001-03-11-02.39.48.000000

DB20000I  The ROLLFORWARD command completed successfully.
```

The general approach to rollforward recovery involves:

1. Invoking the rollforward utility without the STOP option.
2. Invoking the rollforward utility with the QUERY STATUS option

   If you specify recovery to the end of the logs, the QUERY STATUS option can indicate that one or more log files is missing, if the returned point in time is earlier than you expect.

   If you specify point-in-time recovery, the QUERY STATUS option will help you to ensure that the rollforward operation has completed at the correct point.
3. Invoking the rollforward utility with the STOP option. After the operation stops, it is not possible to roll additional changes forward.

A database must be restored successfully (using the restore utility) before it can be rolled forward, but a table space does not. A table space may be temporarily put in rollforward pending state, but not require a restore operation to undo it (following a power interruption, for example).

When the rollforward utility is invoked:

- If the database is in rollforward pending state, the database is rolled forward. If table spaces are also in rollforward pending state, you must invoke the rollforward utility again after the database rollforward operation completes to roll the table spaces forward.
- If the database is *not* in rollforward pending state, but table spaces in the database *are* in rollforward pending state:
  - If you specify a list of table spaces, only those table spaces are rolled forward.
  - If you do not specify a list of table spaces, all table spaces that are in rollforward pending state are rolled forward.

A database rollforward operation runs offline. The database is not available for use until the rollforward operation completes successfully, and the operation cannot complete unless the STOP option was specified when the utility was invoked.

A table space rollforward operation can run offline. The database is not available for use until the rollforward operation completes successfully. This occurs if the end of the logs is reached, or if the STOP option was specified when the utility was invoked.

You can perform an *online* rollforward operation on table spaces, as long as SYSCATSPACE is not included. When you perform an online rollforward operation on a table space, the table space is not available for use, but the other table spaces in the database *are* available.

When you first create a database, it is enabled for circular logging only. This means that logs are reused, rather than being saved or archived. With circular logging, rollforward recovery is not possible: only crash recovery or version recovery can be done. Archived logs document changes to a database that occur after a backup was taken. You enable log archiving (and rollforward recovery) by setting the *logretain* database configuration parameter to `RECOVERY`, or setting the *userexit* database configuration parameter to `YES`, or both. The default value for both of these parameters is `NO`, because initially, there is no backup image that you can use to recover the database. When you change the value of one or both of these parameters, the database is put into backup pending state, and you must take an offline backup of the database before it can be used again.

Related concepts:

- "Using the Load Copy Location File" on page 130
- "Understanding Recovery Logs" on page 34

Related reference:

- "ROLLFORWARD DATABASE" on page 134
- "Configuration Parameters for Database Logging" on page 39

Privileges, Authorities, and Authorization Required to Use Rollforward

Privileges enable users to create or access database resources. Authority levels provide a method of grouping privileges and higher-level database manager maintenance and utility operations. Together, these act to control access to the database manager and its database objects. Users can access only those objects for which they have the appropriate authorization; that is, the required privilege or authority.

You must have SYSADM, SYSCTRL, or SYSMAINT authority to use the rollforward utility.

Using Rollforward

Prerequisites:

You should not be connected to the database that is to be rollforward recovered: the rollforward utility automatically establishes a connection to the specified database, and this connection is terminated at the completion of the rollforward operation.

Do not restore table spaces without cancelling a rollforward operation that is in progress; otherwise, you may have a table space set in which some table spaces are in rollforward in progress state, and some table spaces are in rollforward pending state. A rollforward operation that is in progress will only operate on the tables spaces that are in rollforward in progress state.

The database can be local or remote.

Restrictions:

The following restrictions apply to the rollforward utility:

- You can only invoke one rollforward operation at a time. If there are many table spaces to recover, you can specify all of them in the same operation.
- If you have renamed a table space following the most recent backup operation, ensure that you use the new name when rolling the table space forward. The previous table space name will not be recognized.
- You cannot cancel a rollforward operation that is running. You can only cancel a rollforward operation that has completed, but for which the STOP option has not been specified, or a rollforward operation that has failed before completing.
- You cannot *continue* a table space rollforward operation to a point in time, specifying a time stamp that is less than the previous one. If a point in time is not specified, the previous one is used. You can initiate a rollforward operation to a point in time by just specifying STOP, but this is only allowed if the table spaces involved were all restored from the same offline backup image. In this case, no log processing is required. If you start another rollforward operation with a different table space list before the in-progress rollforward operation is either completed or cancelled, an error message (SQL4908) is returned. Invoke the LIST TABLESPACES command on all nodes to determine which table spaces are currently being rolled forward (rollforward in progress state), and which table spaces are ready to be rolled forward (rollforward pending state). You have three options:
  - Finish the in-progress rollforward operation on all table spaces.
  - Finish the in-progress rollforward operation on a subset of table spaces. (This may not be possible if the rollforward operation is to continue to a specific point in time, which requires the participation of all nodes.)
  - Cancel the in-progress rollforward operation.
- In a partitioned database environment, the rollforward utility must be invoked from the catalog node of the database.

Procedure:

The rollforward utility can be invoked through the command line processor (CLP), the Rollforward Database notebook in the Control Center, or the **db2Rollforward** application programming interface (API).

Following is an example of the ROLLFORWARD DATABASE command issued through the CLP:

```
db2 rollforward db sample to end of logs and stop
```

To open the Rollforward Database notebook:

1. From the Control Center, expand the object tree until you find the Databases folder.
2. Click on the Databases folder. Any existing databases are displayed in the pane on the right side of the window (the contents pane).
3. Click the right mouse button on the database you want in the contents pane, and select Rollforward from the pop-up menu. The Rollforward Database notebook opens.

Detailed information is provided through the online help facility within the Control Center.

Related concepts:

- "Administrative APIs in Embedded SQL or DB2 CLI Programs" in the *Application Development Guide: Programming Client Applications*
- "Introducing the plug-in architecture for the Control Center" in the *Administration Guide: Implementation*

Related reference:

- "db2Rollforward - Rollforward Database" on page 145

Rolling Forward Changes in a Table Space

If the database is enabled for forward recovery, you have the option of backing up, restoring, and rolling forward table spaces instead of the entire database. You may want to implement a recovery strategy for individual table spaces because this can save time: it takes less time to recover a portion of the database than it does to recover the entire database. For example, if a disk is bad, and it contains only one table space, that table space can be restored and rolled forward without having to recover the entire database, and without impacting user access to the rest of the database, unless the damaged table space contains the system catalog tables; in this situation, you cannot connect to the database. (The system catalog table space can be restored independently if a table space-level backup image containing the system catalog table space is available.) Table space-level backups also allow you to back up critical parts of the database more frequently than other parts, and requires less time than backing up the entire database.

After a table space is restored, it is always in rollforward pending state. To make the table space usable, you must perform rollforward recovery on it. In most cases, you have the option of rolling forward to the end of the logs, or rolling forward to a point in time. You cannot, however, roll table spaces containing system catalog tables forward to a point in time. These table spaces must be rolled forward to the end of the logs to ensure that all table spaces in the database remain consistent.

When a table space is rolled forward, DB2® will skip files which are known not to contain any log records affecting that table space. If you want all of the log files to be processed, set the DB2_COLLECT_TS_REC_INFO registry variable to false.

The table space change history file (DB2TSCHG.HIS), located in the database directory, keeps track of which logs should be processed for each table space. You can view the contents of this file using the **db2logsForRfwd** utility, and delete entries from it using the PRUNE HISTORY command. During a database restore operation, DB2TSCHG.HIS is restored from the backup image and then brought up to date during the database rollforward operation. If no information is available for a log file, it is treated as though it is required for the recovery of every table space.

Since information for each log file is flushed to disk after the log becomes inactive, this information can be lost as a result of a crash. To compensate for this, if a recovery operation begins in the middle of a log file, the entire log is treated as though it contains modifications to every table space in the system. After this, the active logs will be processed and the information for them will be rebuilt. If information for older or archived log files is lost in a crash situation and no information for them exists in the data file, they will be treated as though they contain modifications for every table space during the table space recovery operation.

Before rolling a table space forward, invoke the LIST TABLESPACES SHOW DETAIL command. This command returns the *minimum recovery time*, which is the earliest point in time to which the table space can be rolled forward. The minimum recovery time is updated when data definition language (DDL) statements are run against the table space, or against tables in the table space. The table space must be rolled forward to at least the minimum recovery time, so that it becomes synchronized with the information in the system catalog tables. If recovering more than one table space, the table spaces must be rolled forward to at least the highest minimum recovery time of all the table spaces being recovered. In a partitioned database environment, issue the

LIST TABLESPACES SHOW DETAIL command on all partitions. The table spaces must be rolled forward to at least the highest minimum recovery time of all the table spaces on all partitions.

If you are rolling table spaces forward to a point in time, and a table is contained in multiple table spaces, all of these table spaces must be rolled forward simultaneously. If, for example, the table data is contained in one table space, and the index for the table is contained in another table space, you must roll both table spaces forward simultaneously to the same point in time.

If the data and the long objects in a table are in separate table spaces, and the long object data has been reorganized, the table spaces for both the data and the long objects must be restored and rolled forward together. You should take a backup of the affected table spaces after the table is reorganized.

If you want to roll a table space forward to a point in time, and a table in the table space is either:

- An underlying table for a materialized query or staging table that is in another table space
- A materialized query or staging table for a table in another table space

You should roll both table spaces forward to the same point in time. If you do not, the materialized query or staging table is placed in check pending state at the end of the rollforward operation. The materialized query table will need to be fully refreshed, and the staging table will be marked as incomplete.

If you want to roll a table space forward to a point in time, and a table in the table space participates in a referential integrity relationship with another table that is contained in another table space, you should roll both table spaces forward simultaneously to the same point in time. If you do not, the child table in the referential integrity relationship will be placed in check pending state at the end of the rollforward operation. When the child table is later checked for constraint violations, a check on the entire table is required. If any of the following tables exist, they will also be placed in check pending state with the child table:

- Any descendent materialized query tables for the child table
- Any descendent staging tables for the child table
- Any descendent foreign key tables of the child table

These tables will require full processing to bring them out of the check pending state. If you roll both table spaces forward simultaneously, the constraint will remain active at the end of the point-in-time rollforward operation.

Ensure that a point-in-time table space rollforward operation does not cause a transaction to be rolled back in some table spaces, and committed in others. This can happen if:

- A point-in-time rollforward operation is performed on a subset of the table spaces that were updated by a transaction, and that point in time precedes the time at which the transaction was committed.
- Any table contained in the table space being rolled forward to a point in time has an associated trigger, or is updated by a trigger that affects table spaces other than the one that is being rolled forward.

The solution is to find a suitable point in time that will prevent this from happening.

You can issue the QUIESCE TABLESPACES FOR TABLE command to create a transaction-consistent point in time for rolling table spaces forward. The quiesce request (in share, intent to update, or exclusive mode) waits (through locking) for all running transactions against those table spaces to complete, and blocks new requests. When the quiesce request is granted, the table spaces are in a consistent state. To determine a suitable time to stop the rollforward operation, you can look in the recovery history file to find quiesce points, and check whether they occur after the minimum recovery time.

After a table space point-in-time rollforward operation completes, the table space is put in backup pending state. You must take a backup of the table space, because all updates made to it between the point in time to which you rolled forward and the current time have been removed. You can no longer roll the table space forward to the current time from a previous database- or table space-level backup image. The following example shows why the table space-level backup image is required, and how it is used. (To make the table space available, you can either back up the entire database, the table space that is in backup pending state, or a set of table spaces that includes the table space that is in backup pending state.)

```
Database                            Time of rollforward of    Restore
backup                              table space TABSP1 to     database.
                                    T2. Back up TABSP1.       Roll forward
                                                              to end of logs.
T1             T2                   T3                        T4
|              |                    |                         |
|              |                    |                         |
|-----------------------------------------------------------------------------
               | Logs are not
                 applied to TABSP1
                 between T2 and T3
                 when it is rolled
                 forward to T2.
```

Figure 15. Table Space Backup Requirement

In the preceding example, the database is backed up at time T1. Then, at time T3, table space TABSP1 is rolled forward to a specific point in time (T2), The table space is backed up after time T3. Because the table space is in backup pending state, this backup operation is mandatory. The time stamp of the table space backup image is after time T3, but the table space is at time T2. Log records from between T2 and T3 are not applied to TABSP1. At time T4, the database is restored, using the backup image created at T1, and rolled forward to the end of the logs. Table space TABSP1 is put in restore pending state at time T3, because the database manager assumes that operations were performed on TABSP1 between T3 and T4 without the log changes between T2 and T3 having been applied to the table space. If these log changes were in fact applied as part of the rollforward operation against the database, this assumption would be incorrect. The table space-level backup that must be taken after the table space is rolled forward to a point in time allows you to roll that table space forward past a previous point-in-time rollforward operation (T3 in the example).

Assuming that you want to recover table space TABSP1 to T4, you would restore the table space from a backup image that was taken after T3 (either the required backup, or a later one), then roll TABSP1 forward to the end of the logs.

In the preceding example, the most efficient way of restoring the database to time T4 would be to perform the required steps in the following order:
1. Restore the database.
2. Restore the table space.
3. Roll the database forward.
4. Roll the table space forward.

Because you restore the table space before rolling the database forward, resource is not used to apply log records to the table space when the database is rolled forward.

If you cannot find the TABSP1 backup image that follows time T3, or you want to restore TABSP1 to T3 (or earlier), you can:

- Roll the table space forward to T3. You do not need to restore the table space again, because it was restored from the database backup image.
- Restore the table space again, using the database backup taken at time T1, then roll the table space forward to a time that precedes time T3.
- Drop the table space.

In a partitioned database environment:

- You must simultaneously roll all parts of a table space forward to the same point in time at the same time. This ensures that the table space is consistent across database partitions.
- If some database partitions are in rollforward pending state, and on other database partitions, some table spaces are in rollforward pending state (but the database partitions are not), you must first roll the database partitions forward, and then roll the table spaces forward.
- If you intend to roll a table space forward to the end of the logs, you do not have to restore it at each database partition; you only need to restore it at the database partitions that require recovery. If you intend to roll a table space forward to a point in time, however, you must restore it at each database partition.

Related concepts:

- “Using the Load Copy Location File” on page 130

Related reference:

- “ROLLFORWARD DATABASE” on page 134

Recovering a Dropped Table

You may occasionally drop a table whose data you still need. If this is the case, you should consider making your critical tables recoverable following a drop table operation.

You could recover the table data by invoking a database restore operation, followed by a database rollforward operation to a point in time before the table was dropped. This may be time consuming if the database is large, and your data will be unavailable during recovery.

DB2's dropped table recovery feature lets you recover your dropped table data using table space-level restore and rollforward operations. This will be faster than database-level recovery, and your database will remain available to users.

Prerequisites:

For a dropped table to be recoverable, the table space in which the table resides must have the DROPPED TABLE RECOVERY option turned on. This can be done during table space creation, or by invoking the ALTER TABLESPACE statement. The DROPPED TABLE RECOVERY option is table space-specific and limited to regular table spaces. To determine if a table space is enabled for dropped table recovery, you can query the DROP_RECOVERY column in the SYSCAT.TABLESPACES catalog table. Dropped table recovery is enabled by default for newly created data table spaces.

When a DROP TABLE statement is run against a table whose table space is enabled for dropped table recovery, an additional entry (identifying the dropped table) is made in the log files. An entry is also made in the recovery history file, containing information that can be used to recreate the table.

Restrictions:

There are some restrictions on the type of data that is recoverable from a dropped table. It is not possible to recover:

- Large object (LOB) or long field data. The DROPPED TABLE RECOVERY option is not supported for large table spaces. If you attempt to recover a dropped table that contains LOB or LONG VARCHAR columns, these columns will be set to NULL in the generated export file. The DROPPED TABLE RECOVERY option can only be used for regular table spaces, not for temporary or large table spaces.
- The metadata associated with row types. (The data is recovered, but not the metadata.) The data in the hierarchy table of the typed table will be recovered. This data may contain more information than appeared in the typed table that was dropped.

Procedure:

Only one dropped table can be recovered at a time. You can recover a dropped table by doing the following:

1. Identify the dropped table by invoking the LIST HISTORY DROPPED TABLE command. The dropped table ID is listed in the Backup ID column.

2. Restore a database- or table space-level backup image taken before the table was dropped.
3. Create an export directory to which files containing the table data are to be written. This directory must either be accessible to all database partitions, or exist on each partition. Subdirectories under this export directory are created automatically by each database partition. These subdirectories are named `NODE`*nnnn*, where *nnnn* represents the database partition or node number. Data files containing the dropped table data as it existed on each database partition are exported to a lower subdirectory called `data`. For example, `\export_directory\NODE0000\data`.
4. Roll forward to a point in time after the table was dropped, using the RECOVER DROPPED TABLE option on the ROLLFORWARD DATABASE command. Alternatively, roll forward to the end of the logs, so that updates to other tables in the table space or database are not lost.
5. Recreate the table using the CREATE TABLE statement from the recovery history file.
6. Import the table data that was exported during the rollforward operation into the table.

The names of linked files associated with DATALINK columns *can* be recovered. After importing the table data, the table should be reconciled with the DB2 Data Links Manager. Backup images of the files may or may not be restored by the DB2 Data Links Manager, depending on whether garbage collection has already deleted them.

Related reference:

- "ALTER TABLESPACE statement" in the *SQL Reference, Volume 2*
- "CREATE TABLE statement" in the *SQL Reference, Volume 2*
- "ROLLFORWARD DATABASE" on page 134
- "LIST HISTORY" on page 228

Using the Load Copy Location File

The DB2LOADREC registry variable is used to identify the file with the load copy location information. This file is used during rollforward recovery to locate the load copy. It has information about:

- Media type
- Number of media devices to be used
- Location of the load copy generated during a table load operation
- File name of the load copy, if applicable

If the location file does not exist, or no matching entry is found in the file, the information from the log record is used.

The information in the file may be overwritten before rollforward recovery takes place.

Notes:

1. In a partitioned database environment, the DB2LOADREC registry variable must be set for all the database partition servers using the **db2set** command.
2. In a partitioned database environment, the load copy file must exist at each database partition server, and the file name (including the path) must be the same.
3. If an entry in the file identified by the DB2LOADREC registry variable is not valid, the old load copy location file is used to provide information to replace the invalid entry.

The following information is provided in the location file. The first five parameters must have valid values, and are used to identify the load copy. The entire structure is repeated for each load copy recorded. For example:

```
TIMestamp      19950725182542         * Time stamp generated at load time
SCHema         PAYROLL                * Schema of table loaded
TABlename      EMPLOYEES              * Table name
DATabasename   DBT                    * Database name
DB2instance    TORONTO                * DB2INSTANCE
BUFfernumber   NULL                   * Number of buffers to be used for
                                        recovery
SESsionnumber  NULL                   * Number of sessions to be used for
                                        recovery
TYPeofmedia    L                      * Type of media - L for local device
                                                        A for TSM
                                                        O for other vendors
LOCationnumber 3                      * Number of locations
   ENTry       /u/toronto/dbt.payroll.employes.001
   ENT         /u/toronto/dbt.payroll.employes.002
   ENT         /dev/rmt0
TIM            19950725192054
SCH            PAYROLL
TAB            DEPT
DAT            DBT
DB2®            TORONTO
SES            NULL
BUF            NULL
TYP            A
TIM            19940325192054
SCH            PAYROLL
TAB            DEPT
DAT            DBT
DB2            TORONTO
SES            NULL
```

```
BUF          NULL
TYP          O
SHRlib       /@sys/lib/backup_vendor.a
```

Notes:

1. The first three characters in each keyword are significant. All keywords are required in the specified order. Blank lines are not accepted.
2. The time stamp is in the form *yyyymmddhhmmss*.
3. All fields are mandatory, except for BUF and SES, which can be NULL. If SES is NULL, the value specified by the *numloadrecses* configuration parameter is used. If BUF is NULL, the default value is SES+2.
4. If even one of the entries in the location file is invalid, the previous load copy location file is used to provide those values.
5. The media type can be local device (L for tape, disk or diskettes), TSM (A), or other vendor (O). If the type is L, the number of locations, followed by the location entries, is required. If the type is A, no further input is required. If the type is O, the shared library name is required.
6. The SHRlib parameter points to a library that has a function to store the load copy data.
7. If you invoke a load operation, specifying the COPY NO or the NONRECOVERABLE option, and do not take a backup copy of the database or affected table spaces after the operation completes, you cannot restore the database or table spaces to a point in time that follows the load operation. That is, you cannot use rollforward recovery to rebuild the database or table spaces to the state they were in following the load operation. You can only restore the database or table spaces to a point in time that precedes the load operation.

If you want to use a particular load copy, you can use the recovery history file for the database to determine the time stamp for that specific load operation. In a partitioned database environment, the recovery history file is local to each database partition.

Related reference:

- Appendix F, "Tivoli Storage Manager" on page 319

Synchronizing Clocks in a Partitioned Database System

You should maintain relatively synchronized system clocks across the database partition servers to ensure smooth database operations and unlimited forward recoverability. Time differences among the database partition servers, plus any potential operational and communications delays for a transaction should be less than the value specified for the *max_time_diff* (maximum time difference among nodes) database manager configuration parameter.

To ensure that the log record time stamps reflect the sequence of transactions in a partitioned database system, DB2® uses the system clock on each machine as the basis for the time stamps in the log records. If, however, the system clock is set ahead, the log clock is automatically set ahead with it. Although the system clock can be set back, the clock for the logs cannot, and remains at the *same* advanced time until the system clock matches this time. The clocks are then in synchrony. The implication of this is that a short term system clock error on a database node can have a long lasting effect on the time stamps of database logs.

For example, assume that the system clock on database partition server A is mistakenly set to November 7, 1999 when the year is 1997, and assume that the mistake is corrected *after* an update transaction is committed in the partition at that database partition server. If the database is in continual use, and is regularly updated over time, any point between November 7, 1997 and November 7, 1999 is virtually unreachable through rollforward recovery. When the COMMIT on database partition server A completes, the time stamp in the database log is set to 1999, and the log clock remains at November 7, 1999 until the system clock matches this time. If you attempt to roll forward to a point in time within this time frame, the operation will stop at the first time stamp that is beyond the specified stop point, which is November 7, 1997.

Although DB2 cannot control updates to the system clock, the *max_time_diff* database manager configuration parameter reduces the chances of this type of problem occurring:

- The configurable values for this parameter range from 1 minute to 24 hours.
- When the first connection request is made to a non-catalog node, the database partition server sends its time to the catalog node for the database. The catalog node then checks that the time on the node requesting the connection, and its own time are within the range specified by the *max_time_diff* parameter. If this range is exceeded, the connection is refused.
- An update transaction that involves more than two database partition servers in the database must verify that the clocks on the participating database partition servers are in synchrony before the update can be committed. If two or more database partition servers have a time difference that exceeds the limit allowed by *max_time_diff*, the transaction is rolled back to prevent the incorrect time from being propagated to other database partition servers.

Related reference:

- "Maximum Time Difference Among Nodes configuration parameter - max_time_diff" in the *Administration Guide: Performance*

Client/Server Timestamp Conversion

This section explains the generation of timestamps in a client/server environment:

- If you specify a local time for a rollforward operation, all messages returned will also be in local time.

  Note: All times are converted on the server and (in partitioned database environments) on the catalog node.
- The timestamp string is converted to GMT on the server, so the time represents the server's time zone, not the client's. If the client is in a different time zone from the server, the server's local time should be used.
- If the timestamp string is close to the time change due to daylight savings time, it is important to know whether the stop time is before or after the time change so that it is specified correctly.

Related concepts:

- "Rollforward Overview" on page 119
- "Synchronizing Clocks in a Partitioned Database System" on page 132

ROLLFORWARD DATABASE

Recovers a database by applying transactions recorded in the database log files. Invoked after a database or a table space backup image has been restored, or if any table spaces have been taken offline by the database due to a media error. The database must be recoverable (that is, either *logretain*, *userexit*, or both of these database configuration parameters must be enabled) before the database can be rollforward recovered.

Scope:

In a partitioned database environment, this command can only be invoked from the catalog partition. A database or table space rollforward operation to a specified point in time affects all partitions that are listed in the `db2nodes.cfg` file. A database or table space rollforward operation to the end of logs affects the partitions that are specified. If no partitions are specified, it affects all partitions that are listed in the `db2nodes.cfg` file; if rollforward recovery is not needed on a particular partition, that partition is ignored.

Authorization:

One of the following:

- *sysadm*
- *sysctrl*
- *sysmaint*

Required connection:

None. This command establishes a database connection.

Command syntax:

```
>>──ROLLFORWARD──┬─DATABASE─┬──database-alias──┬───────────────────────────────┬──────────>
                 └─DB───────┘                  └─USER──username──┬───────────┬─┘
                                                                 └─USING──password─┘

>──┬─────────────────────────────────────────────────────────────────────────────────────┬──>
   ├─TO──┬─isotime──┬──────────────────┬──┬──────────────────────┬─┬──┬─────────────────┬─┤
   │     │          └─USING LOCAL TIME─┘  └─ON ALL DBPARTITIONNUMS─┘ │  ├─AND COMPLETE─┤   │
   │     └─END OF LOGS──┬──────────────────────────────────┬────────┘  └─AND STOP─────┘   │
   │                    └─┤ On Database Partition clause ├─┘                              │
   └─┬─COMPLETE──────────────────────────────┬──┬──────────────────────────────────┬──────┘
     ├─STOP──────────────────────────────────┤  └─┤ On Database Partition clause ├─┘
     ├─CANCEL────────────────────────────────┤
     └─QUERY STATUS──┬──────────────────┬────┘
                     └─USING LOCAL TIME─┘

>──┬──────────────────────────────────────────────────────┬────────────────────────────────>
   └─TABLESPACE──┬─ONLINE─────────────────────────────────┤
                 │    ┌─,───────────────┐                 │
                 │    ▼                 │                 │
                 └─(────tablespace-name──┴─)──┬────────┬──┘
                                              └─ONLINE─┘

>──┬────────────────────────────────────────────────────────────────────────┬──────────────>
   └─OVERFLOW LOG PATH──(──log-directory──┬─────────────────────────────┬──)─┘
                                          └─,─┤ Log Overflow clause ├─┘

>──┬────────────┬──┬───────────────────────────────────────────────────────────┬─────────><
   └─NORETRIEVE─┘  └─RECOVER DROPPED TABLE──drop-table-id──TO──export-directory─┘
```

On Database Partition clause:

```
├──ON──┬─┤ Database Partition List clause ├──────────────────────────────────┬──────────┤
       └─ALL DBPARTITIONNUMS──┬──────────────────────────────────────────────┤
                              └─EXCEPT──┤ Database Partition List clause ├─┘
```

Database Partition List clause:

```
                       .-,-------------------------------------------------------.
                       V                                                         |
|--+-DBPARTITIONNUM--+--(----db-partition-number1--+--------------------------+--+-->
   '-DBPARTITIONNUMS-'                             '-TO--db-partition-number2-'

>--)-------------------------------------------------------------------------------|
```

Log Overflow clause:

```
   .-,-------------------------------------------------------.
   V                                                         |
|----log-directory--ON DBPARTITIONNUM--db-partition-number1--+---------------------|
```

Command parameters:

DATABASE database-alias
: The alias of the database that is to be rollforward recovered.

USER username
: The user name under which the database is to be rollforward recovered.

USING password
: The password used to authenticate the user name. If the password is omitted, the user is prompted to enter it.

TO

isotime

The point in time to which all committed transactions are to be rolled forward (including the transaction committed precisely at that time, as well as all transactions committed previously).

This value is specified as a time stamp, a 7-part character string that identifies a combined date and time. The format is *yyyy-mm-dd-hh.mm.ss.nnnnnn* (year, month, day, hour, minutes, seconds, microseconds), expressed in Coordinated Universal Time (UTC). UTC helps to avoid having the same time stamp associated with different logs (because of a change in time associated with daylight savings time, for example). The time stamp in a backup image is based on the local time at which the backup operation started. The CURRENT TIMEZONE special register specifies the difference between UTC and local time at the application server. The difference is represented by a time duration (a decimal number in which the first two digits represent the number of hours, the next two digits represent the number of minutes, and the last two digits represent the number of seconds). Subtracting CURRENT TIMEZONE from a local time converts that local time to UTC.

USING LOCAL TIME

Allows the user to rollforward to a point in time that is the user's local time rather than GMT time. This makes it easier for users to rollforward to a specific point in time on their local machines, and eliminates potential user errors due to the translation of local to GMT time.

Notes:

1. If the user specifies a local time for rollforward, all messages returned to the user will also be in local time. Note that all times are converted on the server, and if MPP, on the catalog database partition.
2. The timestamp string is converted to GMT on the server, so the time is local to the server's time zone, not the client's. If the client is in one time zone and the server in another, the server's local time should be used. This is different from the local time option from the Control Center, which is local to the client.
3. If the timestamp string is close to the time change of the clock due to daylight savings, it is important to know if the stop time is before or after the clock change, and specify it correctly.

END OF LOGS

Specifies that all committed transactions from all online archive log files listed in the database configuration parameter *logpath* are to be applied.

ALL DBPARTITIONNUMS

Specifies that transactions are to be rolled forward on all partitions specified in the `db2nodes.cfg` file. This is the default if a database partition clause is not specified.

EXCEPT

Specifies that transactions are to be rolled forward on all partitions specified in the `db2nodes.cfg` file, except those specified in the database partition list.

ON DBPARTITIONNUM / ON DBPARTITIONNUMS

Roll the database forward on a set of database partitions.

db-partition-number1

Specifies a database partition number in the database partition list.

db-partition-number2

Specifies the second database partition number, so that all partitions from *db-partition-number1* up to and including *db-partition-number2* are included in the database partition list.

COMPLETE / STOP

Stops the rolling forward of log records, and completes the rollforward recovery process by rolling back any incomplete transactions and turning off the rollforward pending state of the database. This allows access to the database or table spaces that are being rolled forward. These keywords are equivalent; specify one or the other, but not both. The keyword AND permits specification of multiple operations at once; for example, `db2 rollforward db sample to end of logs and complete`.

Note: When rolling table spaces forward to a point in time, the table spaces are placed in backup pending state.

CANCEL

Cancels the rollforward recovery operation. This puts the database or one or more table spaces on all partitions on which forward recovery has been started in restore pending state:

- If a *database* rollforward operation is not in progress (that is, the database is in rollforward pending state), this option puts the database in restore pending state.
- If a *table space* rollforward operation is not in progress (that is, the table spaces are in rollforward pending state), a table space list must be specified. All table spaces in the list are put in restore pending state.
- If a table space rollforward operation *is* in progress (that is, at least one table space is in rollforward in progress state), all table spaces that are in rollforward in progress state are put in restore pending state. If a table space list is specified, it must include all table spaces that are in rollforward in progress state. All table spaces on the list are put in restore pending state.
- If rolling forward to a point in time, any table space name that is passed in is ignored, and all table spaces that are in rollforward in progress state are put in restore pending state.
- If rolling forward to the end of the logs with a table space list, only the table spaces listed are put in restore pending state.

This option cannot be used to cancel a rollforward operation *that is actually running*. It can only be used to cancel a rollforward operation that is in progress but not actually running at the time. A rollforward operation can be in progress but not running if:

- It terminated abnormally.
- The STOP option was not specified.
- An error caused it to fail. Some errors, such as rolling forward through a non-recoverable load operation, can put a table space into restore pending state.

Note: Use this option with caution, and only if the rollforward operation that is in progress cannot be completed because some of the table spaces have been put in rollforward pending state or in restore pending state. When in doubt, use the LIST TABLESPACES command to identify the table spaces that are in rollforward in progress state, or in rollforward pending state.

QUERY STATUS

Lists the log files that the database manager has rolled forward, the next archive file required, and the time stamp (in CUT) of the last committed transaction since rollforward processing began. In a partitioned database environment, this status information is returned for each partition. The information returned contains the following fields:

Database partition number

Rollforward status

Status can be: database or table space rollforward pending, database or table space rollforward in progress, database or table space rollforward processing STOP, or not pending.

Next log file to be read

A string containing the name of the next required log file. In a partitioned database environment, use this information if the rollforward utility fails with a return code indicating that a log file is missing or that a log information mismatch has occurred.

Log files processed

A string containing the names of processed log files that are no longer needed for recovery, and that can be removed from the directory. If, for example, the oldest uncommitted transaction starts in log file *x*, the range of obsolete log files will not include *x*; the range ends at *x* - 1.

Last committed transaction

A string containing a time stamp in ISO format (*yyyy-mm-dd-hh.mm.ss*). This time stamp marks the last transaction committed after the completion of rollforward recovery. The time stamp applies to the database. For table space rollforward recovery, it is the time stamp of the last transaction committed to the database.

Note: QUERY STATUS is the default value if the TO, STOP, COMPLETE, or CANCEL clauses are omitted. If TO, STOP, or COMPLETE was specified, status information is displayed if the command has completed successfully. If individual table spaces are specified, they are ignored; the status request does not apply only to specified table spaces.

TABLESPACE

This keyword is specified for table space-level rollforward recovery.

tablespace-name

Mandatory for table space-level rollforward recovery to a point in time. Allows a subset of table spaces to be specified for rollforward recovery to the end of the logs. In a partitioned database environment, each table space in the list does not have to exist at each partition that is rolling forward. If it *does* exist, it must be in the correct state.

ONLINE

This keyword is specified to allow table space-level rollforward recovery to be done online. This means that other agents are allowed to connect while rollforward recovery is in progress.

OVERFLOW LOG PATH log-directory

Specifies an alternate log path to be searched for archived logs during recovery. Use this parameter if log files were moved to a location other than that specified by the *logpath* database configuration parameter. In a partitioned database environment, this is the (fully qualified) default overflow log path *for all partitions*. A relative overflow log path can be specified for single-partition databases.

Note: The OVERFLOW LOG PATH command parameter will overwrite the value (if any) of the database configuration parameter OVERFLOWLOGPATH.

log-directory ON DBPARTITIONNUM

In a partitioned database environment, allows a different log path to override the default overflow log path for a specific partition.

NORETRIEVE

Allows the user to control which log files to be rolled forward on the standby machine by allowing the user to disable the retrieval of archived logs. The benefits of this are:

- By controlling the logfiles to be rolled forward, one can ensure that the standby machine is X hours behind the production machine, to prevent the user affecting both systems.

- If the standby system does not have access to archive (eg. if TSM is the archive, it only allows the original machine to retrieve the files)
- It might also be possible that while the production system is archiving a file, the standby system is retrieving the same file, and it might then get an incomplete log file. Noretrieve would solve this problem.

RECOVER DROPPED TABLE drop-table-id

Recovers a dropped table during the rollforward operation. The table ID can be obtained using the LIST HISTORY command.

TO export-directory

Specifies a directory to which files containing the table data are to be written. The directory must be accessible to all database partitions.

Examples:

Example 1

The ROLLFORWARD DATABASE command permits specification of multiple operations at once, each being separated with the keyword AND. For example, to roll forward to the end of logs, and complete, the separate commands:

```
db2 rollforward db sample to end of logs
db2 rollforward db sample complete
```

can be combined as follows:

```
db2 rollforward db sample to end of logs and complete
```

Although the two are equivalent, it is recommended that such operations be done in two steps. It is important to verify that the rollforward operation has progressed as expected, before stopping it and possibly missing logs. This is especially important if a bad log is found during rollforward recovery, and the bad log is interpreted to mean the “end of logs”. In such cases, an undamaged backup copy of that log could be used to continue the rollforward operation through more logs.

Example 2

Roll forward to the end of the logs (two table spaces have been restored):

```
db2 rollforward db sample to end of logs
db2 rollforward db sample to end of logs and stop
```

These two statements are equivalent. Neither AND STOP or AND COMPLETE is needed for table space rollforward recovery to the end of the logs. Table space names are not required. If not specified, all table spaces requiring rollforward recovery will be included. If only a subset of these table spaces is to be rolled forward, their names must be specified.

Example 3

After three table spaces have been restored, roll one forward to the end of the logs, and the other two to a point in time, both to be done online:

```
db2 rollforward db sample to end of logs tablespace(TBS1) online

db2 rollforward db sample to 1998-04-03-14.21.56.245378 and stop
   tablespace(TBS2, TBS3) online
```

Note that two rollforward operations cannot be run concurrently. The second command can only be invoked after the first rollforward operation completes successfully.

Example 4

After restoring the database, roll forward to a point in time, using OVERFLOW LOG PATH to specify the directory where the user exit saves archived logs:

```
db2 rollforward db sample to 1998-04-03-14.21.56.245378 and stop
   overflow log path (/logs)
```

Example 5 (MPP)

There are three database partitions: 0, 1, and 2. Table space TBS1 is defined on all partitions, and table space TBS2 is defined on partitions 0 and 2. After restoring the database on database partition 1, and TBS1 on database partitions 0 and 2, roll the database forward on database partition 1:

```
db2 rollforward db sample to end of logs and stop
```

This returns warning SQL1271 (“Database is recovered but one or more table spaces are off-line on database partition(s) 0 and 2.”).

```
db2 rollforward db sample to end of logs
```

This rolls TBS1 forward on database partitions 0 and 2. The clause TABLESPACE(TBS1) is optional in this case.

Example 6 (MPP)

After restoring table space TBS1 on database partitions 0 and 2 only, roll TBS1 forward on database partitions 0 and 2:

```
db2 rollforward db sample to end of logs
```

Database partition 1 is ignored.

```
db2 rollforward db sample to end of logs tablespace(TBS1)
```

This fails, because TBS1 is not ready for rollforward recovery on database partition 1. Reports SQL4906N.

```
db2 rollforward db sample to end of logs on dbpartitionnums (0, 2)
   tablespace(TBS1)
```

This completes successfully.

```
db2 rollforward db sample to 1998-04-03-14.21.56.245378 and stop
   tablespace(TBS1)
```

This fails, because TBS1 is not ready for rollforward recovery on database partition 1; all pieces must be rolled forward together.

Note: With table space rollforward to a point in time, the database partition clause is not accepted. The rollforward operation must take place on all the database partitions on which the table space resides.

After restoring TBS1 on database partition 1:

```
db2 rollforward db sample to 1998-04-03-14.21.56.245378 and stop
   tablespace(TBS1)
```

This completes successfully.

Example 7 (partitioned database environment)

After restoring a table space on all database partitions, roll forward to PIT2, but do not specify AND STOP. The rollforward operation is still in progress. Cancel and roll forward to PIT1:

```
   db2 rollforward db sample to pit2 tablespace(TBS1)
   db2 rollforward db sample cancel tablespace(TBS1)

 ** restore TBS1 on all database partitions **

   db2 rollforward db sample to pit1 tablespace(TBS1)
   db2 rollforward db sample stop tablespace(TBS1)
```

Example 8 (MPP)

Rollforward recover a table space that resides on eight database partitions (3 to 10) listed in the db2nodes.cfg file:

```
db2 rollforward database dwtest to end of logs tablespace (tssprodt)
```

This operation to the end of logs (not point in time) completes successfully. The database partitions on which the table space resides do not have to be specified. The utility defaults to the db2nodes.cfg file.

Example 9 (partitioned database environment)

Rollforward recover six small table spaces that reside on a single-partition database partition group (on database partition 6):

```
db2 rollforward database dwtest to end of logs on dbpartitionnum (6)
   tablespace(tsstore, tssbuyer, tsstime, tsswhse, tsslscat, tssvendor)
```

This operation to the end of logs (not point in time) completes successfully.

Usage notes:

If restoring from an image that was created during an online backup operation, the specified point in time for the rollforward operation must be later than the time at which the online backup operation completed. If the rollforward operation is stopped before it passes this point, the database is left in rollforward pending state. If a table space is in the process of being rolled forward, it is left in rollforward in progress state.

If one or more table spaces is being rolled forward to a point in time, the rollforward operation must continue at least to the minimum recovery time, which is the last update to the system catalogs for this table space or its tables. The minimum recovery time (in Coordinated Universal Time, or UTC) for a table space can be retrieved using the LIST TABLESPACES SHOW DETAIL command.

Rolling databases forward may require a load recovery using tape devices. If prompted for another tape, the user can respond with one of the following:

c Continue. Continue using the device that generated the warning message (for example, when a new tape has been mounted)

d Device terminate. Stop using the device that generated the warning message (for example, when there are no more tapes)

t Terminate. Terminate all devices.

If the rollforward utility cannot find the next log that it needs, the log name is returned in the SQLCA, and rollforward recovery stops. If no more logs are available, use the STOP option to terminate rollforward recovery. Incomplete transactions are rolled back to ensure that the database or table space is left in a consistent state.

Compatibilities:

For compatibility with versions earlier than Version 8:

- The keyword NODE can be substituted for DBPARTITIONNUM.
- The keyword NODES can be substituted for DBPARTITIONNUMS.

Related reference:

- "BACKUP DATABASE" on page 72
- "RESTORE DATABASE" on page 95

db2Rollforward - Rollforward Database

Recovers a database by applying transactions recorded in the database log files. Called after a database or a table space backup has been restored, or if any table spaces have been taken offline by the database due to a media error. The database must be recoverable (that is, either *logretain*, *userexit*, or both of these database configuration parameters must be set on) before the database can be recovered with roll-forward recovery.

Scope:

In a partitioned database environment, this API can only be called from the catalog partition. A database or table space rollforward call specifying a point-in-time affects all database partition servers that are listed in the `db2nodes.cfg` file. A database or table space rollforward call specifying end of logs affects the database partition servers that are specified. If no database partition servers are specified, it affects all database partition servers that are listed in the `db2nodes.cfg` file; if no roll forward is needed on a particular database partition server, that database partition server is ignored.

Authorization:

One of the following:

- *sysadm*
- *sysctrl*
- *sysmaint*

Required connection:

None. This API establishes a database connection.

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2Rollforward */
/* ... */
SQL_API_RC SQL_API_FN
db2Rollforward_api (
  db2Uint32 versionNumber,
  void *pDB2RollforwardStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2RollforwardStruct
{
  struct db2RfwdInputStruct   *roll_input;
  struct db2RfwdOutputStruct  *roll_output;
} db2RollforwardStruct;

typedef SQL_STRUCTURE db2RfwdInputStruct
{
  sqluint32                   version;
  char                        *pDbAlias;
  db2Uint32                   CallerAction;
  char                        *pStopTime;
  char                        *pUserName;
  char                        *pPassword;
  char                        *pOverflowLogPath;
  db2Uint32                   NumChngLgOvrflw;
  struct sqlurf_newlogpath    *pChngLogOvrflw;
  db2Uint32                   ConnectMode;
  struct sqlu_tablespace_bkrst_list *pTablespaceList;
  db2int32                    AllNodeFlag;
  db2int32                    NumNodes;
  SQL_PDB_NODE_TYPE           *pNodeList;
  db2int32                    NumNodeInfo;
  char                        *pDroppedTblID;
  char                        *pExportDir;
  db2Uint32                   RollforwardFlags;
} db2RfwdInputStruct;

typedef SQL_STRUCTURE db2RfwdOutputStruct
{
  char                        *pApplicationId;
  sqlint32                    *pNumReplies;
  struct sqlurf_info          *pNodeInfo;
} db2RfwdOutputStruct;
/* ... */
```

Generic API syntax:

```
/* File: db2ApiDf.h */
/* API: db2Rollforward */
/* ... */
SQL_API_RC SQL_API_FN
db2gRollforward_api (
  db2Uint32 versionNumber,
  void *pDB2gRollforwardStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2gRollforwardStruct
{
  struct db2gRfwdInputStruct  *roll_input;
  struct db2RfwdOutputStruct  *roll_output;
} db2gRollforwardStruct;

SQL_STRUCTURE db2gRfwdInputStruct
{
  db2Uint32                   DbAliasLen;
  db2Uint32                   StopTimeLen;
  db2Uint32                   UserNameLen;
  db2Uint32                   PasswordLen;
  db2Uint32                   OvrflwLogPathLen;
  db2Uint32                   DroppedTblIDLen;
  db2Uint32                   ExportDirLen;
  sqluint32                   Version;
  char                        *pDbAlias;
  db2Uint32                   CallerAction;
  char                        *pStopTime;
  char                        *pUserName;
  char                        *pPassword;
  char                        *pOverflowLogPath;
  db2Uint32                   NumChngLgOvrflw;
  struct sqlurf_newlogpath    *pChngLogOvrflw;
  db2Uint32                   ConnectMode;
  struct sqlu_tablespace_bkrst_list *pTablespaceList;
  db2int32                    AllNodeFlag;
  db2int32                    NumNodes;
  SQL_PDB_NODE_TYPE           *pNodeList;
  db2int32                    NumNodeInfo;
  char                        *pDroppedTblID;
  char                        *pExportDir;
  db2Uint32                   RollforwardFlags;
};

typedef SQL_STRUCTURE db2RfwdOutputStruct
{
  char                        *pApplicationId;
  sqlint32                    *pNumReplies;
  struct sqlurf_info          *pNodeInfo;
} db2RfwdOutputStruct;
/* ... */
```

API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter.

pDB2RollforwardStruct
Input. A pointer to the *db2RollforwardStruct* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

roll_input
Input. A pointer to the *db2RfwdInputStruct* structure.

roll_output
Output. A pointer to the *db2RfwdOutputStruct* structure.

DbAliasLen
Input. Specifies the length in bytes of the database alias.

StopTimeLen
Input. Specifies the length in bytes of the stop time parameter. Set to zero if no stop time is provided.

UserNameLen
Input. Specifies the length in bytes of the user name. Set to zero if no user name is provided.

PasswordLen
Input. Specifies the length in bytes of the password. Set to zero if no password is provided.

OverflowLogPathLen
Input. Specifies the length in bytes of the overflow log path. Set to zero if no overflow log path is provided.

Version
Input. The version ID of the rollforward parameters. It is defined as SQLUM_RFWD_VERSION.

pDbAlias
Input. A string containing the database alias. This is the alias that is cataloged in the system database directory.

CallerAction
Input. Specifies action to be taken. Valid values (defined in `sqlutil`) are:

SQLUM_ROLLFWD
Rollforward to the point in time specified by *pPointInTime*. For database rollforward, the database is left in *rollforward-pending* state. For table space rollforward to a point in time, the table spaces are left in *rollforward-in-progress* state.

SQLUM_STOP

End roll-forward recovery. No new log records are processed and uncommitted transactions are backed out. The *rollforward-pending* state of the database or table spaces is turned off. Synonym is SQLUM_COMPLETE.

SQLUM_ROLLFWD_STOP

Rollforward to the point in time specified by *pPointInTime*, and end roll-forward recovery. The *rollforward-pending* state of the database or table spaces is turned off. Synonym is SQLUM_ROLLFWD_COMPLETE.

SQLUM_QUERY

Query values for *pNextArcFileName*, *pFirstDelArcFileName*, *pLastDelArcFileName*, and *pLastCommitTime*. Return database status and a node number.

SQLUM_PARM_CHECK

Validate parameters without performing the roll forward.

SQLUM_CANCEL

Cancel the rollforward operation that is currently running. The database or table space are put in recovery pending state.

Note: This option cannot be used while the rollforward is actually running. It can be used if the rollforward is paused (that is, waiting for a STOP), or if a system failure occurred during the rollforward. It should be used with caution.

Rolling databases forward may require a load recovery using tape devices. The rollforward API will return with a warning message if user intervention on a device is required. The API can be called again with one of the following three caller actions:

SQLUM_LOADREC_CONTINUE

Continue using the device that generated the warning message (for example, when a new tape has been mounted).

SQLUM_LOADREC_DEVICE_TERMINATE

Stop using the device that generated the warning message (for example, when there are no more tapes).

SQLUM_LOADREC_TERMINATE

Terminate all devices being used by load recovery.

pStopTime

Input. A character string containing a time stamp in ISO format. Database recovery will stop when this time stamp is exceeded. Specify SQLUM_INFINITY_TIMESTAMP to roll forward as far as possible.

May be NULL for SQLUM_QUERY, SQLUM_PARM_CHECK, and any of the load recovery (SQLUM_LOADREC_*xxx*) caller actions.

pUserName
Input. A string containing the user name of the application. May be NULL.

pPassword
Input. A string containing the password of the supplied user name (if any). May be NULL.

pOverflowLogPath
Input. This parameter is used to specify an alternate log path to be used. In addition to the active log files, archived log files need to be moved (by the user) into the *logpath* before they can be used by this utility. This can be a problem if the user does not have sufficient space in the *logpath*. The overflow log path is provided for this reason. During roll-forward recovery, the required log files are searched, first in the *logpath*, and then in the overflow log path. The log files needed for table space roll-forward recovery can be brought into either the *logpath* or the overflow log path. If the caller does not specify an overflow log path, the default value is the *logpath*. In a partitioned database environment, the overflow log path must be a valid, fully qualified path; the default path is the default overflow log path for each node. In a single-partition database environment, the overflow log path can be relative if the server is local.

NumChngLgOvrflw
Partitioned database environments only. The number of changed overflow log paths. These new log paths override the default overflow log path for the specified database partition server only.

pChngLogOvrflw
Partitioned database environments only. A pointer to a structure containing the fully qualified names of changed overflow log paths. These new log paths override the default overflow log path for the specified database partition server only.

ConnectMode
Input. Valid values (defined in sqlutil) are:

SQLUM_OFFLINE
Offline roll forward. This value must be specified for database roll-forward recovery.

SQLUM_ONLINE
Online roll forward.

pTablespaceList
Input. A pointer to a structure containing the names of the table

spaces to be rolled forward to the end-of-logs or to a specific point in time. If not specified, the table spaces needing rollforward will be selected.

AllNodeFlag
Partitioned database environments only. Input. Indicates whether the rollforward operation is to be applied to all database partition servers defined in `db2nodes.cfg`. Valid values are:

SQLURF_NODE_LIST
Apply to database partition servers in a list that is passed in *pNodeList*.

SQLURF_ALL_NODES
Apply to all database partition servers. *pNodeList* should be NULL. This is the default value.

SQLURF_ALL_EXCEPT
Apply to all database partition servers except those in a list that is passed in *pNodeList*.

SQLURF_CAT_NODE_ONLY
Apply to the catalog partition only. *pNodeList* should be NULL.

NumNodes
Input. Specifies the number of database partition servers in the *pNodeList* array.

pNodeList
Input. A pointer to an array of database partition server numbers on which to perform the roll-forward recovery.

NumNodeInfo
Input. Defines the size of the output parameter *pNodeInfo*, which must be large enough to hold status information from each database partition that is being rolled forward. In a single-partition database environment, this parameter should be set to 1. The value of this parameter should be same as the number of database partition servers for which this API is being called.

pDroppedTblID
Input. A string containing the ID of the dropped table whose recovery is being attempted.

pExportDir
Input. The directory into which the dropped table data will be exported.

RollforwardFlags
Input. Specifies the rollforward flags. Valid values (defined in `sqlpapiRollforward`):

SQLP_ROLLFORWARD_LOCAL_TIME
Allows the user to rollforward to a point in time that is the user's local time rather than GMT time. This makes it easier for users to rollforward to a specific point in time on their local machines, and eliminates potential user errors due to the translation of local to GMT time.

SQLP_ROLLFORWARD_NO_RETRIEVE
Controls which log files to be rolled forward on the standby machine by allowing the user to disable the retrieval of archived logs. By controlling the log files to be rolled forward, one can ensure that the standby machine is `X` hours behind the production machine, to prevent the user affecting both systems. This option is useful if the standby system does not have access to archive, for example, if TSM is the archive, it only allows the original machine to retrieve the files. It will also remove the possibility that the standby system would retrieve an incomplete log file while the production system is archiving a file and the standby system is retrieving the same file.

pApplicationId
Output. The application ID.

pNumReplies
Output. The number of replies received.

pNodeInfo
Output. Database partition reply information.

REXX API syntax:

```
ROLLFORWARD DATABASE database-alias [USING :value] [USER username
USING password]
[rollforward_action_clause | load_recovery_action_clause]
where rollforward_action_clause stands for:
    { TO point-in-time [AND STOP] |
      {
        [TO END OF LOGS [AND STOP] | STOP | CANCEL | QUERY STATUS
        | PARM CHECK }
        [ON {:nodelist | ALL NODES [EXCEPT :nodelist]}]
      }
    }
    [TABLESPACE {ONLINE |:tablespacenames [ONLINE]} ]
    [OVERFLOW LOG PATH default-log-path [:logpaths]]
and load_recovery_action_clause stands for:
    LOAD RECOVERY { CONTINUE | DEVICE_TERMINATE | TERMINATE }
```

REXX API parameters:

database-alias
Alias of the database to be rolled forward.

value A compound REXX host variable containing the output values. In the following, XXX represents the host variable name:

XXX.0	Number of elements in the variable
XXX.1	The application ID
XXX.2	Number of replies received from nodes
XXX.2.1.1	First database partition server number
XXX.2.1.2	First state information
XXX.2.1.3	First next archive file needed
XXX.2.1.4	First first archive file to be deleted
XXX.2.1.5	First last archive file to be deleted
XXX.2.1.6	First last commit time
XXX.2.2.1	Second database partition server number
XXX.2.2.2	Second state information
XXX.2.2.3	Second next archive file needed
XXX.2.2.4	Second first archive file to be deleted
XXX.2.2.5	Second last archive file to be deleted
XXX.2.2.6	Second last commit time
XXX.2.3.x	and so on.

username
Identifies the user name under which the database is to be rolled forward.

password
The password used to authenticate the user name.

point-in-time
A time stamp in ISO format, *yyyy-mm-dd-hh.mm.ss.nnnnnn* (year, month, day, hour, minutes, seconds, microseconds), expressed in Coordinated Universal Time (UTC).

tablespacenames
A compound REXX host variable containing a list of table spaces to be rolled forward. In the following, XXX is the name of the host variable:

XXX.0	Number of table spaces to be rolled forward
XXX.1	First table space name
XXX.2	Second table space name
XXX.x	and so on.

default-log-path
The default overflow log path to be searched for archived logs during recovery

logpaths
A compound REXX host variable containing a list of alternate log paths to be searched for archived logs during recovery. In the following, XXX is the name of the host variable:

XXX.0	Number of changed overflow log paths
XXX.1.1	First node
XXX.1.2	First overflow log path
XXX.2.1	Second node
XXX.2.2	Second overflow log path
XXX.3.1	Third node
XXX.3.2	Third overflow log path
XXX.x.1	and so on.

nodelist
A compound REXX host variable containing a list of database partition servers. In the following, XXX is the name of the host variable:

XXX.0	Number of nodes

XXX.1	First node
XXX.2	Second node
XXX.x	and so on.

Usage notes:

The database manager uses the information stored in the archived and the active log files to reconstruct the transactions performed on the database since its last backup.

The action performed when this API is called depends on the *rollforward_pending* flag of the database prior to the call. This can be queried using "db2CfgGet - Get Configuration Parameters" The *rollforward_pending* flag is set to `DATABASE` if the database is in roll-forward pending state. It is set to `TABLESPACE` if one or more table spaces are in `SQLB_ROLLFORWARD_PENDING` or `SQLB_ROLLFORWARD_IN_PROGRESS` state. The *rollforward_pending* flag is set to `NO` if neither the database nor any of the table spaces needs to be rolled forward.

If the database is in roll-forward pending state when this API is called, the database will be rolled forward. Table spaces are returned to normal state after a successful database roll-forward, unless an abnormal state causes one or more table spaces to go offline. If the *rollforward_pending* flag is set to `TABLESPACE`, only those table spaces that are in roll-forward pending state, or those table spaces requested by name, will be rolled forward.

Note: If table space rollforward terminates abnormally, table spaces that were being rolled forward will be put in `SQLB_ROLLFORWARD_IN_PROGRESS` state. In the next invocation of ROLLFORWARD DATABASE, only those table spaces in `SQLB_ROLLFORWARD_IN_PROGRESS` state will be processed. If the set of selected table space names does not include all table spaces that are in `SQLB_ROLLFORWARD_IN_PROGRESS` state, the table spaces that are not required will be put into `SQLB_RESTORE_PENDING` state.

If the database is not in roll-forward pending state and no point in time is specified, any table spaces that are in rollforward-in-progress state will be rolled forward to the end of logs. If no table spaces are in rollforward-in-progress state, any table spaces that are in rollforward pending state will be rolled forward to the end of logs.

This API reads the log files, beginning with the log file that is matched with the backup image. The name of this log file can be determined by calling this API with a caller action of `SQLUM_QUERY` before rolling forward any log files.

The transactions contained in the log files are reapplied to the database. The log is processed as far forward in time as information is available, or until the time specified by the stop time parameter.

Recovery stops when any one of the following events occurs:

- No more log files are found
- A time stamp in the log file exceeds the completion time stamp specified by the stop time parameter
- An error occurs while reading the log file.

Some transactions might not be recovered. The value returned in *pLastCommitTime* indicates the time stamp of the last committed transaction that was applied to the database.

If the need for database recovery was caused by application or human error, the user may want to provide a time stamp value in *pStopTime*, indicating that recovery should be stopped before the time of the error. This applies only to full database roll-forward recovery, and to table space rollforward to a point in time. It also permits recovery to be stopped before a log read error occurs, determined during an earlier failed attempt to recover.

When the *rollforward_recovery* flag is set to `DATABASE`, the database is not available for use until roll-forward recovery is terminated. Termination is accomplished by calling the API with a caller action of `SQLUM_STOP` or `SQLUM_ROLLFORWARD_STOP` to bring the database out of roll-forward pending state. If the *rollforward_recovery* flag is `TABLESPACE`, the database is available for use. However, the table spaces in `SQLB_ROLLFORWARD_PENDING` and `SQLB_ROLLFORWARD_IN_PROGRESS` states will not be available until the API is called to perform table space roll-forward recovery. If rolling forward table spaces to a point in time, the table spaces are placed in backup pending state after a successful rollforward.

When the *RollforwardFlags* option is set to `SQLP_ROLLFORWARD_LOCAL_TIME`, all messages returned to the user will also be in local time. All times are converted on the server, and on the catalog partition, if it is a partitioned database environment. The timestamp string is converted to GMT on the server, so the time is local to the server's time zone, not the client's. If the client is in one time zone and the server in another, the server's local time should be used. This is different from the local time option from the Control Center, which is local to the client. If the timestamp string is close to the time change of the clock due to daylight savings, it is important to know if the stop time is before or after the clock change, and specify it correctly.

Related reference:

- "SQLCA" in the *Administrative API Reference*

Related samples:

- "dbrecov.sqc -- How to recover a database (C)"
- "dbrecov.sqC -- How to recover a database (C++)"

Rollforward Sessions - CLP Examples

Example 1

The ROLLFORWARD DATABASE command permits specification of multiple operations at once, each being separated with the keyword AND. For example, to roll forward to the end of logs, and complete, the separate commands:

```
db2 rollforward db sample to end of logs
db2 rollforward db sample complete
```

can be combined as follows:

```
db2 rollforward db sample to end of logs and complete
```

Although the two are equivalent, it is recommended that such operations be done in two steps. It is important to verify that the rollforward operation has progressed as expected, before stopping it and possibly missing logs. This is especially important if a bad log is found during rollforward recovery, and the bad log is interpreted to mean the "end of logs". In such cases, an undamaged backup copy of that log could be used to continue the rollforward operation through more logs.

Example 2

Roll forward to the end of the logs (two table spaces have been restored):

```
db2 rollforward db sample to end of logs
db2 rollforward db sample to end of logs and stop
```

These two statements are equivalent. Neither AND STOP or AND COMPLETE is needed for table space rollforward recovery to the end of the logs. Table space names are not required. If not specified, all table spaces requiring rollforward recovery will be included. If only a subset of these table spaces is to be rolled forward, their names must be specified.

Example 3

After three table spaces have been restored, roll one forward to the end of the logs, and the other two to a point in time, both to be done online:

```
db2 rollforward db sample to end of logs tablespace(TBS1) online

db2 rollforward db sample to 1998-04-03-14.21.56.245378 and stop
   tablespace(TBS2, TBS3) online
```

Note that two rollforward operations cannot be run concurrently. The second command can only be invoked after the first rollforward operation completes successfully.

Example 4

After restoring the database, roll forward to a point in time, using OVERFLOW LOG PATH to specify the directory where the user exit saves archived logs:

```
db2 rollforward db sample to 1998-04-03-14.21.56.245378 and stop
   overflow log path (/logs)
```

Example 5 (MPP)

There are three nodes: 0, 1, and 2. Table space TBS1 is defined on all nodes, and table space TBS2 is defined on nodes 0 and 2. After restoring the database on node 1, and TBS1 on nodes 0 and 2, roll the database forward on node 1:

```
db2 rollforward db sample to end of logs and stop
```

This returns warning SQL1271 ("Database is recovered but one or more table spaces are offline on node(s) 0 and 2.").

```
db2 rollforward db sample to end of logs
```

This rolls TBS1 forward on nodes 0 and 2. The clause TABLESPACE(TBS1) is optional in this case.

Example 6 (MPP)

After restoring table space TBS1 on nodes 0 and 2 only, roll TBS1 forward on nodes 0 and 2:

```
db2 rollforward db sample to end of logs
```

Node 1 is ignored.

```
db2 rollforward db sample to end of logs tablespace(TBS1)
```

This fails, because TBS1 is not ready for rollforward recovery on node 1. Reports SQL4906N.

```
db2 rollforward db sample to end of logs on nodes (0, 2) tablespace(TBS1)
```

This completes successfully.

```
db2 rollforward db sample to 1998-04-03-14.21.56.245378 and stop
   tablespace(TBS1)
```

This fails, because TBS1 is not ready for rollforward recovery on node 1; all pieces must be rolled forward together.

Note: With table space rollforward to a point in time, the node clause is not accepted. The rollforward operation must take place on all the nodes on which the table space resides.

After restoring TBS1 on node 1:

```
db2 rollforward db sample to 1998-04-03-14.21.56.245378 and stop
   tablespace(TBS1)
```

This completes successfully.

Example 7 (MPP)

After restoring a table space on all nodes, roll forward to PIT2, but do not specify `AND STOP`. The rollforward operation is still in progress. Cancel and roll forward to PIT1:

```
  db2 rollforward db sample to pit2 tablespace(TBS1)
  db2 rollforward db sample cancel tablespace(TBS1)

** restore TBS1 on all nodes **

  db2 rollforward db sample to pit1 tablespace(TBS1)
  db2 rollforward db sample stop tablespace(TBS1)
```

Example 8 (MPP)

Rollforward recover a table space that resides on eight nodes (3 to 10) listed in the `db2nodes.cfg` file:

```
db2 rollforward database dwtest to end of logs tablespace (tssprodt)
```

This operation to the end of logs (not point in time) completes successfully. The nodes on which the table space resides do not have to be specified. The utility defaults to the `db2nodes.cfg` file.

Example 9 (MPP)

Rollforward recover six small table spaces that reside on a single node database partition group (on node 6):

```
db2 rollforward database dwtest to end of logs on node (6)
   tablespace(tsstore, tssbuyer, tsstime, tsswhse, tsslscat, tssvendor)
```

This operation to the end of logs (not point in time) completes successfully.

Part 2. High Availability

Chapter 5. Introducing High Availability and Failover Support

Successful e-businesses depend on the uninterrupted availability of transaction processing systems, which in turn are driven by database management systems, such as DB2, that must be available 24 hours a day and 7 days a week (“24 x 7”). This section discusses the following:

- “High Availability”
- “High Availability through Online Split Mirror and Suspended I/O Support” on page 167
- “Fault Monitor Facility for UNIX Based Systems” on page 171
- “db2fm - DB2 Fault Monitor” on page 173

High Availability

High availability (HA) is the term that is used to describe systems that run and are available to customers more or less all the time. For this to occur:

- Transactions must be processed efficiently, without appreciable performance degradations (or even loss of availability) during peak operating periods. In a partitioned database environment, DB2® can take advantage of both intrapartition and interpartition parallelism to process transactions efficiently. *Intrapartition parallelism* can be used in an SMP environment to process the various components of a complex SQL statement simultaneously. *Interpartition parallelism* in a partitioned database environment, on the other hand, refers to the simultaneous processing of a query on all participating nodes; each node processes a subset of the rows in the table.
- Systems must be able to recover quickly when hardware or software failures occur, or when disaster strikes. DB2 has an advanced continuous checkpointing system and a parallel recovery capability that allow for extremely fast crash recovery.

  The ability to recover quickly can also depend on having a proven backup and recovery strategy in place.
- Software that powers the enterprise databases must be continuously running and available for transaction processing. To keep the database manager running, you must ensure that another database manager can take over if it fails. This is called failover. *Failover* capability allows for the automatic transfer of workload from one system to another when there is hardware failure.

Failover protection can be achieved by keeping a copy of your database on another machine that is perpetually rolling the log files forward. *Log shipping* is the process of copying whole log files to a standby machine, either from an archive device, or through a user exit program running against the primary database. With this approach, the primary database is restored to the standby machine, using either the DB2 restore utility or the split mirror function. You can use the new suspended I/O support to quickly initialize the new database. The secondary database on the standby machine continuously rolls the log files forward. If the primary database fails, any remaining log files are copied over to the standby machine. After a rollforward to the end of the logs and stop operation, all clients are reconnected to the secondary database on the standby machine.

Failover support can also be provided through platform-specific software that you can add to your system. For example:

- High Availability Cluster Multi-Processing, Enhanced Scalability, for AIX.

  For detailed information about HACMP/ES, see the white paper entitled "IBM® DB2 Universal Database™ Enterprise Edition for AIX® and HACMP/ES", which is available from the "DB2 UDB and DB2 Connect Online Support" web site (http://www.ibm.com/software/data/pubs/papers/).
- Microsoft® Cluster Server, for Windows® operating systems.

  For information about Microsoft Cluster Server see the following white papers which are available from the "DB2 UDB and DB2 Connect™ Online Support" web site (http://www.ibm.com/software/data/pubs/papers/): "Implementing IBM DB2 Universal Database Enterprise - Extended Edition with Microsoft Cluster Server", "Implementing IBM DB2 Universal Database Enterprise Edition with Microsoft Cluster Server", and "DB2 Universal Database for Windows: High Availability Support Using Microsoft Cluster Server - Overview".
- Sun Cluster, or VERITAS Cluster Server, for the Solaris Operating Environment.

  For information about Sun Cluster 3.0, see the white paper entitled "DB2 and High Availability on Sun Cluster 3.0", which is available from the "DB2 UDB and DB2 Connect Online Support" web site (http://www.ibm.com/software/data/pubs/papers/). For information about VERITAS Cluster Server, see the white paper entitled "DB2 and High Availability on VERITAS Cluster Server", which is also available from the "DB2 UDB and DB2 Connect Online Support" Web site.
- Multi-Computer/ServiceGuard, for Hewlett-Packard.

  For detailed information about HP MC/ServiceGuard, see the white paper which discusses IBM DB2 implementation and certification with Hewlett-Packard's MC/ServiceGuard high availability software, which is available from the "IBM Data Management Products for HP" web site (http://www.ibm.com/software/data/hp/pdfs/db2mc.pdf).

Failover strategies are usually based on clusters of systems. A *cluster* is a group of connected systems that work together as a single system. Each processor is known as a node within the cluster. Clustering allows servers to back each other up when failures occur, by picking up the workload of the failed server.

IP address takeover (or IP takeover) is the ability to transfer a server IP address from one machine to another when a server goes down; to a client application, the two machines appear at different times to be the same server.

Failover software may use *heartbeat monitoring* or *keepalive packets* between systems to confirm availability. Heartbeat monitoring involves system services that maintain constant communication between all the nodes in a cluster. If a heartbeat is not detected, failover to a backup system starts. End users are usually not aware that a system has failed.

The two most common failover strategies on the market are known as *idle standby* and *mutual takeover*, although the configurations associated with these terms may also be associated with different terms that depend on the vendor:

Idle Standby
In this configuration, one system is used to run a DB2 instance, and the second system is "idle", or in standby mode, ready to take over the instance if there is an operating system or hardware failure involving the first system. Overall system performance is not impacted, because the standby system is idle until needed.

Mutual Takeover
In this configuration, each system is the designated backup for another system. Overall system performance may be impacted, because the backup system must do extra work following a failover: it must do its own work plus the work that was being done by the failed system.

Failover strategies can be used to failover an instance, a partition, or multiple logical nodes.

Related concepts:

- "Parallelism" in the *Administration Guide: Planning*
- "Developing a Backup and Recovery Strategy" on page 3
- "High Availability through Online Split Mirror and Suspended I/O Support" on page 167
- "High Availability in the Solaris Operating Environment" on page 189

- “High Availability on Sun Cluster 3.0” on page 192
- “High Availability with VERITAS Cluster Server” on page 195
- Chapter 7, “High Availability on the Windows Operating System” on page 183
- Chapter 6, “High Availability on AIX” on page 177

High Availability through Log Shipping

Log shipping is the process of copying whole log files to a standby machine either from an archive device, or through a user exit program running against the primary database. The standby database is continuously rolling forward through the log files produced by the production machine. When the production machine fails, a failover occurs and the following takes place:

- The remaining logs are transferred over to the standby machine.
- The standby database rolls forward to the end of the logs and stops.
- The clients reconnect to the standby database and resume operations.

The standby machine has its own resources (i.e., disks), but must have the same physical and logical definitions as the production database. When using this approach the primary database is restored to the standby machine, by using restore utility or the split mirror function.

To ensure that you are able to recover your database in a disaster recovery situation consider the following:

- The archive location should be geographically separate from the primary site.
- Remotely mirror the log at the standby database site
- Use a synchronous mirror for no loss support. You can do this through DB2® log mirroring or modern disk subsystems such as ESS and EMC. NVRAM cache (both local and remote) is also recommended to minimize the performance impact of a disaster recovery situation.

Notes:

1. When the standby database processes a log record indicating that an index rebuild took place on the primary database, the indexes on the standby server are not automatically rebuilt. The index will be rebuilt on the standby server either at the first connection to the database, or at the first attempt to access the index after the standby server is taken out of rollforward pending state. It is recommended that the standby server be resynchronized with the primary server if any indexes on the primary server are rebuilt.
2. If the load utility is run on the primary database with the COPY YES option specified, the standby database must have access to the copy image.

3. If the load utility is run on the primary database with the COPY NO option specified, the standby database should be resynchronized, otherwise the table space will be placed in restore pending state.
4. There are two ways to initialize a standby machine:
   a. By restoring to it from a backup image.
   b. By creating a split mirror of the production system and issuing the **db2inidb** command with the STANDBY option.

   Only after the standby machine has been initialized can you issue the ROLLFORWARD command on the standby system.
5. Operations that are not logged will not be replayed on the standby database. As a result, it is recommended that you re-sync the standby database after such operations. You can do this through online split mirror and suspended I/O support.

Related concepts:

- "High Availability through Online Split Mirror and Suspended I/O Support" on page 167

Related tasks:

- "Using a Split Mirror as a Standby Database" on page 169

Related reference:

- Appendix G, "User Exit for Database Recovery" on page 323

High Availability through Online Split Mirror and Suspended I/O Support

Suspended I/O supports continuous system availability by providing a full implementation for online split mirror handling; that is, splitting a mirror without shutting down the database. A *split mirror* is an "instantaneous" copy of the database that can be made by mirroring the disks containing the data, and splitting the mirror when a copy is required. *Disk mirroring* is the process of writing all of your data to two separate hard disks; one is the mirror of the other. *Splitting* a mirror is the process of separating the primary and secondary copies of the database.

If you would rather not back up a large database using the DB2® backup utility, you can make copies from a mirrored image by using suspended I/O and the split mirror function. This approach also:

- Eliminates backup operation overhead from the production machine
- Represents a fast way to clone systems

- Represents a fast implementation of idle standby failover. There is no initial restore operation, and if a rollforward operation proves to be too slow, or encounters errors, reinitialization is very fast.

The **db2inidb** command initializes the split mirror so that it can be used:

- As a clone database
- As a standby database
- As a backup image

This command can only be issued against a split mirror, and it must be run before the split mirror can be used.

In a partitioned database environment, you do not have to suspend I/O writes on all partitions simultaneously. You can suspend a subset of one or more partitions to create split mirrors for performing offline backups. If the catalog node is included in the subset, it must be the last partition to be suspended.

In a partitioned database environment, the **db2inidb** command must be run on every partition before the split image from any of the partitions can be used. The tool can be run on all partitions simultaneously using the **db2_all** command.

Note: Ensure that the split mirror contains all containers and directories which comprise the database, including the volume directory.

Related reference:

- "db2inidb - Initialize a Mirrored Database" on page 220

Online Split Mirror Handling

Making a Clone Database

Restrictions:

You cannot back up a cloned database, restore the backup image on the original system, and roll forward through log files produced on the original system.

Procedure:

To clone a database, follow these steps:

1. Suspend I/O on the primary database:

   ```
   db2 set write suspend for database
   ```

2. Use appropriate operating system-level commands to split the mirror or mirrors from the primary database.
3. Resume I/O on the primary database:

   ```
   db2 set write resume for database
   ```
4. Attach to the mirrored database from another machine.
5. Start the database instance:

   ```
   db2start
   ```
6. Initialize the mirrored database as a clone of the primary database:

   ```
   db2inidb database_alias as snapshot
   ```

   Note: This command will roll back transactions that are in flight when the split occurs, and start a new log chain sequence so that any logs from the primary database cannot be replayed on the cloned database.

Related concepts:

- "High Availability through Online Split Mirror and Suspended I/O Support" on page 167

Related reference:

- "db2inidb - Initialize a Mirrored Database" on page 220

Using a Split Mirror as a Standby Database

Procedure:

To use a split mirror as a standby database, follow these steps:

1. Suspend I/O on the primary database:

   ```
   db2 set write suspend for database
   ```
2. Use appropriate operating system-level commands to split the mirror or mirrors from the primary database.
3. Resume I/O on the primary database:

   ```
   db2 set write resume for database
   ```
4. Attach the mirrored database to another instance.
5. Put the mirrored database in rollforward pending state:

   ```
   db2inidb database_alias as standby
   ```

   Note: If you have only DMS table spaces (database managed space), you can take a full database backup to offload the overhead of taking a backup on the production database.
6. Set up a user exit program to retrieve the log files from the primary system.

7. Roll the database forward to the end of the logs or to a point-in-time.
8. Continue retrieving log files, and rolling the database forward through the logs until you reach the end of the logs or the point-in-time required for the standby database.
9. To bring the standby database online issue the ROLLFORWARD command with the STOP option specified.

Related concepts:

- "High Availability through Online Split Mirror and Suspended I/O Support" on page 167

Related tasks:

- "Making a Clone Database" on page 168
- "Using a Split Mirror as a Backup Image" on page 170

Related reference:

- "db2inidb - Initialize a Mirrored Database" on page 220

Using a Split Mirror as a Backup Image

Procedure:

To use a split mirror as a "backup image", follow these steps:

1. Suspend I/O on the primary database:
   ```
   db2 set write suspend for database
   ```
2. Use appropriate operating system-level commands to split the mirror or mirrors from the primary database.
3. Resume I/O on the primary database:
   ```
   db2 set write resume for database
   ```
4. A failure occurs on the primary system, necessitating a restore from backup.
5. Stop the primary database instance:
   ```
   db2stop
   ```
6. Use operating system-level commands to copy the split-off data over the primary system. **Do not copy the split-off log files**, because the primary logs will be needed for rollforward recovery.
7. Start the primary database instance:
   ```
   db2start
   ```
8. Initialize the primary database:
   ```
   db2inidb database_alias as mirror
   ```
9. Roll the primary database forward to the end of the logs or to a point-in-time and stop.

Related concepts:

- "High Availability through Online Split Mirror and Suspended I/O Support" on page 167

Related tasks:

- "Making a Clone Database" on page 168
- "Using a Split Mirror as a Standby Database" on page 169

Related reference:

- "db2inidb - Initialize a Mirrored Database" on page 220

Fault Monitor Facility for UNIX Based Systems

On UNIX® based systems, the Fault Monitor Facility improves the availability of non-clustered DB2® environments through a sequence of processes that work together to ensure that DB2 is running. That is, the *init* daemen monitors the Fault Monitor Coordinator (FMC), the FMC monitors the fault monitors and the fault monitors monitor DB2.

The Fault Monitor Coordinator (FMC) is the process of the Fault Monitor Facility that is started at the UNIX boot sequence. The *init* daemon starts the FMC and will restart it if it terminates abnormally. The FMC starts one fault monitor for each DB2 instance. Each fault monitor runs as a daemon process and has the same user privileges as the DB2 instance. Once a fault monitor is started, it will be monitored to make sure it does not exit prematurely. If a fault monitor fails, it will be restarted by the FMC. Each fault monitor will, in turn, be responsible for monitoring one DB2 instance. If the DB2 instance exits prematurely, the fault monitor will restart it.

Notes:

1. If you are using a high availability clustering product (i.e., HACMP or MSCS), the fault monitor facility must be turned off since the instance startup and shut down is controlled by the clustering product.
2. The fault monitor will only become inactive if the **db2stop** command is issued. If a DB2 instance is shut down in any other way, the fault monitor will start it up again.

Fault Monitor Registry File

A fault monitor registry file is created for every instance on each physical machine when the fault monitor daemon is started. The values in this file specify the behavior of the fault monitors. The file can be found in the `/sqllib/` directory and is called `fm.<machine_name>.reg`. This file can be altered using the **db2fm** command. The entries are as follows:

```
FM_ON = no
FM_ACTIVE = yes
START_TIMEOUT = 600
STOP_TIMEOUT = 600
STATUS_TIMEOUT = 20
STATUS_INTERVAL = 20
RESTART_RETRIES = 3
ACTION_RETRIES = 3
NOTIFY_ADDRESS = <instance_name>@<machine_name>
```

where:

FM_ON
Specifies whether or not the fault monitor should be started. If the value is set to NO, the fault monitor daemon will not be started, or will be turned off if it had already been started. The default value is NO.

FM_ACTIVE
Specifies whether or note the fault monitor is active. The fault monitor will only take action if both FM_ON and FM_ACTIVE are set to YES. If FM_ON is set to YES and FM_ACTIVE is set to NO, the fault monitor daemon will be started, but it will not be active. That means that is will not try to bring DB2 back online if it shuts down. The default value is YES.

START_TIMEOUT
Specifies the amount of time within which the fault monitor must start the service it is monitoring. The default value is 600 seconds.

STOP_TIMEOUT
Specifies the amount of time within which the fault monitor must bring down the service it is monitoring. The default value is 600 seconds.

STATUS_TIMEOUT
Specifies the amount of time within which the fault monitor must get the status of the service it is monitoring. The default value is 20 seconds.

STATUS_INTERVAL
Specifies the minimum time between two consecutive calls to obtain the status of the service that is being monitored. The default value is 20 seconds.

RESTART_RETRIES
Specifies the number of times the fault monitor will try to obtain the status of the service being monitored after a failed attempt. Once this number is reached the fault monitor will take action to bring the service back online. The default value is 3.

ACTION_RETRIES
Specifies the number of times the fault monitor will attempt to bring the service back online. The default value is 3.

NOTIFY_ADDRESS
Specifies the e-mail address to which the fault monitor will send notification messages. The default is <instance_name>@<machine_name>)

This file can be altered using the **db2fm** command. For example:

To update the START_TIMEOUT value to 100 seconds for instance DB2INST1, issue:

```
db2fm -i db2inst1 -T 100
```

To update the STOP_TIMEOUT value to 200 seconds for instance DB2INST1, issue:

```
db2fm -i db2inst1 -T /200
```

To update the START_TIMEOUT value to 100 seconds and the STOP_TIMEOUT value to 200 seconds for instance DB2INST1, issue:

```
db2fm -i db2inst1 -T 100/200
```

To turn on fault monitoring for instance DB2INST1, issue:

```
db2fm -i db2inst1 -f yes
```

To turn off fault monitoring for instance DB2INST1, issue:

```
db2fm -i db2inst1 -f no
```

Note: If the fault monitor registry file does not exist, the default values will be used.

Related reference:

- "db2fm - DB2 Fault Monitor" on page 173

db2fm - DB2 Fault Monitor

Controls the DB2 fault monitor daemon. You can use db2fm to configure the fault monitor.

Authorization:

Authorization over the instance against which you are running the command.

Required Connection:

None.

Command Syntax:

```
►►──db2fm──┬─-t──service──┬──-m──module path──┬────────────┬──────────────────────►◄
           └─-i──instance─┘                   ├─-u─────────┤
                                              ├─-d─────────┤
                                              ├─-s─────────┤
                                              ├─-k─────────┤
                                              ├─-U─────────┤
                                              ├─-D─────────┤
                                              ├─-S─────────┤
                                              ├─-K─────────┤
                                              ├─-f──┬─on──┬─┤
                                              │     └─off─┘ │
                                              ├─-a──┬─on──┬─┤
                                              │     └─off─┘ │
                                              ├─-T──T1/T2──┤
                                              ├─-l──I1/I2──┤
                                              ├─-R──R1/R2──┤
                                              ├─-n──email──┤
                                              ├─-h─────────┤
                                              └─-?─────────┘
```

Command Parameters:

-m *module-path*
: Defines the full path of the fault monitor shared library for the product being monitored. The default is $INSTANCEHOME/sqllib/lib/libdb2gcf.

-t *service*
: Gives the unique text descriptor for a service.

-i *instance*
: Defines the instance of the service.

-u Brings the service up.

-U Brings the fault monitor daemon up.

-d Brings the service down.

-D Brings the fault monitor daemon down.

-k Kills the service.

-K Kills the fault monitor daemon.

-s Returns the status of the service.

-S Returns the status of the fault monitor daemon.

Note: the status of the service or fault monitor can be one of the following

- Not properly installed,
- INSTALLED PROPERLY but NOT ALIVE,
- ALIVE but NOT AVAILABLE (maintenance),
- AVAILABLE, or
- UNKNOWN

-f *on|off*
Turns fault monitor on or off.

Note: If this option is set off, the fault monitor daemon will not be started, or the daemon will exit if it was running.

-a *on|off*
Activates or deactivate fault monitoring.

Note: If this option if set off, the fault monitor will not be actively monitoring, which means if the service goes down it will not try to bring it back.

-T *T1/T2*
Overwrites the start and stop time-out.

e.g.

- -T 15/10 updates the two time-outs respectively
- -T 15 updates the start time-out to 15 secs
- -T /10 updates the stop time-out to 10 secs

-I *I1/I2*
Sets the status interval and time-out respectively.

-R *R1/R2*
Sets the number of retries for the status method and action before giving up.

-n *email*
Sets the email address for notification of events.

-h Prints usage.

-? Prints usage.

Usage Notes:

1. This command may be used on UNIX platforms only.

Chapter 6. High Availability on AIX

Enhanced Scalability (ES) is a feature of High Availability Cluster Multi-Processing (HACMP) for AIX. This feature provides the same failover recovery and has the same event structure as HACMP. Enhanced scalability also provides:

- Larger HACMP clusters.
- Additional error coverage through *user-defined events*. Monitored areas can trigger user-defined events, which can be as diverse as the death of a process, or the fact that paging space is nearing capacity. Such events include pre- and post-events that can be added to the failover recovery process, if needed. Extra functions that are specific to the different implementations can be placed within the HACMP pre- and post-event streams.

  A *rules file* (`/usr/sbin/cluster/events/rules.hacmprd`) contains the HACMP events. User-defined events are added to this file. The script files that are to be run when events occur are part of this definition.
- HACMP client utilities for monitoring and detecting status changes (in one or more clusters) from AIX® physical nodes outside of the HACMP cluster.

The nodes in HACMP ES clusters exchange messages called *heartbeats*, or *keepalive packets*, by which each node informs the other nodes about its availability. A node that has stopped responding causes the remaining nodes in the cluster to invoke recovery. The recovery process is called a *node_down event* and may also be referred to as *failover*. The completion of the recovery process is followed by the re-integration of the node into the cluster. This is called a *node_up event*.

There are two types of events: standard events that are anticipated within the operations of HACMP ES, and user-defined events that are associated with the monitoring of parameters in hardware and software components.

One of the standard events is the node_down event. When planning what should be done as part of the recovery process, HACMP allows two failover options: hot (or idle) standby, and mutual takeover.

Note: When using HACMP, ensure that DB2® instances are not started at boot time by using the **db2iauto** utility as follows:

```
   db2iauto -off InstName
where
```

`InstName` is the login name of the instance.

Cluster Configuration

In a *hot standby* configuration, the AIX processor node that is the takeover node *is not* running any other workload. In a *mutual takeover* configuration, the AIX processor node that is the takeover node *is* running other workloads.

Generally, in a partioned database environment, DB2 Universal Database runs in mutual takeover mode with partitions on each node. One exception is a scenario in which the catalog node is part of a hot standby configuration.

When planning a large DB2 installation on an RS/6000® SP™ using HACMP ES, you need to consider how to divide the nodes of the cluster within or between the RS/6000 SP frames. Having a node and its backup in different SP frames allows takeover in the event one frame goes down (that is, the frame power/switch board fails). However, such failures are expected to be exceedingly rare, because there are *N*+1 power supplies in each SP frame, and each SP switch has redundant paths, along with *N*+1 fans and power. In the case of a frame failure, manual intervention may be required to recover the remaining frames. This recovery procedure is documented in the SP Administration Guide. HACMP ES provides for recovery of SP node failures; recovery of frame failures is dependent on the proper layout of clusters within one or more SP frames.

Another planning consideration is how to manage big clusters. It is easier to manage a small cluster than a big one; however, it is also easier to manage one big cluster than many smaller ones. When planning, consider how your applications will be used in your cluster environment. If there is a single, large, homogeneous application running, for example, on 16 nodes, it is probably easier to manage the configuration as a single cluster rather than as eight two-node clusters. If the same 16 nodes contain many different applications with different networks, disks, and node relationships, it is probably better to group the nodes into smaller clusters. Keep in mind that nodes integrate into an HACMP cluster one at a time; it will be faster to start a configuration of multiple clusters rather than one large cluster. HACMP ES supports both single and multiple clusters, as long as a node and its backup are in the same cluster.

HACMP ES failover recovery allows pre-defined (also known as *cascading*) assignment of a resource group to a physical node. The failover recovery procedure also allows floating (or *rotating*) assignment of a resource group to a physical node. IP addresses, and external disk volume groups, or file systems, or NFS file systems, and application servers within each resource group specify either an application or an application component, which can be manipulated by HACMP ES between physical nodes by failover and reintegration. Failover and reintegration behavior is specified by the type of resource group created, and by the number of nodes placed in the resource group.

For example, consider a DB2 database partition (logical node). If its log and table space containers were placed on external disks, and other nodes were linked to those disks, it would be possible for those other nodes to access these disks and to restart the database partition (on a takeover node). It is this type of operation that is automated by HACMP. HACMP ES can also be used to recover NFS file systems used by DB2 instance main user directories.

Read the HACMP ES documentation thoroughly as part of your planning for recovery with DB2 UDB in a partitioned database environment. You should read the Concepts, Planning, Installation, and Administration guides, then build the recovery architecture for your environment. For each subsystem that you have identified for recovery, based on known points of failure, identify the HACMP clusters that you need, as well as the recovery nodes (either hot standby or mutual takeover).

It is strongly recommended that both disks and adapters be mirrored in your external disk configuration. For DB2 physical nodes that are configured for HACMP, care is required to ensure that nodes on the volume group can vary from the shared external disks. In a mutual takeover configuration, this arrangement requires some additional planning, so that the paired nodes can access each other's volume groups without conflicts. In a partitioned database environment, this means that all container names must be unique across all databases.

One way to achieve uniqueness is to include the partition number as part of the name. You can specify a node expression for container string syntax when creating either SMS or DMS containers. When you specify the expression, the node number can be part of the container name or, if you specify additional arguments, the results of those arguments can be part of the container name. Use the argument ″ $N″ ([blank]$N) to indicate the node expression. The argument must occur at the end of the container string, and can only be used in one of the following forms:

Table 1. Arguments for Creating Containers. The node number is assumed to be five.

Syntax	Example	Value
[blank]$N	″ $N″	5
[blank]$N+[number]	″ $N+1011″	1016
[blank]$N%[number]	″ $N%3″	2
[blank]$N+[number]%[number]	″ $N+12%13″	4
[blank]$N%[number]+[number]	″ $N%3+20″	22

Notes:

1. % is modulus.
2. In all cases, the operators are evaluated from left to right.

Following are some examples of how to create containers using this special argument:

- Creating containers for use on a two-node system.

  ```
  CREATE TABLESPACE TS1 MANAGED BY DATABASE USING
     (device '/dev/rcont $N' 20000)
  ```

  The following containers would be used:

  ```
  /dev/rcont0   - on Node 0
  /dev/rcont1   - on Node 1
  ```

- Creating containers for use on a four-node system.

  ```
  CREATE TABLESPACE TS2 MANAGED BY DATABASE USING
     (file '/DB2/containers/TS2/container $N+100' 10000)
  ```

  The following containers would be used:

  ```
  /DB2/containers/TS2/container100   - on Node 0
  /DB2/containers/TS2/container101   - on Node 1
  /DB2/containers/TS2/container102   - on Node 2
  /DB2/containers/TS2/container103   - on Node 3
  ```

- Creating containers for use on a two-node system.

  ```
  CREATE TABLESPACE TS3 MANAGED BY SYSTEM USING
     ('/TS3/cont $N%2, '/TS3/cont $N%2+2')
  ```

  The following containers would be used:

  ```
  /TS3/cont0   - on Node 0
  /TS3/cont2   - on Node 0
  /TS3/cont1   - on Node 1
  /TS3/cont3   - on Node 1
  ```

Configuring DB2 Database Partitions for HACMP ES

Once configured, each database partition in an instance is started by HACMP ES, one physical node at a time. Multiple clusters are recommended for starting parallel DB2 configurations that are larger than four nodes. Note that in a 64-node parallel DB2 configuration, it is faster to start 32 two-node HACMP clusters in parallel, than four 16-node clusters.

A script file, `rc.db2pe`, is packaged with DB2 UDB Enterprise Server Edition (and installed on each node in `/usr/bin`) to assist in configuring for HACMP ES failover or recovery in either hot standby or mutual takeover nodes. In addition, DB2 buffer pool sizes can be customized during failover in mutual takeover configurations from within `rc.db2pe`. (Buffer pool sizes can be configured to ensure proper resource allocation when two database partitions run on one physical node.)

HACMP ES Event Monitoring and User-defined Events

Initiating a failover operation if a process dies on a given node, is an example of a user-defined event. Examples that illustrate user-defined events, such as shutting down a database partition and forcing a transaction abort to free paging space, can be found in the `samples/hacmp/es` subdirectory.

A rules file, `/user/sbin/cluster/events/rules.hacmprd`, contains HACMP events. Each event description in this file has the following nine components:

- Event name, which must be unique.
- State, or qualifier for the event. The event name and state are the rule triggers. HACMP ES Cluster Manager initiates recovery only if it finds a rule with a trigger corresponding to the event name and state.
- Resource program path, a full-path specification of the *`xxx`*`.rp` file containing the recovery program.
- Recovery type. This is reserved for future use.
- Recovery level. This is reserved for future use.
- Resource variable name, which is used for Event Manager events.
- Instance vector, which is used for Event Manager events. This is a set of elements of the form ″name=value″. The values uniquely identify the copy of the resource in the system and, by extension, the copy of the resource variable.
- Predicate, which is used for Event Manager events. This is a relational expression between a resource variable and other elements. When this expression is true, the Event Management subsystem generates an event to notify the Cluster Manager and the appropriate application.
- Rearm predicate, which is used for Event Manager events. This is a predicate used to generate an event that alters the status of the primary

predicate. This predicate is typically the inverse of the primary predicate. It can also be used with the event predicate to establish an upper and a lower boundary for a condition of interest.

Each object requires one line in the event definition, even if the line is not used. If these lines are removed, HACMP ES Cluster Manager cannot parse the event definition properly, and this may cause the system to hang. Any line beginning with ″#″ is treated as a comment line.

Note: The rules file requires exactly nine lines for each event definition, not counting any comment lines. When adding a user-defined event at the bottom of the rules file, it is important to remove the unnecessary empty line at the end of the file, or the node will hang.

HACMP ES uses PSSP event detection to treat user-defined events. The PSSP Event Management subsystem provides comprehensive event detection by monitoring various hardware and software resources.

The process can be summarized as follows:

1. Either Group Services/ES (for predefined events) or Event Management (for user-defined events) notifies HACMP ES Cluster Manager of the event.
2. Cluster Manager reads the `rules.hacmprd` file, and determines the recovery program that is mapped to the event.
3. Cluster Manager runs the recovery program, which consists of a sequence of recovery commands.
4. The recovery program executes the recovery commands, which may be shell scripts or binary commands. (In HACMP for AIX, the recovery commands are the same as the HACMP event scripts.)
5. Cluster Manager receives the return status from the recovery commands. An unexpected status ″hangs″ the cluster until manual intervention (using `smit cm_rec_aids` or the `/usr/sbin/cluster/utilities/clruncmd` command) is carried out.

For detailed information on the implementation and design of highly available IBM® DB2 Universal Database™ environments on AIX see the following white papers which are available from the ″DB2 UDB and DB2 Connect™ Support″ web site (http://www.ibm.com/software/data/pubs/papers/).:

- ″IBM DB2 Universal Database Enterprise Edition for AIX and HACMP/ES″
- ″IBM DB2 Universal Database Enterprise - Extended Edition for AIX and HACMP/ES″

Related reference:

- “db2start - Start DB2” in the *Command Reference*

Chapter 7. High Availability on the Windows Operating System

Introduction

MSCS is a feature of Windows® NT Server, Windows 2000 Server, and Windows .NET Server operating systems. It is the software that supports the connection of two servers (up to four servers in DataCenter Server) into a cluster for high availability and easier management of data and applications. MSCS can also automatically detect and recover from server or application failures. It can be used to move server workloads to balance machine utilization and to provide for planned maintenance without downtime.

The following DB2® products have support for MSCS:

- DB2 Universal Database™ Workgroup Server Edition
- DB2 Universal Database Enterprise Server Edition (DB2 ESE)
- DB2 Universal Database Connect Enterprise Edition (DB2 CEE)

DB2 MSCS Components

A cluster is a configuration of two or more nodes, each of which is an independent computer system. The cluster appears to network clients as a single server.

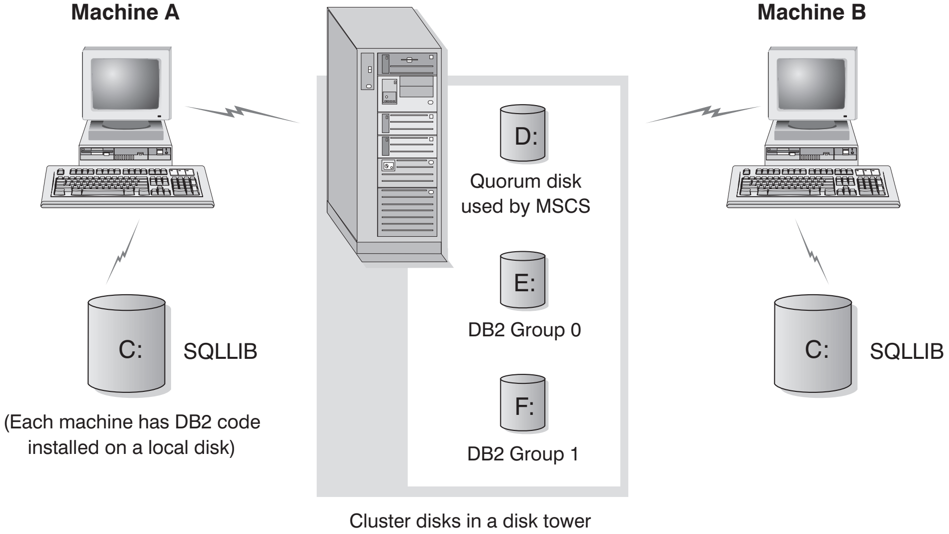


Figure 16. Example MSCS Configuration

The nodes in an MSCS cluster are connected using one or more shared storage buses and one or more physically independent networks. The network that connects only the servers but does not connect the clients to the cluster is referred to as a *private* network. The network that supports client connections is referred to as the *public* network. There are one or more local disks on each node. Each shared storage bus attaches to one or more disks. Each disk on the shared bus is owned by only one node of the cluster at a time. The DB2 software resides on the local disk. DB2 database files (tables, indexes, log files, etc.) reside on the shared disks. Because MSCS does not support the use of raw partitions in a cluster, it is not possible to configure DB2 to use raw devices in an MSCS environment.

The DB2 Resource

In an MSCS environment, a resource is an entity that is managed by the clustering software. For example, a disk, an IP address, or a generic service can be managed as a resource. DB2 integrates with MSCS by creating its own resource type called ôDB2.ö Each DB2 resource manages a DB2 instance, and when running in a partitioned database environment, each DB2 resource manages a database partition. The name of the DB2 resource is the instance name, although in the case of a partitioned database environment, the name of the DB2 resource consists of both the instance name and the partition (or node) number.

Pre-online and Post-online Script

You can run scripts both before and after a DB2 resource is brought online. These scripts are referred to as pre-online and post-online scripts respectively. Pre-online and post-online scripts are .BAT files that can run DB2 and system commands.

In a situation when multiple instances of DB2 may be running on the same machine, you can use the pre-online and post-online scripts to adjust the configuration so that both instances can be started successfully. In the event of a failover, you can use the post-online script to perform manual database recovery. Post-online script can also be used to start any applications or services that depend on DB2.

The DB2 Group

Related or dependent resources are organized into resource groups. All resources in a group move between cluster nodes as a unit. For example, in a typical DB2 single partition cluster environment, there will be a DB2 group that contains the following resources:

1. DB2 resource. The DB2 resource manages the DB2 instance (or node).
2. IP Address resource. The IP Address resource allows client applications to connect to the DB2 server.
3. Network Name resource. The Network Name resource allows client applications to connect to the DB2 server by using a name rather than using an IP address. The Network Name resource has a dependency on the IP Address resource. The Network Name resource is optional. (Configuring a Network Name resource may affect the failover performance.)
4. One or more Physical Disk resources. Each Physical Disk resource manages a shared disk in the cluster.

Note: The DB2 resource is configured to depend on all other resources in the same group so the DB2 server can only be started after all other resources are online.

Failover Configurations

Two types of configuration are available:

- Hot standby
- Mutual takeover

In a partitioned database environment, the clusters do not all have to have the same type of configuration. You can have some clusters that are set up to use hot standby, and others that are set up for mutual takeover. For example, if your DB2 instance consists of five workstations, you can have two machines set up to use a mutual takeover configuration, two to use a hot standby configuration, and one machine not configured for failover support.

Hot Standby Configuration

In a hot standby configuration, one machine in the MSCS cluster provides dedicated failover support, and the other machine participates in the database system. If the machine participating in the database system fails, the database server on it will be started on the failover machine. If, in a partitioned database system, you are running multiple logical nodes on a machine and it fails, the logical nodes will be started on the failover machine. Figure 17 shows an example of a hot standby configuration.

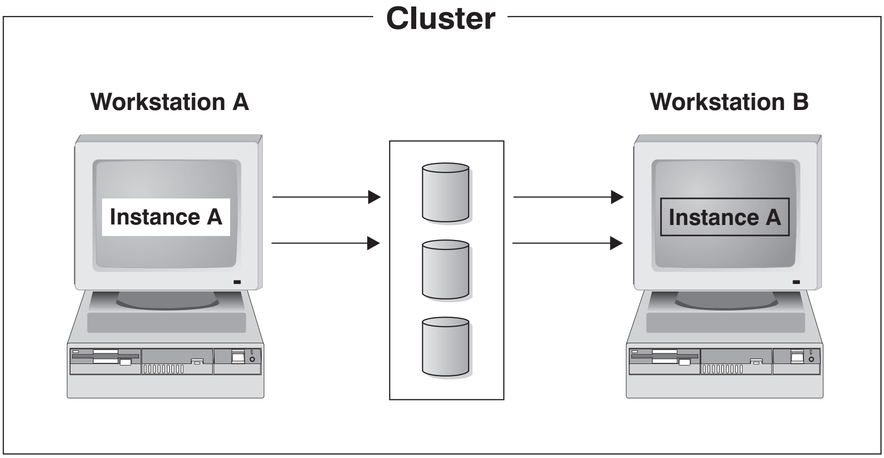


Figure 17. Hot Standby Configuration

Mutual Takeover Configuration

In a mutual takeover configuration, both workstations participate in the database system (that is, each machine has at least one database server running on it). If one of the workstations in the MSCS cluster fails, the database server on the failing machine will be started to run on the other machine. In a mutual takeover configuration, a database server on one machine can fail independently of the database server on another machine. Any database server can be active on any machine at any given point in time. Figure 18 on page 187 shows an example of a mutual takeover configuration.

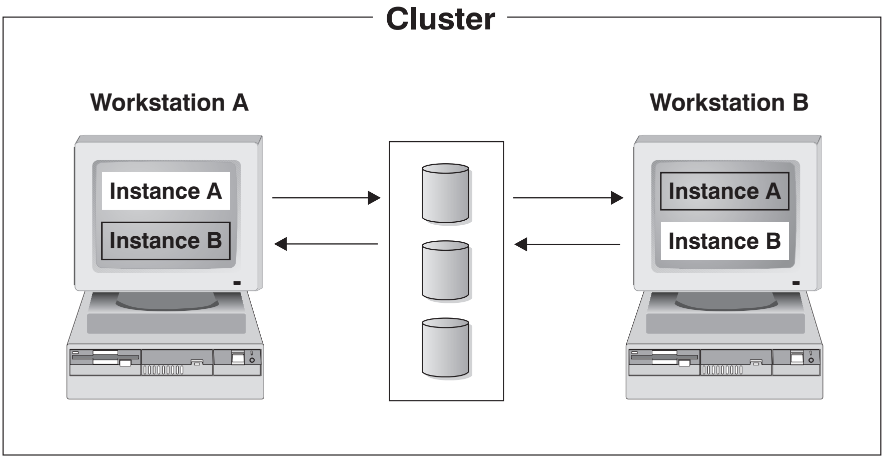


Figure 18. Mutual Takeover Configuration

For detailed information on the implementation and design of highly available IBM® DB2 Universal Database environments on the Windows Operating System see the following white papers which are available from the ″DB2 UDB and DB2 Connect™ Support″ web site (http://www.ibm.com/software/data/pubs/papers/).:

- ″Implementing IBM DB2 Universal Database Enterprise - Extended Edition with Microsoft® Cluster Server″
- ″Implementing IBM DB2 Universal Database Enterprise Edition with Microsoft Cluster Server″
- ″DB2 Universal Database for Windows: High Availability Support Using Microsoft Cluster Server - Overview″

Chapter 8. High Availability in the Solaris Operating Environment

High Availability in the Solaris Operating Environment

High availability in the Solaris Operating Environment can be achieved through DB2® working with Sun Cluster 3.0 (SC3.0), or Veritas Cluster Server (VCS). For information about Sun Cluster 3.0, see the white paper entitled “DB2 and High Availability on Sun Cluster 3.0”, which is available from the “DB2 UDB and DB2 Connect Online Support” web site (http://www.ibm.com/software/data/pubs/papers/). For information about VERITAS Cluster Server, see the white paper entitled “DB2 and High Availability on VERITAS Cluster Server”, which is also available from the “DB2 UDB and DB2 Connect™ Online Support” web site.

Note: When using Sun Cluster 3.0 or Veritas Cluster Server, ensure that DB2 instances are not started at boot time by using the **db2iauto** utility as follows:

```
   db2iauto -off InstName
where
```

`InstName` is the login name of the instance.

High Availability

The computer systems that host data services contain many distinct components, and each component has a ″mean time before failure″ (MTBF) associated with it. The MTBF is the average time that a component will remain usable. The MTBF for a quality hard drive is in the order of one million hours (approximately 114 years). While this seems like a long time, one out of 200 disks is likely to fail within a 6-month period.

Although there are a number of methods to increase availability for a data service, the most common is an HA cluster. A cluster, when used for high availability, consists of two or more machines, a set of private network interfaces, one or more public network interfaces, and some shared disks. This special configuration allows a data service to be moved from one machine to another. By moving the data service to another machine in the cluster, it should be able to continue providing access to its data. Moving a data service from one machine to another is called a *failover*, as illustrated in Figure 19 on page 190.

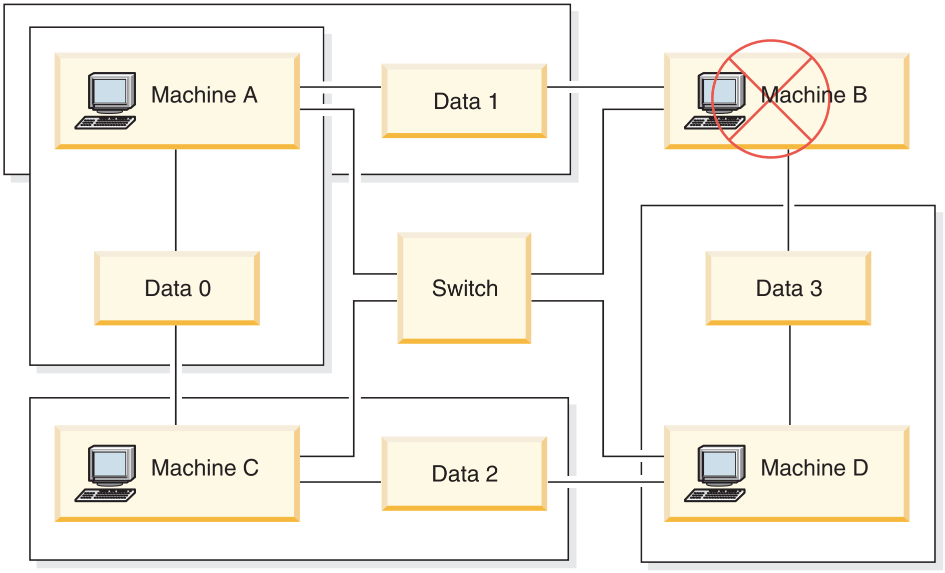


Figure 19. Failover. When Machine B fails its data service is moved to another machine in the cluster so that the data can still be accessed.

The private network interfaces are used to send *heartbeat* messages, as well as control messages, among the machines in the cluster. The public network interfaces are used to communicate directly with clients of the HA cluster. The disks in an HA cluster are connected to two or more machines in the cluster, so that if one machine fails, another machine has access to them.

A data service running on an HA cluster has one or more logical public network interfaces and a set of disks associated with it. The clients of an HA data service connect via TCP/IP to the logical network interfaces of the data service only. If a failover occurs, the data service, along with its logical network interfaces and set of disks, are moved to another machine.

One of the benefits of an HA cluster is that a data service can recover without the aid of support staff, and it can do so at any time. Another benefit is redundancy. All of the parts in the cluster should be redundant, including the machines themselves. The cluster should be able to survive any single point of failure.

Even though highly available data services can be very different in nature, they have some common requirements. Clients of a highly available data service expect the network address and host name of the data service to remain the same, and expect to be able to make requests in the same way, regardless of which machine the data service is on.

Consider a web browser that is accessing a highly available web server. The request is issued with a URL (Uniform Resource Locator), which contains both a host name, and the path to a file on the web server. The browser expects both the host name and the path to remain the same after a failover ofáthe web server. If the browser is downloading a file from the web server, and the server is failed over, the browser will need to reissue the request.

Availability of a data service is measured by the amount of time the data service is available to its users. The most common unit of measurement for availability is the percentage of ″up time″; this is often referred to as the number of ″nines″:

```
99.99% => service is down for (at most) 52.6 minutes / yr
99.999% => service is down for (at most) 5.26 minutes / yr
99.9999% => service is down for (at most) 31.5 seconds / yr
```

When designing and testing an HA cluster:

1. Ensure that the administrator of the cluster is familiar with the system and what should happen when a failover occurs.
2. Ensure that each part of the cluster is truly redundant and can be replaced quickly if it fails.
3. Force a test system to fail in a controlled environment, and make sure that it fails over correctly each time.
4. Keep track of the reasons for each failover. Although this should not happen often, it is important to address any issues that make the cluster unstable. For example, if one piece of the cluster caused a failover five times in one month, find out why and fix it.
5. Ensure that the support staff for the cluster is notified when a failover occurs.
6. Do not overload the cluster. Ensure that the remaining systems can still handle the workload at an acceptable level after a failover.
7. Check failure-prone components (such as disks) often, so that they can be replaced before problems occur.

Fault Tolerance

Another way to increase the availability of a data service is fault tolerance. A *fault tolerant* machine has all of its redundancy built in, and should be able to withstand a single failure of any part, including CPU and memory. Fault tolerant machines are most often used in niche markets, and are usually expensive to implement. An HA cluster with machines in different geographical locations has the added advantage of being able to recover from a disaster affecting only a subset of those locations.

An HA cluster is the most common solution to increase availability because it is scalable, easy to use, and relatively inexpensive to implement.

Related concepts:

- "High Availability on Sun Cluster 3.0" on page 192
- "High Availability with VERITAS Cluster Server" on page 195

High Availability on Sun Cluster 3.0

This section provides an overview of how DB2® works with Sun Cluster 3.0 to achieve high availability, and includes a description of the high availability agent, which acts as a mediator between the two software products (see Figure 20).

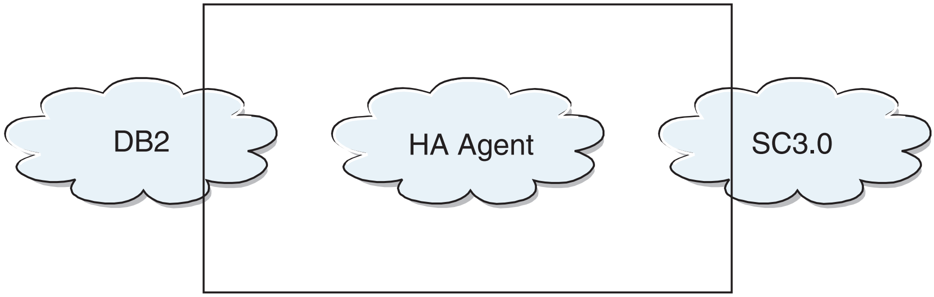


Figure 20. DB2, Sun Cluster 3.0, and High Availability. The relationship between DB2, Sun Cluster 3.0 and the high availability agent.

Failover

Sun Cluster 3.0 provides high availability by enabling application failover. Each node is periodically monitored and the cluster software automatically relocates a cluster-aware application from a failed primary node to a designated secondary node. When a failover occurs, clients may experience a brief interruption in service and may have to reconnect to the server. However, they will not be aware of the physical server from which they are accessing the application and the data. By allowing other nodes in a cluster to automatically host workloads when the primary node fails, Sun Cluster 3.0 can significantly reduce downtime and increase productivity.

Multihost Disks

Sun Cluster 3.0 requires multihost disk storage. This means that disks can be connected to more than one node at a time. In the Sun Cluster 3.0 environment, multihost storage allows disk devices to become highly available. Disk devices that reside on multihost storage can tolerate single node failures since there is still a physical path to the data through the alternate server node. Multihost disks can be accessed globally through a primary node. If client requests are accessing the data through one node and that node fails, the requests are switched over to another node that has a direct connection to the same disks. A volume manager provides for mirrored or RAID 5 configurations for data redundancy of the multihost disks. Currently, Sun Cluster 3.0 supports Solstice DiskSuite and VERITAS Volume Manager as volume managers. Combining multihost disks with disk mirroring and striping protects against both node failure and individual disk failure.

Global Devices

Global devices are used to provide cluster-wide, highly available access to any device in a cluster, from any node, regardless of the deviceÆs physical location. All disks are included in the global namespace with an assigned device ID (DID) and are configured as global devices. Therefore, the disks themselves are visible from all cluster nodes.

File systems/Global File Systems

A cluster or global file system is a proxy between the kernel (on one node) and the underlying file system volume manager (on a node that has a physical connection to one or more disks). Cluster file systems are dependent on global devices with physical connections to one or more nodes. They are independent of the underlying file system and volume manager. Currently, cluster file systems can be built on UFS using either Solstice DiskSuite or VERITAS Volume Manager. The data only becomes available to all nodes if the file systems on the disks are mounted globally as a cluster file system.

Device Group

All multihost disks must be controlled by the Sun Cluster framework. Disk groups, managed by either Solstice DiskSuite or VERITAS Volume Manager, are first created on the multihost disk. Then, they are registered as Sun Cluster disk device groups. A disk device group is a type of global device. Multihost device groups are highly available. Disks are accessible through an alternate path if the node currently mastering the device group fails. The failure of the node mastering the device group does not affect access to the device group except for the time required to perform the recovery and consistency checks. During this time, all requests are blocked (transparently to the application) until the system makes the device group available.

Resource Group Manager (RGM)

The RGM, provides the mechanism for high availability and runs as a daemon on each cluster node. It automatically starts and stops resources on selected nodes according to pre-configured policies. The RGM allows a resource to be highly available in the event of a node failure or to reboot by stopping the resource on the affected node and starting it on another. The RGM also automatically starts and stops resource-specific monitors that can detect resource failures and relocate failing resources onto another node.

Data Services

The term data service is used to describe a third-party application that has been configured to run on a cluster rather than on a single server. A data service includes the application software and Sun Cluster 3.0 software that starts, stops and monitors the application. Sun Cluster 3.0 supplies data service methods that are used to control and monitor the application within the cluster. These methods run under the control of the Resource Group Manager (RGM), which uses them to start, stop, and monitor the application on the cluster nodes. These methods, along with the cluster framework software and multihost disks, enable applications to become highly available data services. As highly available data services, they can prevent significant application interruptions after any single failure within the cluster, regardless of whether the failure is on a node, on an interface component or in the application itself. The RGM also manages resources in the cluster, including network resources (logical host names and shared addresses)and application instances.

Resource Type, Resource and Resource Group

A resource type is made up of the following:

1. A software application to be run on the cluster.
2. Control programs used as callback methods by the RGM to manage the application as a cluster resource.
3. A set of properties that form part of the static configuration of a cluster.

The RGM uses resource type properties to manage resources of a particular type.

A resource inherits the properties and values of its resource type. It is an instance of the underlying application running on the cluster. Each instance requires a unique name within the cluster. Each resource must be configured in a resource group. The RGM brings all resources in a group online and offline together on the same node. When the RGM brings a resource group online or offline, it invokes callback methods on the individual resources in the group.

The nodes on which a resource group is currently online are called its primary nodes, or its primaries.A resource group is mastered by each of its primaries. Each resource group has an associated Nodelist property, set by the cluster administrator, to identify all potential primaries or masters of the resource group.

For detailed information on the implementation and design of highly available IBM® DB2 Universal Database environments on the Sun Cluster 3.0 platform see the white paper entitled ″DB2 and High Availability on Sun Cluster 3.0″ which is available from the ″DB2 UDB and DB2 Connect™ Support″ web site (http://www.ibm.com/software/data/pubs/papers/).

Related concepts:

- "High Availability in the Solaris Operating Environment" on page 189
- "High Availability with VERITAS Cluster Server" on page 195

High Availability with VERITAS Cluster Server

VERITAS Cluster Server can be used to eliminate both planned and unplanned downtime. It can facilitate server consolidation and effectively manage a wide range of applications in heterogeneous environments. VERITAS Cluster Server supports up to 32 node clusters in both storage area network (SAN) and traditional client/server environments, VERITAS Cluster Server can protect everything from a single critical database instance, to very large multi-application clusters in networked storage environments. This section provides a brief summary of the features of VERITAS Cluster Server.

Hardware Requirements

Following is a list of hardware currently supported by VERITAS Cluster Server:

- For server nodes:
  - Any SPARC/Solaris server from Sun Microsystems running Solaris 2.6 or later with a minimum of 128MB RAM.
- For disk storage:
  - EMC Symmetrix, IBM® Enterprise Storage Server, HDS 7700 and 9xxx, Sun T3, Sun A5000, Sun A1000, Sun D1000 and any other disk storage supported by VCS 2.0 or later; your VERITAS representative can confirm which disk subsystems are supported or you can refer to VCS documentation.
  - Typical environments will require mirrored private disks (in each cluster node) for the DB2® UDB binaries and shared disks between nodes for the DB2 UDB data.

- For network interconnects:
  - For the public network connections, any network connection supporting IP-based addressing.
  - For the heartbeat connections (internal to the cluster), redundant heartbeat connections are required; this requirement can be met through the use of two additional Ethernet controllers per server or one additional Ethernet controller per server and the use of one shared GABdisk per cluster

Software Requirements

The following VERITAS software components are qualified configurations:

- VERITAS Volume Manager 3.2 or later, VERITAS File System 3.4 or later, VERITAS Cluster Server 2.0 or later.
- DB Edition for DB2 for Solaris 1.0 or later.

While VERITAS Cluster Server does not require a volume manager, the use of VERITAS Volume Manager is strongly recommended for ease of installation, configuration and management.

Failover

VERITAS Cluster Server is an availability clustering solution that manages the availability of application services, such as DB2 UDB, by enabling application failover. The state of each individual cluster node and its associated software services are regularly monitored. When a failure occurs that disrupts the application service (in this case, the DB2 UDB service), VERITAS Cluster Server and/or the VCS HA-DB2 Agent detect the failure and automatically take steps to restore the service. This can include restarting DB2 UDB on the same node or moving DB2 UDB to another node in the cluster and restarting it on that node. If an application needs to be migrated to a new node, VERITAS Cluster Server moves everything associated with the application (i.e., network IP addresses, ownership of underlying storage) to the new node so that users will not be aware that the service is actually running on another node. They will still access the service using the same IP addresses, but those addresses will now point to a different cluster node.

When a failover occurs with VERITAS Cluster Server, users may or may not see a disruption in service. This will be based on the type of connection (stateful or stateless) that the client has with the application service. In application environments with stateful connections (like DB2 UDB), users may see a brief interruption in service and may need to reconnect after the failover has completed. In application environments with stateless connections (like NFS), users may see a brief delay in service but generally will not see a disruption and will not need to log back on.

By supporting an application as a service that can be automatically migrated between cluster nodes, VERITAS Cluster Server can not only reduce unplanned downtime, but can also shorten the duration of outages associated with planned downtime (i.e., for maintenance and upgrades). Failovers can also be initiated manually. If a hardware or operating system upgrade must be performed on a particular node, DB2 UDB can be migrated to another node in the cluster, the upgrade can be performed, and then DB2 UDB can be migrated back to the original node.

Applications recommended for use in these types of clustering environments should be crash tolerant. A crash tolerant application can recover from an unexpected crash while still maintaining the integrity of committed data. Crash tolerant applications are sometimes referred to as *cluster friendly* applications. DB2 UDB is a crash tolerant application.

Shared Storage

When used with the VCS HA-DB2 Agent, Veritas Cluster Server requires shared storage. Shared storage is storage that has a physical connection to multiple nodes in the cluster. Disk devices resident on shared storage can tolerate node failures since a physical path to the disk devices still exists through one or more alternate cluster nodes.

Through the control of VERITAS Cluster Server, cluster nodes can access shared storage through a logical construct called "disk groups". Disk groups represent a collection of logically defined storage devices whose ownership can be atomically migrated between nodes in a cluster. A disk group can only be imported to a single node at any given time. For example, if Disk Group A is imported to Node 1 and Node 1 fails, Disk Group A can be exported from the failed node and imported to a new node in the cluster. VERITAS Cluster Server can simultaneously control multiple disk groups within a single cluster.

In addition to allowing disk group definition, a volume manager can provide for redundant data configurations, using mirroring or RAID 5, on shared storage. VERITAS Cluster Server supports VERITAS Volume Manager and Solstice DiskSuite as logical volume managers. Combining shared storage with disk mirroring and striping can protect against both node failure and individual disk or controller failure.

VERITAS Cluster Server Global Atomic Broadcast(GAB) and Low Latency Transport (LLT)

An internode communication mechanism is required in cluster configurations so that nodes can exchange information concerning hardware and software status, keep track of cluster membership, and keep this information synchronized across all cluster nodes. The Global Atomic Broadcast (GAB) facility, running across a low latency transport (LLT), provides the high speed, low latency mechanism used by VERITAS Cluster Server to do this. GAB is loaded as a kernel module on each cluster node and provides an atomic broadcast mechanism that ensures that all nodes get status update information at the same time.

By leveraging kernel-to-kernel communication capabilities, LLT provides high speed, low latency transport for all information that needs to be exchanged and synchronized between cluster nodes. GAB runs on top of LLT. VERITAS Cluster Server does not use IP as a heartbeat mechanism, but offers two other more reliable options. GAB with LLT, can be configured to act as a heartbeat mechanism, or a GABdisk can be configured as a disk-based heartbeat. The heartbeat must run over redundant connections. These connections can either be two private Ethernet connections between cluster nodes, or one private Ethernet connection and one GABdisk connection. The use of two GABdisks is not a supported configuration since the exchange of cluster status between nodes requires a private Ethernet connection.

For more information about GAB or LLT, or how to configure them in VERITAS Cluster Server configurations, please consult the VERITAS Cluster Server 2.0 User’s Guide for Solaris.

Bundled and Enterprise Agents

An agent is a program that is designed to manage the availability of a particular resource or application. When an agent is started, it obtains the necessary configuration information from VCS and then periodically monitors the resource or application and updates VCS with the status. In general, agents are used to bring resources online, take resources offline, or monitor resources and provide four types of services: start, stop, monitor and clean. Start and stop are used to bring resources online or offline, monitor is used to test a particular resource or application for its status, and clean is used in the recovery process.

A variety of bundled agents are included as part of VERITAS Cluster Server and are installed when VERITAS Cluster Server is installed. The bundled agents are VCS processes that manage predefined resource types commonly found in cluster configurations (i.e., IP, mount, process and share), and they help to simplify cluster installation and configuration considerably. There are over 20 bundled agents with VERITAS Cluster Server.

Enterprise agents tend to focus on specific applications such as DB2 UDB. The VCS HA-DB2 Agent can be considered an Enterprise Agent, and it interfaces with VCS through the VCS Agent framework.

VCS Resources, Resource Types and Resource Groups

A resource type is an object definition used to define resources within a VCS cluster that will be monitored. A resource type includes the resource type name and a set of properties associated with that resource that are salient from a high availability point of view. A resource inherits the properties and values of its resource type, and resource names must be unique on a cluster-wide basis.

There are two types of resources: persistent and standard (non-persistent). Persistent resources are resources such as network interface controllers (NICs) that are monitored but are not brought online or taken offline by VCS. Standard resources are those whose online and offline status is controlled by VCS.

The lowest level object that is monitored is a resource, and there are various resource types (i.e., share, mount). Each resource must be configured into a resource group, and VCS will bring all resources in a particular resource group online and offline together. To bring a resource group online or offline, VCS will invoke the start or stop methods for each of the resources in the group. There are two types of resource groups: failover and parallel. A highly available DB2 UDB configuration, regardless of whether it is partitioned or not, will use failover resource groups.

A ″primary″ or ″master″ node is a node that can potentially host a resource. A resource group attribute called `systemlist` is used to specify which nodes within a cluster can be primaries for a particular resource group. In a two node cluster, usually both nodes are included in the `systemlist`, but in larger, multi-node clusters that may be hosting several highly available applications there may be a requirement to ensure that certain application services (defined by their resources at the lowest level) can never fail over to certain nodes.

Dependencies can be defined between resource groups, and VERITAS Cluster Server depends on this resource group dependency hierarchy in assessing the impact of various resource failures and in managing recovery. For example, if the resource group ClientApp1 can not be brought online unless the resource group DB2 has already been successfully started, resource group ClientApp1 is considered dependent on resource group DB2.

For detailed information on the implementation and design of highly available IBM DB2 Universal Database environments with the VERITAS Cluster Server see the technote entitled ″DB2 UDB and High Availability with VERITAS Cluster Server″ which you can view by going to the following web site: http://www.ibm.com/support, and searching for the keyword ″1045033″.

Related concepts:

- "High Availability in the Solaris Operating Environment" on page 189

- “High Availability on Sun Cluster 3.0” on page 192

Part 3. Appendixes

Appendix A. How to Read the Syntax Diagrams

A syntax diagram shows how a command should be specified so that the operating system can correctly interpret what is typed.

Read a syntax diagram from left to right, and from top to bottom, following the horizontal line (the main path). If the line ends with an arrowhead, the command syntax is continued, and the next line starts with an arrowhead. A vertical bar marks the end of the command syntax.

When typing information from a syntax diagram, be sure to include punctuation, such as quotation marks and equal signs.

Parameters are classified as keywords or variables:

- Keywords represent constants, and are shown in uppercase letters; at the command prompt, however, keywords can be entered in upper, lower, or mixed case. A command name is an example of a keyword.
- Variables represent names or values that are supplied by the user, and are shown in lowercase letters; at the command prompt, however, variables can be entered in upper, lower, or mixed case, unless case restrictions are explicitly stated. A file name is an example of a variable.

A parameter can be a combination of a keyword and a variable.

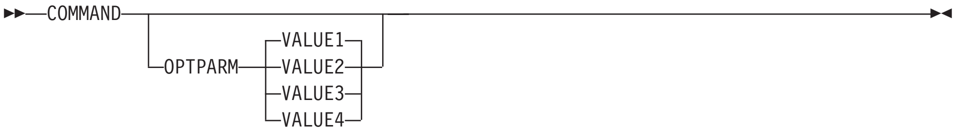
Required parameters are displayed on the main path:

```
►►──COMMAND──required parameter──────────────────────────────────────────►◄
```

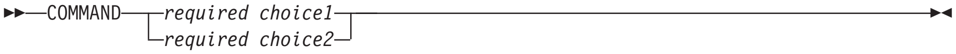
Optional parameters are displayed below the main path:

```
►►──COMMAND──┬────────────────────┬──────────────────────────────────────►◄
             └─optional parameter─┘
```

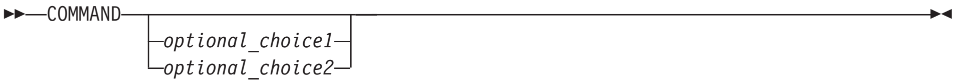
A parameter's default value is displayed above the path:



A stack of parameters, with the first parameter displayed on the main path, indicates that one of the parameters must be selected:

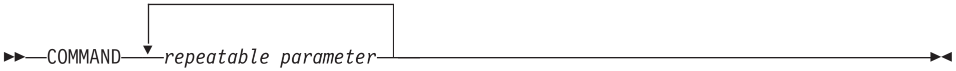


A stack of parameters, with the first parameter displayed below the main path, indicates that one of the parameters can be selected:

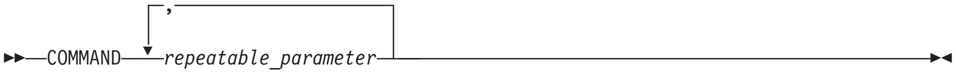


An arrow returning to the left, above the path, indicates that items can be repeated in accordance with the following conventions:

- If the arrow is uninterrupted, the item can be repeated in a list with the items separated by blank spaces:

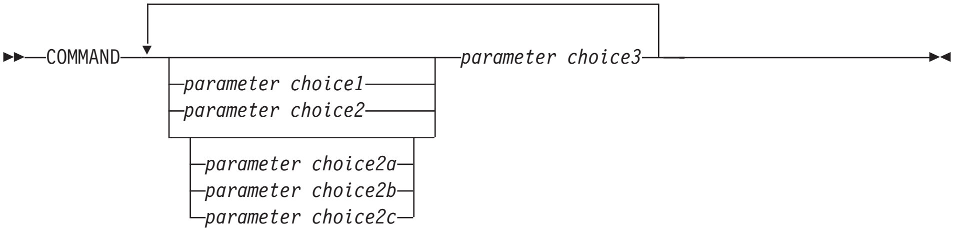


- If the arrow contains a comma, the item can be repeated in a list with the items separated by commas:

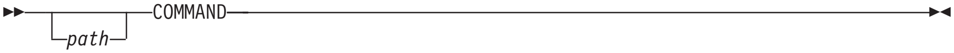


Items from parameter stacks can be repeated in accordance with the stack conventions for required and optional parameters discussed previously.

Some syntax diagrams contain parameter stacks within other parameter stacks. Items from stacks can only be repeated in accordance with the conventions discussed previously. That is, if an inner stack does not have a repeat arrow above it, but an outer stack does, only one parameter from the inner stack can be chosen and combined with any parameter from the outer stack, and that combination can be repeated. For example, the following diagram shows that one could combine parameter *choice2a* with parameter *choice2*, and then repeat that combination again (*choice2* plus *choice2a*):

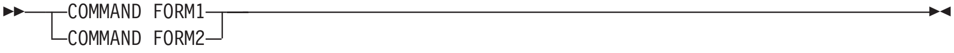


Some commands are preceded by an optional path parameter:



If this parameter is not supplied, the system searches the current directory for the command. If it cannot find the command, the system continues searching for the command in all the directories on the paths listed in the `.profile`.

Some commands have syntactical variants that are functionally equivalent:



Appendix B. Warning, Error and Completion Messages

Messages generated by the various utilities are included among the SQL messages. These messages are generated by the database manager when a warning or error condition has been detected. Each message has a message identifier that consists of a prefix (SQL) and a four- or five-digit message number. There are three message types: notification, warning, and critical. Message identifiers ending with an `N` are error messages. Those ending with a `W` indicate warning or informational messages. Message identifiers ending with a `C` indicate critical system errors.

The message number is also referred to as the *SQLCODE*. The SQLCODE is passed to the application as a positive or negative number, depending on its message type (N, W, or C). N and C yield negative values, whereas W yields a positive value. DB2 returns the SQLCODE to the application, and the application can get the message associated with the SQLCODE. DB2 also returns an *SQLSTATE* value for conditions that could be the result of an SQL statement. Some SQLCODE values have associated SQLSTATE values.

You can use the information contained in this book to identify an error or problem, and to resolve the problem by using the appropriate recovery action. This information can also be used to understand where messages are generated and logged.

SQL messages, and the message text associated with SQLSTATE values, are also accessible from the operating system command line. To access help for these error messages, enter the following at the operating system command prompt:

```
db2 ? SQLnnnnn
```

where *nnnnn* represents the message number. On UNIX based systems, the use of double quotation mark delimiters is recommended; this will avoid problems if there are single character file names in the directory:

```
db2 "? SQLnnnnn"
```

The message identifier accepted as a parameter for the **db2** command is not case sensitive, and the terminating letter is not required. Therefore, the following commands will produce the same result:

```
db2 ? SQL0000N
db2 ? sql0000
db2 ? SQL0000n
```

If the message text is too long for your screen, use the following command (on UNIX based operating systems and others that support the ″more″ pipe):

```
db2 ? SQLnnnnn | more
```

You can also redirect the output to a file which can then be browsed.

Help can also be invoked from interactive input mode. To access this mode, enter the following at the operating system command prompt:

```
db2
```

To get DB2 message help in this mode, type the following at the command prompt (`db2 =>`):

```
? SQLnnnnn
```

The message text associated with SQLSTATEs can be retrieved by issuing:

```
db2 ? nnnnn
 or
db2 ? nn
```

where *nnnnn* is a five-character SQLSTATE value (alphanumeric), and *nn* is a two-digit SQLSTATE class code (the first two digits of the SQLSTATE value).

Appendix C. Additional DB2 Commands

System Commands

db2adutl - Work with TSM Archived Images

Allows users to query, extract, verify, and delete backup images, logs, and load copy images saved using Tivoli Storage Manager (formerly ADSM).

On UNIX-based operating systems, this utility is located in the sqllib/adsm directory. On Windows it is located in sqllib\bin.

Authorization:

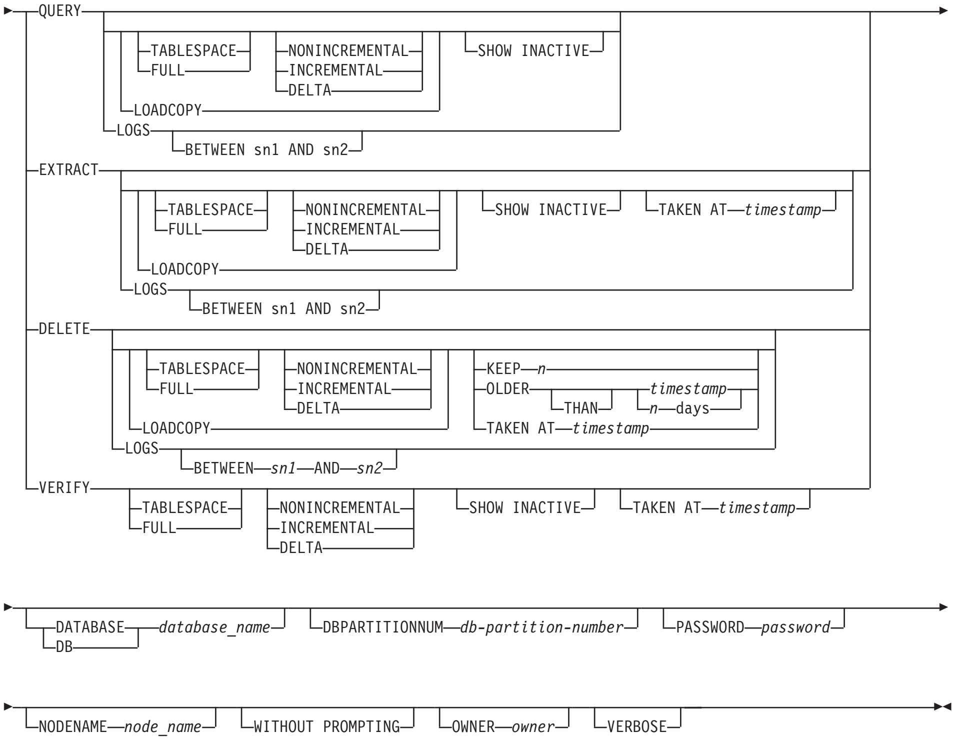
None

Required connection:

None

Command syntax:

```
►►—db2adutl———————————————————————————————————————————————►
```



Command parameters:

QUERY
: Queries the TSM server for DB2 objects.

EXTRACT
: Copies DB2 objects from the TSM server to the current directory on the local machine.

DELETE
: Either deactivates backup objects or deletes log archives on the TSM server.

VERIFY
: Performs consistency checking on the backup copy that is on the server.

  Note: This parameter causes the entire backup image to be transferred over the network.

TABLESPACE
: Includes only table space backup images.

FULL Includes only full database backup images.

NONINCREMENTAL
Include only non-incremental backup images.

INCREMENTAL
Include only incremental backup images.

DELTA
Include only incremental delta backup images.

LOADCOPY
Includes only load copy images.

LOGS Includes only log archive images

BETWEEN *sn1* **AND** *sn2*
Specifies that the logs between log sequence number 1 and log sequence number 2 are to be used.

SHOW INACTIVE
Includes backup objects that have been deactivated.

TAKEN AT *timestamp*
Specifies a backup image by its time stamp.

KEEP *n*
Deactivates all objects of the specified type except for the most recent *n* by time stamp.

OLDER THAN *timestamp* **or** *n* **days**
Specifies that objects with a time stamp earlier than *timestamp* or *n* days will be deactivated.

DATABASE *database_name*
Considers only those objects associated with the specified database name.

DBPARTITIONNUM *db-partition-number*
Considers only those objects created by the specified database partition number.

PASSWORD *password*
Specifies the TSM client password for this node, if required. If a database is specified and the password is not provided, the value specified for the *tsm_password* database configuration parameter is passed to TSM; otherwise, no password is used.

NODENAME *node_name*
Considers only those images associated with a specific TSM node name.

WITHOUT PROMPTING
The user is not prompted for verification before objects are deleted.

OWNER *owner*
Considers only those objects created by the specified owner.

VERBOSE
Displays additional file information

Examples:

The following is sample output from: `db2 backup database rawsampl use tsm`

```
Backup successful. The timestamp for this backup is : 19970929130942

db2adutl query

Query for database RAWSAMPL

Retrieving full database backup information.
   full database backup image: 1, Time: 19970929130942,
                              Oldest log: S0000053.LOG, Sessions used: 1
   full database backup image: 2, Time: 19970929142241,
                              Oldest log: S0000054.LOG, Sessions used: 1

Retrieving table space backup information.
   table space backup image: 1, Time: 19970929094003,
                            Oldest log: S0000051.LOG, Sessions used: 1
   table space backup image: 2, Time: 19970929093043,
                            Oldest log: S0000050.LOG, Sessions used: 1
   table space backup image: 3, Time: 19970929105905,
                            Oldest log: S0000052.LOG, Sessions used: 1

Retrieving log archive information.
   Log file: S0000050.LOG
   Log file: S0000051.LOG
   Log file: S0000052.LOG
   Log file: S0000053.LOG
   Log file: S0000054.LOG
   Log file: S0000055.LOG
```

The following is sample output from: `db2adutl delete full taken at 19950929130942 db rawsampl`

```
Query for database RAWSAMPL

Retrieving full database backup information.  Please wait.

  full database backup image: RAWSAMPL.0.db26000.0.19970929130942.001

  Do you want to deactivate this backup image (Y/N)? y

    Are you sure (Y/N)? y

db2adutl query

Query for database RAWSAMPL
```

```
Retrieving full database backup information.
   full database backup image: 2, Time: 19950929142241,
                            Oldest log: S0000054.LOG, Sessions used: 1

Retrieving table space backup information.
   table space backup image: 1, Time: 19950929094003,
                          Oldest log: S0000051.LOG, Sessions used: 1
   table space backup image: 2, Time: 19950929093043,
                          Oldest log: S0000050.LOG, Sessions used: 1
   table space backup image: 3, Time: 19950929105905,
                          Oldest log: S0000052.LOG, Sessions used: 1

Retrieving log archive information.
   Log file: S0000050.LOG
   Log file: S0000051.LOG
   Log file: S0000052.LOG
   Log file: S0000053.LOG
   Log file: S0000054.LOG
   Log file: S0000055.LOG
```

Usage Notes:

One parameter from each group below can be used to restrict what backup images types are included in the operation:

Granularity:

- FULL - include only database backup images.
- TABLESPACE - include only table space backup images.

Cumulativiness:

- NONINCREMENTAL - include only non-incremental backup images.
- INCREMENTAL - include only incremental backup images.
- DELTA - include only incremental delta backup images.

Compatibilities:

For compatibility with versions earlier than Version 8:

- The keyword NODE can be substituted for DBPARTITIONNUM.

db2ckbkp - Check Backup

This utility can be used to test the integrity of a backup image and to determine whether or not the image can be restored. It can also be used to display the meta-data stored in the backup header.

Authorization:

Anyone can access the utility, but users must have read permissions on image backups in order to execute this utility against them.

Required connection:

None

Command syntax:

```
               ┌─,───────┐ ┌─,───────┐
               ↓         │ ↓         │
►►──db2ckbkp────┬─────┬──┴──filename─┴──────────────────────────────►◄
                ├─-a──┤
                ├─-c──┤
                ├─-d──┤
                ├─-h──┤
                ├─-H──┤
                ├─-l──┤
                ├─-n──┤
                └─-o──┘
```

Command parameters:

-a Displays all available information.

-c Displays results of checkbits and checksums.

-d Displays information from the headers of DMS table space data pages.

-h Displays media header information including the name and path of the image expected by the restore utility.

-H Displays the same information as -h but only reads the 4K media header information from the beginning of the image. It does not validate the image.

Note: This option cannot be used in combination with any other options.

-l Displays Log File Header data.

-n Prompt for tape mount. Assume one tape per device.

-o Displays detailed information from the object headers.

filename
The name of the backup image file. One or more files can be checked at a time.

Notes:

1. If the complete backup consists of multiple objects, the validation will only succeed if **db2ckbkp** is used to validate all of the objects at the same time.
2. When checking multiple parts of an image, the first backup image object (.001) must be specified first.

Examples:

```
db2ckbkp SAMPLE.0.krodger.NODE0000.CATN0000.19990817150714.*
[1] Buffers processed:  ##
[2] Buffers processed:  ##
[3] Buffers processed:  ##
Image Verification Complete - successful.

db2ckbkp -h SAMPLE2.0.krodger.NODE0000.CATN0000.19990818122909.001


=====================
MEDIA HEADER REACHED:
=====================
        Server Database Name           -- SAMPLE2
        Server Database Alias          -- SAMPLE2
        Client Database Alias          -- SAMPLE2
        Timestamp                      -- 19990818122909
        Database Partition Number      -- 0
        Instance                       -- krodger
        Sequence Number                -- 1
        Release ID                     -- 900
        Database Seed                  -- 65E0B395
        DB Comment's Codepage (Volume) -- 0
        DB Comment (Volume)            --
        DB Comment's Codepage (System) -- 0
        DB Comment (System)            --
        Authentication Value           -- 255
        Backup Mode                    -- 0
        Backup Type                    -- 0
        Backup Gran.                   -- 0
        Status Flags                   -- 11
        System Cats inc                -- 1
        Catalog Database Partition No. -- 0
        DB Codeset                     -- ISO8859-1
        DB Territory                   --
        Backup Buffer Size             -- 4194304
        Number of Sessions             -- 1
        Platform                       -- 0

The proper image file name would be:
SAMPLE2.0.krodger.NODE0000.CATN0000.19990818122909.001

[1] Buffers processed:  ####
Image Verification Complete - successful.
```

Usage notes:

1. If a backup image was created using multiple sessions, **db2ckbkp** can examine all of the files at the same time. Users are responsible for ensuring that the session with sequence number `001` is the first file specified.
2. This utility can also verify backup images that are stored on tape (except images that were created with a variable block size). This is done by preparing the tape as for a restore operation, and then invoking the utility, specifying the tape device name. For example, on UNIX based systems:

   ```
   db2ckbkp -h /dev/rmt0
   ```

   and on Windows:

   ```
   db2ckbkp -d \\.\tape1
   ```

3. If the image is on a tape device, specify the tape device path. You will be prompted to ensure it is mounted, unless option ’-n’ is given. If there are multiple tapes, the first tape must be mounted on the first device path given. (That is the tape with sequence 001 in the header).

   The default when a tape device is detected is to prompt the user to mount the tape. The user has the choice on the prompt. Here is the prompt and options: (where the device I specified is on device path `/dev/rmt0`)

   ```
    Please mount the source media on device /dev/rmt0.
    Continue(c), terminate only this device(d), or abort this tool(t)?
    (c/d/t)
   ```

   The user will be prompted for each device specified, and when the device reaches the end of tape.

Related reference:

- “db2adutl - Work with TSM Archived Images” on page 209

db2ckrst - Check Incremental Restore Image Sequence

Queries the database history and generates a list of timestamps for the backup images that are required for an incremental restore. A simplified restore syntax for a manual incremental restore is also generated.

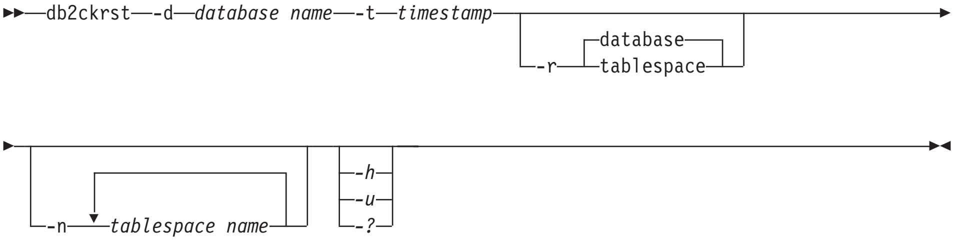
Authorization:

None

Required connection:

None

Command syntax:



Command parameters:

-d database name file-name
: Specifies the alias name for the database that will be restored.

-t timestamp
: Specifies the timestamp for a backup image that will be incrementally restored.

-r
: Specifies the type of restore that will be executed. The default is database.

  Note: If tablespace is chosen and no table space names are given, the utility looks into the history entry of the specified image and uses the table space names listed to do the restore.

-n tablespace name
: Specifies the name of one or more table spaces that will be restored.

  Note: If a database restore type is selected and a list of table space names is specified, the utility will continue as a tablespace restore using the table space names given.

-h/-u/-?
: Displays help information. When this option is specified, all other options are ignored, and only the help information is displayed.

Examples:

```
db2ckrst -d mr -t 20001015193455 -r database
db2ckrst -d mr -t 20001015193455 -r tablespace
db2ckrst -d mr -t 20001015193455 -r tablespace -n tbsp1 tbsp2

> db2 backup db mr

Backup successful. The timestamp for this backup image is : 20001016001426

> db2 backup db mr incremental

Backup successful. The timestamp for this backup image is : 20001016001445

> db2ckrst -d mr -t 20001016001445
```

```
Suggested restore order of images using timestamp 20001016001445 for
database mr.
===================================================================
  db2 restore db mr incremental taken at 20001016001445
  db2 restore db mr incremental taken at 20001016001426
  db2 restore db mr incremental taken at 20001016001445
===================================================================

> db2ckrst -d mr -t 20001016001445 -r tablespace -n userspace1
Suggested restore order of images using timestamp 20001016001445 for
database mr.
===================================================================
  db2 restore db mr tablespace ( USERSPACE1 ) incremental taken at
  20001016001445
  db2 restore db mr tablespace ( USERSPACE1 ) incremental taken at
  20001016001426
  db2 restore db mr tablespace ( USERSPACE1 ) incremental taken at
  20001016001445
===================================================================
```

Usage notes:

The database history must exist in order for this utility to be used. If the database history does not exist, specify the HISTORY FILE option in the RESTORE command before using this utility.

If the FORCE option of the PRUNE HISTORY command is used, you will be able to delete entries that are required for recovery from the most recent, full database backup image. The default operation of the PRUNE HISTORY command prevents required entries from being deleted. It is recommended that you do not use the FORCE option of the PRUNE HISTORY command.

This utility should not be used as a replacement for keeping records of your backups.

db2flsn - Find Log Sequence Number

Returns the name of the file that contains the log record identified by a specified log sequence number (LSN).

Authorization:

None

Command syntax:

```
►►──db2flsn──┬──────┬──input_LSN──────────────────────────────────────────►◄
             └─-q───┘
```

Command parameters:

-q
Specifies that only the log file name be printed. No error or warning messages will be printed, and status can only be determined through the return code. Valid error codes are:

- -100 Invalid input
- -101 Cannot open LFH file
- -102 Failed to read LFH file
- -103 Invalid LFH
- -104 Database is not recoverable
- -105 LSN too big
- -500 Logical error.

Other valid return codes are:

- 0 Successful execution
- 99 Warning: the result is based on the last known log file size.

input_LSN
A 12-byte string that represents the internal (6-byte) hexadecimal value with leading zeros.

Examples:

```
db2flsn 000000BF0030
   Given LSN is contained in log file S0000002.LOG

db2flsn -q 000000BF0030
   S0000002.LOG

db2flsn 000000BE0030
   Warning: the result is based on the last known log file size.
   The last known log file size is 23 4K pages starting from log extent 2.

   Given LSN is contained in log file S0000001.LOG

db2flsn -q 000000BE0030
   S0000001.LOG
```

Usage notes:

The log header control file SQLOGCTL.LFH must reside in the current directory. Since this file is located in the database directory, the tool can be run from the database directory, or the control file can be copied to the directory from which the tool will be run.

The tool uses the *logfilsiz* database configuration parameter. DB2 records the three most recent values for this parameter, and the first log file that is created with each *logfilsiz* value; this enables the tool to work correctly when *logfilsiz* changes. If the specified LSN predates the earliest recorded value of *logfilsiz*, the tool uses this value, and returns a warning. The tool can be used with database managers prior to UDB Version 5.2; in this case, the warning is returned even with a correct result (obtained if the value of *logfilsiz* remains unchanged).

This tool can only be used with recoverable databases. A database is recoverable if it is configured with *logretain* set to `RECOVERY` or *userexit* set to `ON`.

db2inidb - Initialize a Mirrored Database

Initializes a mirrored database in a split mirror environment. The mirrored database can be initialized as a clone of the primary database, placed in roll forward pending state, or used as a backup image to restore the primary database.

Authorization:

One of the following:

- *sysadm*
- *sysctrl*
- *sysmaint*

Required connection:

None

Command syntax:

```
►►──db2inidb──database_alias──AS──┬─SNAPSHOT─┬──┬────────────────────────────┬──────────►◄
                                  ├─STANDBY──┤  └─RELOCATE USING──configFile─┘
                                  └─MIRROR───┘
```

Command parameters:

database_alias
: Specifies the alias of the database to be initialized.

SNAPSHOT
: Specifies that the mirrored database will be initialized as a clone of the primary database.

STANDBY
: Specifies that the database will be placed in roll forward pending state.

Note: New logs from the primary database can be fetched and applied to the standby database. The standby database can then be used in place of the primary database if it goes down.

MIRROR
Specifies that the mirrored database is to be used as a backup image which can be used to restore the primary database.

RELOCATE USING configFile
Specifies that the database files are to be relocated based on the information listed in the configuration file.

Related reference:

- "db2relocatedb - Relocate Database" in the *Command Reference*

db2mscs - Set up Windows Failover Utility

Creates the infrastructure for DB2 failover support on Windows using Microsoft Cluster Server (MSCS). This utility can be used to enable failover in both single-partition and partitioned database environments.

Authorization:

The user must be logged on to a domain user account that belongs to the Administrators group of each machine in the MSCS cluster.

Command syntax:

```
►►──db2mscs──┬────────────────────┬──────────────────────────────────►◄
             ├─-f:──input_file────┤
             └─-u:──instance_name─┘
```

Command parameters:

-f:input_file
Specifies the `DB2MSCS.CFG` input file to be used by the MSCS utility. If this parameter is not specified, the DB2MSCS utility reads the `DB2MSCS.CFG` file that is in the current directory.

-u:instance_name
This option allows you to undo the db2mscs operation and revert the instance back to the non-MSCS instance specified by `instance_name`.

Usage notes:

The DB2MSCS utility is a standalone command line utility used to transform a non-MSCS instance into an MSCS instance. The utility will create all MSCS groups, resources, and resource dependencies. It will also copy all DB2 information stored in the Windows registry to the cluster portion of the registry as well as moving the instance directory to a shared cluster disk. The DB2MSCS utility takes as input a configuration file provided by the user specifying how the cluster should be set up. The DB2MSCS.CFG file is an ASCII text file that contains parameters that are read by the DB2MSCS utility. You specify each input parameter on a separate line using the following format: `PARAMETER_KEYWORD=parameter_value`. For example:

```
CLUSTER_NAME=FINANCE
GROUP_NAME=DB2 Group
IP_ADDRESS=9.21.22.89
```

Two example configuration files can be found in the `CFG` subdirectory under the DB2 install directory. The first, `DB2MSCS.EE`, is an example for single-partition database environments. The second, `DB2MSCS.EEE`, is an example for partitioned database environments.

The parameters for the DB2MSCS.CFG file are as follows:

DB2_INSTANCE
The name of the DB2 instance. This parameter has a global scope and should be specified only once in the DB2MSCS.CFG file.

DAS_INSTANCE
The name of the DB2 Admin Server instance. Specify this parameter to migrate the DB2 Admin Server to run in the MSCS environment. This parameter has a global scope and should be specified only once in the DB2MSCS.CFG file.

CLUSTER_NAME
The name of the MSCS cluster. All the resources specified following this line are created in this cluster until another CLUSTER_NAME parameter is specified.

DB2_LOGON_USERNAME
The username of the domain account for the DB2 service (i.e. domain\user). This parameter has a global scope and should be specified only once in the DB2MSCS.CFG file.

DB2_LOGON_PASSWORD
The password of the domain account for the DB2 service. This parameter has a global scope and should be specified only once in the DB2MSCS.CFG file.

GROUP_NAME
The name of the MSCS group. If this parameter is specified, a new MSCS group is created if it does not exist. If the group already exists, it is used as the target group. Any MSCS resource specified after this

parameter is created in this group or moved into this group until another GROUP_NAME parameter is specified. Specify this parameter once for each group.

DB2_NODE
The partition number of the database partition server (or database partition) to be included in the current MSCS group. If multiple logical database partitions exist on the same machine, each database partition requires a separate DB2_NODE parameter. Specify this parameter after the GROUP_NAME parameter so that the DB2 resources are created in the correct MSCS group. This parameter is required for a multi-partitioned database system.

IP_NAME
The name of the IP Address resource. The value for the IP_NAME is arbitrary, but it must be unique in the cluster. When this parameter is specified, an MSCS resource of type IP Address is created. This parameter is required for remote TCP/IP connections. This parameter is optional in a single partition environment. A recommended name is the hostname that corresponds to the IP address.

IP_ADDRESS
The TCP/IP address for the IP resource specified by the preceding IP_NAME parameter. This parameter is required if the IP_NAME parameter is specified. This is a new IP address that is not used by any machine in the network.

IP_SUBNET
The TCP/IP subnet mask for the IP resource specified by the preceding IP_NAME parameter. This parameter is required if the IP_NAME parameter is specified.

IP_NETWORK
The name of the MSCS network to which the preceding IP Address resource belongs. This parameter is optional. If it is not specified, the first MSCS network detected by the system is used. The name of the MSCS network must be entered exactly as seen under the Networks branch in Cluster Administrator.

Note: The previous four IP keywords are used to create an IP Address resource.

NETNAME_NAME
The name of the Network Name resource. Specify this parameter to create the Network Name resource. This parameter is optional for single partition database environment. You must specify this parameter for the instance owning machine in a partitioned database environment.

NETNAME_VALUE
The value for the Network Name resource. This parameter must be specified if the NETNAME_NAME parameter is specified.

NETNAME_DEPENDENCY
The name for the IP resource that the Network Name resource depends on. Each Network Name resource must have a dependency on an IP Address resource. This parameter is optional. If it is not specified, the Network Name resource has a dependency on the first IP resource in the group.

SERVICE_DISPLAY_NAME
The display name of the Generic Service resource. Specify this parameter if you want to create a Generic Service resource.

SERVICE_NAME
The service name of the Generic Service resource. This parameter must be specified if the SERVICE_DISPLAY_NAME parameter is specified.

SERVICE_STARTUP
Optional startup parameter for the Generic Resource service.

DISK_NAME
The name of the physical disk resource to be moved to the current group. Specify as many disk resources as you need. The disk resources must already exist. When the DB2MSCS utility configures the DB2 instance for failover support, the instance directory is copied to the first MSCS disk in the group. To specify a different MSCS disk for the instance directory, use the INSTPROF_DISK parameter. The disk name used should be entered exactly as seen in Cluster Administrator.

INSTPROF_DISK
An optional parameter to specify an MSCS disk to contain the DB2 instance directory. If this parameter is not specified the DB2MSCS utility uses the first disk that belongs to the same group.

INSTPROF_PATH
An optional parameter to specify the exact path where the instance directory will be copied. This parameter MUST be specified when using IPSHAdisks, a ServerRAID Netfinity disk resource (i.e. INSTPROF_PATH=p:\db2profs). INSTPROF_PATH will take precedence over INSTPROF_DISK if both are specified.

TARGET_DRVMAP_DISK
An optional parameter to specify the target MSCS disk for database drive mapping for a the multi-partitioned database system. This parameter will specify the disk the database will be created on by mapping it from the drive the create database command specifies. If

this parameter is not specified, the database drive mapping must be manually registered using the DB2DRVMP utility.

DB2_FALLBACK
An optional parameter to control whether or not the applications should be forced off when the DB2 resource is brought offline. If not specified, then the setting for DB2_FALLBACK will beYES. If you do not want the applications to be forced off, then set DB2_FALLBACK to NO.

CLP Commands

ARCHIVE LOG

Closes and truncates the active log file for a recoverable database. If user exit is enabled, an archive request is issued.

Authorization:

One of the following:

- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*

Required connection:

None. This command establishes a database connection for the duration of the command.

Command syntax:

```
►►──ARCHIVE LOG FOR──┬─DATABASE─┬──database-alias──────────────────────────►
                     └─DB───────┘

►──┬───────────────────────────────────┬───────────────────────────────────►
   └─USER──username──┬──────────────────┬─┘
                     └─USING──password──┘

►──┬─────────────────────────────────────────┬─────────────────────────────►◄
   └─┤ On Database Partition Number Clause ├─┘
```

On Database Partition Number Clause:

```
├──ON──────────────────────────────────────────────────────────────────────►

►──┬─┤ Database Partition Number List Clause ├───────────────────────┬─────┤
   └─ALL DBPARTITIONNUMS──┬─────────────────────────────────────────┬─┘
                          └─EXCEPT─┤ Database Partition Number List Clause ├─┘
```

Database Partition Number List Clause:

```
                          ┌─,──────────────────────────────────────────┐
                          ▼                                            │
├──┬─DBPARTITIONNUM──┬──(────db-partition-number──┬──────────────────────────┬─┴──)──┤
   └─DBPARTITIONNUMS─┘                            └─TO──db-partition-number─┘
```

Command parameters:

DATABASE database-alias
Specifies the alias of the database whose active log is to be archived.

USER username
Identifies the user name under which a connection will be attempted.

USING password
Specifies the password to authenticate the user name.

ON ALL DBPARTITIONNUMS
Specifies that the command should be issued on all database partitions in the `db2nodes.cfg` file. This is the default if a database partition number clause is not specified.

EXCEPT
Specifies that the command should be issued on all database partitions in the `db2nodes.cfg` file, except those specified in the database partition number list.

ON DBPARTITIONNUM/ON DBPARTITIONNUMS
Specifies that the logs should be archived for the specified database on a set of database partitions.

db-partition-number
Specifies a database partition number in the database partition number list.

TO db-partition-number
Used when specifying a range of database partitions for which the logs should be archived. All database partitions from the first

database partition number specified up to and including the second database partition number specified are included in the database partition number list.

Usage notes:

This command can be used to collect a complete set of log files up to a known point. The log files can then be used to update a standby database.

This command can only be executed when the invoking application or shell does not have a database connection to the specified database. This prevents a user from executing the command with uncommitted transactions. As such, the ARCHIVE LOG command will not forcibly commit the user's incomplete transactions. If the invoking application or shell already has a database connection to the specified database, the command will terminate and return an error. If another application has transactions in progress with the specified database when this command is executed, there will be a slight performance degradation since the command flushes the log buffer to disk. Any other transactions attempting to write log records to the buffer will have to wait until the flush is complete.

If used in a partitioned database environment, a subset of database partitions may be specified by using a database partition number clause. If the database partition number clause is not specified, the default behaviour for this command is to close and archive the active log on all database partitions.

Using this command will use up a portion of the active log space due to the truncation of the active log file. The active log space will resume its previous size when the truncated log becomes inactive. Frequent use of this command may drastically reduce the amount of the active log space available for transactions.

Compatibilities:

For compatibility with versions earlier than Version 8:

- The keyword NODE can be substituted for DBPARTITIONNUM.
- The keyword NODES can be substituted for DBPARTITIONNUMS.

INITIALIZE TAPE

When running on Windows NT-based operating systems, DB2 supports backup and restore operations to streaming tape devices. Use this command for tape initialization.

Authorization:

None

Required connection:

None

Command syntax:

```
►►──INITIALIZE TAPE──┬────────────┬──┬───────────────┬──────────────────────────►◄
                     └─ON──device─┘  └─USING──blksize─┘
```

Command parameters:

ON device
Specifies a valid tape device name. The default value is `\\.\TAPE0`.

USING blksize
Specifies the block size for the device, in bytes. The device is initialized to use the block size specified, if the value is within the supported range of block sizes for the device.

Note: The buffer size specified for the BACKUP DATABASE command and for RESTORE DATABASE must be divisible by the block size specified here.

If a value for this parameter is not specified, the device is initialized to use its default block size. If a value of zero is specified, the device is initialized to use a variable length block size; if the device does not support variable length block mode, an error is returned.

Related reference:

- "BACKUP DATABASE" on page 72
- "RESTORE DATABASE" on page 95
- "REWIND TAPE" on page 232
- "SET TAPE POSITION" on page 233

LIST HISTORY

Lists entries in the history file. The history file contains a record of recovery and administrative events. Recovery events include full database and table space level backup, incremental backup, restore, and rollforward operations. Additional logged events include create, alter, drop, or rename table space, reorganize table, drop table, and load.

Authorization:

None

Required connection:

Instance. An explicit attachment is not required. If the database is listed as remote, an instance attachment to the remote node is established for the duration of the command.

Command syntax:

```
►►──LIST HISTORY──┬───────────────────┬──┬─ALL──────────────────────────────┬──────────►
                  ├─BACKUP────────────┤  ├─SINCE──timestamp─────────────────┤
                  ├─ROLLFORWARD───────┤  └─CONTAINING──┬─schema.object_name─┬┘
                  ├─DROPPED TABLE─────┤                └─object_name────────┘
                  ├─LOAD──────────────┤
                  ├─CREATE TABLESPACE─┤
                  ├─ALTER TABLESPACE──┤
                  ├─RENAME TABLESPACE─┤
                  └─REORG─────────────┘

►─FOR──┬──────────┬─database-alias───────────────────────────────────────────────►◄
       ├─DATABASE─┤
       └─DB───────┘
```

Command parameters:

HISTORY
Lists all events that are currently logged in the history file.

BACKUP
Lists backup and restore operations.

ROLLFORWARD
Lists rollforward operations.

DROPPED TABLE
Lists dropped table records.

LOAD
Lists load operations.

CREATE TABLESPACE
Lists table space create and drop operations.

RENAME TABLESPACE
Lists table space renaming operations.

REORG
Lists reorganization operations.

ALTER TABLESPACE
Lists alter table space operations.

ALL Lists all entries of the specified type in the history file.

SINCE timestamp
A complete time stamp (format *yyyymmddhhnnss*), or an initial prefix (minimum *yyyy*) can be specified. All entries with time stamps equal to or greater than the time stamp provided are listed.

CONTAINING schema.object_name
This qualified name uniquely identifies a table.

CONTAINING object_name
This unqualified name uniquely identifies a table space.

FOR DATABASE database-alias
Used to identify the database whose recovery history file is to be listed.

Examples:

```
db2 list history since 19980201 for sample
db2 list history backup containing userspace1 for sample
db2 list history dropped table all for db sample
```

Usage notes:

The report generated by this command contains the following symbols:

```
Operation

   A - Create table space
   B - Backup
   C - Load copy
   D - Dropped table
   F - Roll forward
   G - Reorganize table
   L - Load
   N - Rename table space
   O - Drop table space
   Q - Quiesce
   R - Restore
   T - Alter table space
   U - Unload

Type

Backup types:

   F - Offline
   N - Online
   I - Incremental offline
   O - Incremental online
   D - Delta offline
   E - Delta online

Rollforward types:

   E - End of logs
   P - Point in time

Load types:

   I - Insert
   R - Replace

Alter tablespace types:

   C - Add containers
   R - Rebalance

Quiesce types:

   S - Quiesce share
   U - Quiesce update
   X - Quiesce exclusive
   Z - Quiesce reset
```

PRUNE HISTORY/LOGFILE

Used to delete entries from the recovery history file, or to delete log files from the active log file path. Deleting entries from the recovery history file may be necessary if the file becomes excessively large and the retention period is high. Deleting log files from the active log file path may be necessary if logs are being archived manually (rather than through a user exit program).

Authorization:

One of the following:

- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*

Required connection:

Database

Command syntax:

```
►►──PRUNE──┬─HISTORY──timestamp──┬───────────────────┬─┬──────────────────────►◄
           │                     └─WITH FORCE OPTION─┘ │
           └─LOGFILE PRIOR TO──log-file-name───────────┘
```

Command parameters:

HISTORY timestamp

Identifies a range of entries in the recovery history file that will be deleted. A complete time stamp (in the form *yyyymmddhhmmss*), or an initial prefix (minimum *yyyy*) can be specified. All entries with time stamps equal to or less than the time stamp provided are deleted from the recovery history file.

WITH FORCE OPTION

Specifies that the entries will be pruned according to the time stamp specified, even if some entries from the most recent restore set are deleted from the file. A restore set is the most recent full database backup including any restores of that backup image. If this parameter is not specified, all entries from the backup image forward will be maintained in the history.

LOGFILE PRIOR TO log-file-name

Specifies a string for a log file name, for example `S0000100.LOG`. All log files prior to (but not including) the specified log file will be deleted. The LOGRETAIN database configuration parameter must be set to `RECOVERY` or `CAPTURE`.

Examples:

To remove the entries for all restores, loads, table space backups, and full database backups taken before and including December 1, 1994 from the recovery history file, enter:

```
db2 prune history 199412
```

Note: 199412 is interpreted as 19941201000000.

Usage notes:

Pruning backup entries from the history file causes related file backups on DB2 Data Links Manager servers to be deleted.

REWIND TAPE

When running on Windows NT-based operating systems, DB2 supports backup and restore operations to streaming tape devices. Use this command for tape rewinding.

Authorization:

None

Required connection:

None

Command syntax:

```
►►──REWIND TAPE──┬─────────────┬──────────────────────────────────────────────►◄
                 └─ON──device──┘
```

Command parameters:

ON device
Specifies a valid tape device name. The default value is `\\.\TAPE0`.

Related reference:

- "INITIALIZE TAPE" on page 227
- "SET TAPE POSITION" on page 233

SET TAPE POSITION

When running on Windows NT-based operating systems, DB2 supports backup and restore operations to streaming tape devices. Use this command for tape positioning.

Authorization:

None

Required connection:

None

Command syntax:

```
►►──SET TAPE POSITION──┬─────────────┬──TO──position──────────────────────────►◄
                       └─ON──device──┘
```

Command parameters:

ON device
Specifies a valid tape device name. The default value is `\\.\TAPE0`.

TO position

Specifies the mark at which the tape is to be positioned. DB2 for Windows NT/2000 writes a tape mark after every backup image. A value of 1 specifies the first position, 2 specifies the second position, and so on. If the tape is positioned at tape mark 1, for example, archive 2 is positioned to be restored.

Related reference:

- "INITIALIZE TAPE" on page 227
- "REWIND TAPE" on page 232

UPDATE HISTORY FILE

Updates the location, device type, or comment in a history file entry.

Authorization:

One of the following:

- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*

Required connection:

Database

Command syntax:

```
►►──UPDATE HISTORY FOR──object-part──WITH──────────────────────────────────────►

►──┬─LOCATION──new-location──DEVICE TYPE──new-device-type─┬───────────────────►◄
   └─COMMENT──new-comment─────────────────────────────────┘
```

Command parameters:

FOR object-part

Specifies the identifier for the backup or copy image. It is a time stamp with an optional sequence number from 001 to 999.

LOCATION new-location

Specifies the new physical location of a backup image. The interpretation of this parameter depends on the device type.

DEVICE TYPE new-device-type
Specifies a new device type for storing the backup image. Valid device types are:

D	Disk
K	Diskette
T	Tape
A	TSM
U	User exit
P	Pipe
N	Null device
X	XBSA
Q	SQL statement
O	Other

COMMENT new-comment
Specifies a new comment to describe the entry.

Examples:

To update the history file entry for a full database backup taken on April 13, 1997 at 10:00 a.m., enter:

```
db2 update history for 19970413100000001 with
   location /backup/dbbackup.1 device type d
```

Usage notes:

The history file is used by database administrators for record keeping. It is used internally by DB2 for the automatic recovery of incremental backups.

Related reference:

- "PRUNE HISTORY/LOGFILE" on page 231

Appendix D. Additional APIs and Associated Data Structures

db2ArchiveLog - Archive Active Log API

Closes and truncates the active log file for a recoverable database. If user exit is enabled, issues an archive request.

Scope

In an MPP environment, this API closes and truncates the active logs on all nodes.

Authorization

One of the following:

- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*

Required Connection

This API automatically establishes a connection to the specified database. If a connection already exists, an error is returned.

API Include File

db2ApiDf.h

C API Syntax

```
/* File: db2ApiDf.h */
/* API: Archive Active Log */
/* ... */
SQL_API_RC SQL_API_FN
   db2ArchiveLog (
      db2Uint32 version,
      void * pDB2ArchiveLogStruct,
      struct sqlca * pSqlca);

typedef struct
{
   char * piDatabaseAlias;
   char * piUserName;
   char * piPassword;
   db2Uint16 iAllNodeFlag;
   db2Uint16 iNumNodes;
   SQL_PDB_NODE_TYPE * piNodeList;
   db2Uint32 iOptions;
} db2ArchiveLogStruct;
/* ... */
```

Generic API Syntax

```
/* File: db2ApiDf.h */
/* API: Archive Active Log */
/* ... */
SQL_API_RC SQL_API_FN
   db2gArchiveLog (
      db2Uint32 version,
      void * pDB2gArchiveLogStruct,
      struct sqlca * pSqlca);

typedef struct
{
   db2Uint32 iAliasLen;
   db2Uint32 iUserNameLen;
   db2Uint32 iPasswordLen;
   char * piDatabaseAlias;
   char * piUserName;
   char * piPassword;
   db2Uint16 iAllNodeFlag;
   db2Uint16 iNumNodes;
   SQL_PDB_NODE_TYPE * piNodeList;
   db2Uint32 iOptions;
} db2gArchiveLogStruct;
/* ... */
```

API Parameters

version
Input. Specifies the version and release level of the variable passed in as the second parameter, *pDB2ArchiveLogStruct*.

pDB2ArchiveLogStruct
Input. A pointer to the *db2ArchiveLogStruct* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

iAliasLen
Input. A 4-byte unsigned integer representing the length in bytes of the database alias.

iUserNameLen
Input. A 4-byte unsigned integer representing the length in bytes of the user name. Set to zero if no user name is used.

iPasswordLen
Input. A 4-byte unsigned integer representing the length in bytes of the password. Set to zero if no password is used.

piDatabaseAlias
Input. A string containing the database alias (as cataloged in the system database directory) of the database for which the active log is to be archived.

piUserName
Input. A string containing the user name to be used when attempting a connection.

piPassword
Input. A string containing the password to be used when attempting a connection.

iAllNodeFlag
Input. MPP only. Flag indicating whether the operation should apply to all nodes listed in the `db2nodes.cfg` file. Valid values are:

DB2ARCHIVELOG_ALL_NODES
Apply to all nodes (piNodeList should be NULL). This is the default value.

DB2ARCHIVELOG_NODE_LIST
Apply to all nodes specified in a node list that is passed in *piNodeList*.

DB2ARCHIVELOG_ALL_EXCEPT
Apply to all nodes *except* those specified in a node list that is passed in *piNodeList*.

iNumNodes
Input. MPP only. Specifies the number of nodes in the *piNodeList* array.

piNodeList
Input. MPP only. A pointer to an array of node numbers against which to apply the archive log operation.

iOptions
Input. Reserved for future use.

Usage Notes

This API can be used to collect a complete set of log files up to a known point. The log files can then be used to update a standby database.

If other applications have transactions in progress when this API is called, a slight performance decrement will be noticed when the log buffer is flushed to disk; other transactions attempting to write log records to the buffer must wait until the flush has completed.

This API causes the database to lose a portion of its LSN space, thereby hastening the exhaustion of valid LSNs.

db2HistoryCloseScan - Close History File Scan

Ends a history file scan and frees DB2 resources required for the scan. This API must be preceded by a successful call to db2HistoryOpenScan.

Authorization:

None

Required connection:

Instance. It is not necessary to call sqleatin before calling this API.

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2HistoryCloseScan */
/* ... */
SQL_API_RC  SQL_API_FN
  db2HistoryCloseScan (
    db2Uint32 version,
    void *piHandle,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: db2ApiDf.h */
/* API: db2GenHistoryCloseScan */
/* ... */
SQL_API_RC  SQL_API_FN
  db2GenHistoryCloseScan (
    db2Uint32 version,
    void *piHandle,
    struct sqlca *pSqlca);
/* ... */
```

API parameters:

version
Input. Specifies the version and release level of the second parameter, *piHandle*.

piHandle
Input. Specifies a pointer to the handle for scan access that was returned by db2HistoryOpenScan.

pSqlca
Output. A pointer to the *sqlca* structure.

REXX API syntax:

```
CLOSE RECOVERY HISTORY FILE :scanid
```

REXX API parameters:

scanid Host variable containing the scan identifier returned from OPEN RECOVERY HISTORY FILE SCAN.

Usage notes:

For a detailed description of the use of the history file APIs, see db2HistoryOpenScan.

Related reference:

- "db2Prune - Prune History File" on page 253
- "db2HistoryUpdate - Update History File" on page 250
- "db2HistoryOpenScan - Open History File Scan" on page 245
- "db2HistoryGetEntry - Get Next History File Entry" on page 242
- "SQLCA" in the *Administrative API Reference*

Related samples:

- "dbrecov.sqc -- How to recover a database (C)"
- "dbrecov.sqC -- How to recover a database (C++)"

db2HistoryGetEntry - Get Next History File Entry

Gets the next entry from the history file. This API must be preceded by a successful call to db2HistoryOpenScan.

Authorization:

None

Required connection:

Instance. It is not necessary to call sqleatin before calling this API.

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2HistoryGetEntry */
/* ... */
SQL_API_RC SQL_API_FN
  db2HistoryGetEntry (
    db2Uint32 version,
    void *pDB2HistoryGetEntryStruct,
    struct sqlca *pSqlca);

typedef struct
{
  db2Uint16 iHandle,
  db2Uint16 iCallerAction,
  struct db2HistData *pioHistData
} db2HistoryGetEntryStruct;
/* ... */
```

Generic API syntax:

```
/* File: db2ApiDf.h */
/* API: db2GenHistoryGetEntry */
/* ... */
SQL_API_RC SQL_API_FN
  db2GenHistoryGetEntry (
    db2Uint32 version,
    void *pDB2GenHistoryGetEntryStruct,
    struct sqlca *pSqlca);

typedef struct
{
  db2Uint16 iHandle,
  db2Uint16 iCallerAction,
  struct db2HistData *pioHistData
} db2GenHistoryGetEntryStruct;
/* ... */
```

API parameters:

version
Input. Specifies the version and release level of the structure passed in as the second parameter, *pDB2HistoryGetEntryStruct*.

pDB2HistoryGetEntryStruct
Input. A pointer to the *db2HistoryGetEntryStruct* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

iHandle
Input. Contains the handle for scan access that was returned by db2HistoryOpenScan.

iCallerAction
Input. Specifies the type of action to be taken. Valid values (defined in `db2ApiDf`) are:

DB2HISTORY_GET_ENTRY
Get the next entry, but without any command data.

DB2HISTORY_GET_DDL
Get only the command data from the previous fetch.

DB2HISTORY_GET_ALL
Get the next entry, including all data.

pioHistData
Input. A pointer to the *db2HistData* structure.

REXX API syntax:

```
GET RECOVERY HISTORY FILE ENTRY :scanid [USING :value]
```

REXX API parameters:

scanid Host variable containing the scan identifier returned from OPEN RECOVERY HISTORY FILE SCAN.

value A compound REXX host variable into which the history file entry information is returned. In the following, XXX represents the host variable name:

XXX.0	Number of first level elements in the variable (always 15)
XXX.1	Number of table space elements
XXX.2	Number of used table space elements
XXX.3	OPERATION (type of operation performed)
XXX.4	OBJECT (granularity of the operation)
XXX.5	OBJECT_PART (time stamp and sequence number)
XXX.6	OPTYPE (qualifier of the operation)
XXX.7	DEVICE_TYPE (type of device used)
XXX.8	FIRST_LOG (earliest log ID)
XXX.9	LAST_LOG (current log ID)
XXX.10	BACKUP_ID (identifier for the backup)
XXX.11	SCHEMA (qualifier for the table name)
XXX.12	TABLE_NAME (name of the loaded table)

XXX.13.0	NUM_OF_TABLESPACES (number of table spaces involved in backup or restore)
XXX.13.1	Name of the first table space backed up/restored
XXX.13.2	Name of the second table space backed up/restored
XXX.13.3	and so on
XXX.14	LOCATION (where backup or copy is stored)
XXX.15	COMMENT (text to describe the entry).

Usage notes:

The records that are returned will have been selected using the values specified on the call to db2HistoryOpenScan.

For a detailed description of the use of the history file APIs, see db2HistoryOpenScan.

Related reference:

- "db2Prune - Prune History File" on page 253
- "db2HistoryUpdate - Update History File" on page 250
- "db2HistoryOpenScan - Open History File Scan" on page 245
- "db2HistoryCloseScan - Close History File Scan" on page 241
- "SQLCA" in the *Administrative API Reference*
- "db2HistData" on page 267

Related samples:

- "dbrecov.sqc -- How to recover a database (C)"
- "dbrecov.sqC -- How to recover a database (C++)"

db2HistoryOpenScan - Open History File Scan

Starts a history file scan.

Authorization:

None

Required connection:

Instance. It is not necessary to call sqleatin before calling this API. If the database is cataloged as remote, an instance attachment to the remote node is established.

db2HistoryOpenScan - Open History File Scan

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2HistoryOpenScan */
/* ... */
SQL_API_RC SQL_API_FN
  db2HistoryOpenScan (
    db2Uint32 version,
    void *pDB2HistoryOpenStruct,
    struct sqlca *pSqlca);

typedef struct
{
  char *piDatabaseAlias,
  char *piTimestamp,
  char *piObjectName,
  db2Uint32 oNumRows,
  db2Uint16 iCallerAction,
  db2Uint16 oHandle
} db2HistoryOpenStruct;
/* ... */
```

Generic API syntax:

```
/* File: db2ApiDf.h */
/* API: db2GenHistoryOpenScan */
/* ... */
SQL_API_RC SQL_API_FN
  db2GenHistoryOpenScan (
    db2Uint32 version,
    void *pDB2GenHistoryOpenStruct,
    struct sqlca *pSqlca);

typedef struct
{
  char *piDatabaseAlias,
  char *piTimestamp,
  char *piObjectName,
  db2Uint32 oNumRows,
  db2Uint16 iCallerAction,
  db2Uint16 oHandle
} db2GenHistoryOpenStruct;
/* ... */
```

API parameters:

version

Input. Specifies the version and release level of the structure passed in as the second parameter, *pDB2HistoryOpenStruct*.

pDB2HistoryOpenStruct
Input. A pointer to the *db2HistoryOpenStruct* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

piDatabaseAlias
Input. A pointer to a string containing the database alias.

piTimestamp
Input. A pointer to a string specifying the time stamp to be used for selecting records. Records whose time stamp is equal to or greater than this value are selected. Setting this parameter to NULL, or pointing to zero, prevents the filtering of entries using a time stamp.

piObjectName
Input. A pointer to a string specifying the object name to be used for selecting records. The object may be a table or a table space. If it is a table, the fully qualified table name must be provided. Setting this parameter to NULL, or pointing to zero, prevents the filtering of entries using the object name.

oNumRows
Output. Upon return from the API, this parameter contains the number of matching history file entries.

iCallerAction
Input. Specifies the type of action to be taken. Valid values (defined in `db2ApiDf`) are:

DB2HISTORY_LIST_HISTORY
Lists all events that are currently logged in the history file.

DB2HISTORY_LIST_BACKUP
Lists backup and restore operations.

DB2HISTORY_LIST_ROLLFORWARD
Lists rollforward operations.

DB2HISTORY_LIST_DROPPED_TABLE
Lists dropped table records. The DDL field associated with an entry is not returned. To retrieve the DDL information for an entry, db2HistoryGetEntry must be called with a caller action of `DB2HISTORY_GET_DDL` immediately after the entry is fetched.

DB2HISTORY_LIST_LOAD
Lists load operations.

DB2HISTORY_LIST_CRT_TABLESPACE
Lists table space create and drop operations.

DB2HISTORY_LIST_REN_TABLESPACE
Lists table space renaming operations.

DB2HISTORY_LIST_ALT_TABLESPACE
Lists alter table space operations. The DDL field associated with an entry is not returned. To retrieve the DDL information for an entry, db2HistoryGetEntry must be called with a caller action of `DB2HISTORY_GET_DDL` immediately after the entry is fetched.

DB2HISTORY_LIST_REORG
Lists REORGANIZE TABLE operations. This value is not currently supported.

oHandle
Output. Upon return from the API, this parameter contains the handle for scan access. It is subsequently used in db2HistoryGetEntry, and db2HistoryCloseScan.

REXX API syntax:

```
OPEN [BACKUP] RECOVERY HISTORY FILE FOR database_alias
[OBJECT objname] [TIMESTAMP :timestamp]
USING :value
```

REXX API parameters:

database_alias
The alias of the database whose history file is to be listed.

objname
Specifies the object name to be used for selecting records. The object may be a table or a table space. If it is a table, the fully qualified table name must be provided. Setting this parameter to NULL prevents the filtering of entries using *objname*.

timestamp
Specifies the time stamp to be used for selecting records. Records whose time stamp is equal to or greater than this value are selected. Setting this parameter to NULL prevents the filtering of entries using *timestamp*.

value A compound REXX host variable to which history file information is returned. In the following, XXX represents the host variable name.

XXX.0 Number of elements in the variable (always 2)

XXX.1 Identifier (handle) for future scan access

XXX.2 Number of matching history file entries.

Usage notes:

The combination of time stamp, object name and caller action can be used to filter records. Only records that pass all specified filters are returned.

The filtering effect of the object name depends on the value specified:

- Specifying a table will return records for load operations, because this is the only information for tables in the history file.
- Specifying a table space will return records for backup, restore, and load operations for the table space.

Note: To return records for tables, they must be specified as *schema.tablename*. Specifying *tablename* will only return records for table spaces.

A maximum of eight history file scans per process is permitted.

To list every entry in the history file, a typical application will perform the following steps:

1. Call db2HistoryOpenScan, which will return *oNumRows*.
2. Allocate an *db2HistData* structure with space for *n oTablespace* fields, where *n* is an arbitrary number.
3. Set the *iDB2NumTablespace* field of the *db2HistData* structure to *n*.
4. In a loop, perform the following:
   - Call db2HistoryGetEntry to fetch from the history file.
   - If db2HistoryGetEntry returns an SQLCODE of `SQL_RC_OK`, use the *sqld* field of the *db2HistData* structure to determine the number of table space entries returned.
   - If db2HistoryGetEntry returns an SQLCODE of `SQLUH_SQLUHINFO_VARS_WARNING`, not enough space has been allocated for all of the table spaces that DB2 is trying to return; free and reallocate the *db2HistData* structure with enough space for *oDB2UsedTablespace* table space entries, and set *iDB2NumTablespace* to *oDB2UsedTablespace*.
   - If db2HistoryGetEntry returns an SQLCODE of `SQLE_RC_NOMORE`, all history file entries have been retrieved.
   - Any other SQLCODE indicates a problem.
5. When all of the information has been fetched, call db2HistoryCloseScan to free the resources allocated by the call to db2HistoryOpenScan.

The macro SQLUHINFOSIZE(*n*) (defined in `sqlutil`) is provided to help determine how much memory is required for an *db2HistData* structure with space for *n oTablespace* fields.

Related reference:

- "db2Prune - Prune History File" on page 253
- "db2HistoryUpdate - Update History File" on page 250
- "db2HistoryGetEntry - Get Next History File Entry" on page 242
- "db2HistoryCloseScan - Close History File Scan" on page 241
- "SQLCA" in the *Administrative API Reference*

Related samples:

- "dbrecov.sqc -- How to recover a database (C)"
- "dbrecov.sqC -- How to recover a database (C++)"

db2HistoryUpdate - Update History File

Updates the location, device type, or comment in a history file entry.

Authorization:

One of the following:

- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*

Required connection:

Database. To update entries in the history file for a database other than the default database, a connection to the database must be established before calling this API.

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2HistoryUpdate */
/* ... */
SQL_API_RC SQL_API_FN
  db2HistoryUpdate (
    db2Uint32 version,
    void *pDB2HistoryUpdateStruct,
    struct sqlca *pSqlca);

typedef struct
{
  char *piNewLocation,
  char *piNewDeviceType,
  char *piNewComment,
  db2Uint32 iEID
} db2HistoryUpdateStruct;
/* ... */
```

Generic API syntax:

```
/* File: db2ApiDf.h */
/* API: db2GenHistoryUpdate */
/* ... */
SQL_API_RC SQL_API_FN
  db2GenHistoryUpdate (
    db2Uint32 version,
    void *pDB2GenHistoryUpdateStruct,
    struct sqlca *pSqlca);

typedef struct
{
  char *piNewLocation,
  char *piNewDeviceType,
  char *piNewComment,
  db2Uint32 iEID
} db2GenHistoryUpdateStruct;
/* ... */
```

API parameters:

version
Input. Specifies the version and release level of the structure passed in as the second parameter, *pDB2HistoryUpdateStruct*.

pDB2HistoryUpdateStruct
Input. A pointer to the *db2HistoryUpdateStruct* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

piNewLocation
Input. A pointer to a string specifying a new location for the backup, restore, or load copy image. Setting this parameter to NULL, or pointing to zero, leaves the value unchanged.

piNewDeviceType
Input. A pointer to a string specifying a new device type for storing the backup, restore, or load copy image. Setting this parameter to NULL, or pointing to zero, leaves the value unchanged.

piNewComment
Input. A pointer to a string specifying a new comment to describe the entry. Setting this parameter to NULL, or pointing to zero, leaves the comment unchanged.

iEID Input. A unique identifier that can be used to update a specific entry in the history file.

REXX API syntax:

```
UPDATE RECOVERY HISTORY USING :value
```

REXX API parameters:

value A compound REXX host variable containing information pertaining to the new location of a history file entry. In the following, XXX represents the host variable name:

XXX.0 Number of elements in the variable (must be between 1 and 4)

XXX.1 OBJECT_PART (time stamp with a sequence number from 001 to 999)

XXX.2 New location for the backup or copy image (this parameter is optional)

XXX.3 New device used to store the backup or copy image (this parameter is optional)

XXX.4 New comment (this parameter is optional).

Usage notes:

This is an update function, and all information prior to the change is replaced and cannot be recreated. These changes are not logged.

The history file is used for recording purposes only. It is not used directly by the restore or the rollforward functions. During a restore operation, the location of the backup image can be specified, and the history file is useful for tracking this location. The information can subsequently be provided to the backup utility. Similarly, if the location of a load copy image is moved, the rollforward utility must be provided with the new location and type of storage media.

Related reference:

- "db2Rollforward - Rollforward Database" on page 145
- "db2Prune - Prune History File" on page 253
- "db2HistoryOpenScan - Open History File Scan" on page 245
- "db2HistoryGetEntry - Get Next History File Entry" on page 242
- "db2HistoryCloseScan - Close History File Scan" on page 241
- "SQLCA" in the *Administrative API Reference*
- "db2Backup - Backup database" on page 77

Related samples:

- "dbrecov.sqc -- How to recover a database (C)"
- "dbrecov.sqC -- How to recover a database (C++)"

db2Prune - Prune History File

Deletes entries from the history file or log files from the active log path.

Authorization:

One of the following:

- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*

Required connection:

Database. To delete entries from the history file for any database other than the default database, a connection to the database must be established before calling this API.

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2Prune */
/* ... */
SQL_API_RC SQL_API_FN
  db2Prune (
    db2Uint32 version,
    void *pDB2PruneStruct,
    struct sqlca *pSqlca);

typedef struct
{
  char *piString,
  db2Uint32 iEID,
  db2Uint32 iCallerAction,
  db2Uint32 iOptions
} db2PruneStruct;
/* ... */
```

Generic API syntax:

```
/* File: db2ApiDf.h */
/* API: db2GenPrune */
/* ... */
SQL_API_RC SQL_API_FN
  db2GenPrune (
    db2Uint32 version,
    void *pDB2GenPruneStruct,
    struct sqlca *pSqlca);

typedef struct
{
  db2Uint32 iStringLen;
  char *piString,
  db2Uint32 iEID,
  db2Uint32 iCallerAction,
  db2Uint32 iOptions
} db2GenPruneStruct;
/* ... */
```

API parameters:

version
Input. Specifies the version and release level of the structure passed in as the second parameter, *pDB2PruneStruct*.

pDB2PruneStruct
Input. A pointer to the *db2PruneStruct* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

iStringLen
Input. Specifies the length in bytes of *piString*.

piString
Input. A pointer to a string specifying a time stamp or a log sequence number (LSN). The time stamp or part of a time stamp (minimum *yyyy*, or year) is used to select records for deletion. All entries equal to or less than the time stamp will be deleted. A valid time stamp must be provided; there is no default behavior for a NULL parameter.

This parameter can also be used to pass an LSN, so that inactive logs can be pruned.

iEID Input. Specifies a unique identifier that can be used to prune a single entry from the history file.

iCallerAction
Input. Specifies the type of action to be taken. Valid values (defined in `db2ApiDf`) are:

DB2PRUNE_ACTION_HISTORY
Remove history file entries.

DB2PRUNE_ACTION_LOG
Remove log files from the active log path.

iOptions
Input. Valid values (defined in `db2ApiDf`) are:

DB2PRUNE_OPTION_FORCE
Force the removal of the last backup.

DB2PRUNE_OPTION_LSNSTRING
Specify that the value of *piString* is an LSN, used when a caller action of `DB2PRUNE_ACTION_LOG` is specified.

REXX API syntax:

```
PRUNE RECOVERY HISTORY BEFORE :timestamp [WITH FORCE OPTION]
```

REXX API parameters:

timestamp
A host variable containing a time stamp. All entries with time stamps equal to or less than the time stamp provided are deleted from the history file.

WITH FORCE OPTION
If specified, the history file will be pruned according to the time stamp specified, even if some entries from the most recent restore set are deleted from the file. If not specified, the most recent restore set will be kept, even if the time stamp is less than or equal to the time stamp specified as input.

Usage notes:

Pruning the history file does not delete the actual backup or load files. The user must manually delete these files to free up the space they consume on storage media.

CAUTION:
If the latest full database backup is deleted from the media (in addition to being pruned from the history file), the user must ensure that all table spaces, including the catalog table space and the user table spaces, are backed up. Failure to do so may result in a database that cannot be recovered, or the loss of some portion of the user data in the database.

Related reference:

- "db2HistoryUpdate - Update History File" on page 250
- "db2HistoryOpenScan - Open History File Scan" on page 245
- "db2HistoryGetEntry - Get Next History File Entry" on page 242
- "db2HistoryCloseScan - Close History File Scan" on page 241
- "SQLCA" in the *Administrative API Reference*

Related samples:

- "dbrecov.sqc -- How to recover a database (C)"
- "dbrecov.sqC -- How to recover a database (C++)"

db2ReadLogNoConn - Read Log Without a Database Connection

Extract log records from the DB2 UDB database logs and query the Log Manager for current log state information. Prior to using this API, use db2ReadLogNoConnInit to allocate the memory that is passed as an input parameter to this API. After using this API, use db2ReadLogNoConnTerm to deallocate the memory.

Authorization:

One of the following:

- *sysadm*
- *dbadm*

Required connection:

Database

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2ReadLogNoConn */
/* ... */
SQL_API_RC SQL_API_FN
db2ReadLogNoConn (
  db2Uint32 versionNumber,
  void *pDB2ReadLogNoConnStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2ReadLogNoConnStruct
{
  db2Uint32                  iCallerAction;
  SQLU_LSN                   *piStartLSN;
  SQLU_LSN                   *piEndLSN;
  char                       *poLogBuffer;
  db2Uint32                  iLogBufferSize;
  char                       *piReadLogMemPtr;
  db2ReadLogNoConnInfoStruct  *poReadLogInfo;
} db2ReadLogNoConnStruct;

typedef SQL_STRUCTURE db2ReadLogNoConnInfoStruct
{
  SQLU_LSN                   firstAvailableLSN;
  SQLU_LSN                   firstReadLSN;
  SQLU_LSN                   nextStartLSN;
  db2Uint32                  logRecsWritten;
  db2Uint32                  logBytesWritten;
  db2Uint32                  lastLogFullyRead;
  db2TimeOfLog               currentTimeValue;
} db2ReadLogNoConnInfoStruct;

/* ... */
```

API parameters:

version
Input. Specifies the version and release level of the structure passed as the second parameter *pDB2ReadLogNoConnStruct*.

pParamStruct
Input. A pointer to the *db2ReadLogNoConnStruct* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

iCallerAction
Input. Specifies the action to be performed. Valid values are:

DB2READLOG_READ
Read the database log from the starting log sequence to the ending log sequence number and return log records within this range.

DB2READLOG_READ_SINGLE
Read a single log record (propagatable or not) identified by the starting log sequence number.

DB2READLOG_QUERY
Query the database log. Results of the query will be sent back via the db2ReadLogNoConnInfoStruct structure.

piStartLSN
Input. The starting log sequence number specifies the starting relative byte address for the reading of the log. This value must be the start of an actual log record.

piEndLSN
Input. The ending log sequence number specifies the ending relative byte address for the reading of the log. This value must be greater than *piStartLsn*, and does not need to be the end of an actual log record.

poLogBuffer
Output. The buffer where all the propagatable log records read within the specified range are stored sequentially. This buffer must be large enough to hold a single log record. As a guideline, this buffer should be a minimum of 32 bytes. Its maximum size is dependent on the size of the requested range. Each log record in the buffer is prefixed by a six byte log sequence number (LSN), representing the LSN of the following log record.

iLogBufferSize
Input. Specifies the size, in bytes, of the log buffer.

piReadLogMemPtr
Input. Block of memory of size iReadLogMemoryLimit that was allocated in the initialization call. This memory contains persistent data that the API requires at each invocation. This memory block must not be reallocated or altered in any way by the caller.

poReadLogInfo
Output. A pointer to the *db2ReadLogNoConnInfoStruct* structure.

firstAvailableLSN
First available LSN in available logs.

firstReadLSN
First LSN read on this call.

nextStartLSN
Next readable LSN.

logRecsWritten
Number of log records written to the log buffer field, *poLogBuffer*.

logBytesWritten
Number of bytes written to the log buffer field, *poLogBuffer*.

lastLogFullyRead
Number indicating the last log file that was read to completion.

Usage notes:

The db2ReadLogNoConn API requires a memory block that must be allocated using the db2ReadLogNoConnInit API. The memory block must be passed as an input parameter to all subsequent db2ReadLogNoConn API calls, and must not be altered.

When requesting a sequential read of log, the API requires a log sequence number (LSN) range and the allocated memory . The API will return a sequence of log records based on the filter option specified when initialized and the LSN range. When requesting a query, the read log information structure will contain a valid starting LSN, to be used on a read call. The value used as the ending LSN on a read can be one of the following:

- A value greater than the caller-specified startLSN.
- FFFF FFFF FFFF which is interpreted by the asynchronous log reader as the end of the available logs.

The propagatable log records read within the starting and ending LSN range are returned in the log buffer. A log record does not contain its LSN, it is contained in the buffer before the actual log record. Descriptions of the various DB2 UDB log records returned by db2ReadLogNoConn can be found in the DB2 UDB Log Records section.

After the initial read, in order to read the next sequential log record, use the nextStartLSN value returned in db2ReadLogNoConnInfoStruct. Resubmit the call, with this new starting LSN and a valid ending LSN and the next block of records is then read. An sqlca code of SQLU_RLOG_READ_TO_CURRENT means the log reader has read to the end of the available log files.

When the API will no longer be used, use db2ReadLogNoConnTerm to terminate the memory.

Related reference:

- "db2ReadLogNoConnInit - Initialize Read Log Without a Database Connection" on page 260

- "db2ReadLogNoConnTerm - Terminate Read Log Without a Database Connection" on page 262

db2ReadLogNoConnInit - Initialize Read Log Without a Database Connection

Allocates the memory to be used by db2ReadLogNoConn in order to extract log records from the DB2 UDB database logs and query the Log Manager for current log state information.

Required connection:

Database

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2ReadLogNoConnInit */
/* ... */
SQL_API_RC SQL_API_FN
db2ReadLogNoConnInit (
  db2Uint32 versionNumber,
  void * pDB2ReadLogNoConnInitStruct,
  struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2ReadLogNoConnInitStruct
{
  db2Uint32                   iFilterOption;
  char                        *piLogFilePath;
  char                        *piOverflowLogPath;
  db2Uint32                   iRetrieveLogs;
  char                        *piDatabaseName;
  char                        *piNodeName;
  db2Uint32                   iReadLogMemoryLimit;
  char                        **poReadLogMemPtr;
} db2ReadLogNoConnInitStruct;
/* ... */
```

API parameters:

version
Input. Specifies the version and release level of the structure passed as the second parameter *pDB2ReadLogNoConnInitStruct*.

pParamStruct
Input. A pointer to the *db2ReadLogNoConnInitStruct* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

iFilterOption
Input. Specifies the level of log record filtering to be used when reading the log records. Valid values are:

DB2READLOG_FILTER_OFF
Read all log records in the given LSN range.

DB2READLOG_FILTER_ON
Reads only log records in the given LSN range marked as propagatable. This is the traditional behavior of the asynchronous log read API.

piLogFilePath
Input. Path where the log files to be read are located.

piOverflowLogPath
Input. Alternate path where the log files to be read may be located.

iRetrieveLogs
Input. Option specifying if userexit should be invoked to retrieve log files that cannot be found in either the log file path or the overflow log path. Valid values are:

DB2READLOG_RETRIEVE_OFF
Userexit should not be invoked to retrieve missing log files.

DB2READLOG_RETRIEVE_LOGPATH
Userexit should be invoked to retrieve missing log files into the specified log file path.

DB2READLOG_RETRIEVE_OVERFLOW
Userexit should be invoked to retrieve missing log files into the specified overflow log path.

piDatabaseName
Input. Name of the database that owns the recovery logs being read. This is required if the retrieve option above is specified.

piNodeName
Input. Name of the node that owns the recovery logs being read. This is required if the retrieve option above is specified.

iReadLogMemoryLimit
Input. Maximum number of bytes that the API may allocate internally.

poReadLogMemPtr
Output. API-allocated block of memory of size *iReadLogMemoryLimit*.

This memory contains persistent data that the API requires at each invocation. This memory block must not be reallocated or altered in any way by the caller.

Usage notes:

The memory initialized by db2ReadLogNoConnInit must not be altered.

When db2ReadLogNoConn will no longer be used, invoke db2ReadLogNoConnTerm to deallocate the memory initialized by db2ReadLogNoConnInit.

Related reference:

- "db2ReadLogNoConn - Read Log Without a Database Connection" on page 256
- "db2ReadLogNoConnTerm - Terminate Read Log Without a Database Connection" on page 262

db2ReadLogNoConnTerm - Terminate Read Log Without a Database Connection

Deallocates the memory used by db2ReadLogNoConn, originally initialized by db2ReadLogNoConnInit.

Authorization:

One of the following:

- *sysadm*
- *dbadm*

Required connection:

Database

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2ReadLogNoConnTerm */
/* ... */
SQL_API_RC SQL_API_FN
db2ReadLogNoConnTerm (
  db2Uint32 versionNumber,
  void * pDB2ReadLogNoConnTermStruct,
  struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2ReadLogNoConnTermStruct
{
  char                        **poReadLogMemPtr;
} db2ReadLogNoConnTermStruct;
/* ... */
```

API parameters:

version
Input. Specifies the version and release level of the structure passed as the second parameter *pDB2ReadLogNoConnTermStruct*.

pParamStruct
Input. A pointer to the *db2ReadLogNoConnTermStruct* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

poReadLogMemPtr
Output. Pointer to the block of memory allocated in the initialization call. This pointer will be freed and set to NULL.

Related reference:

- “db2ReadLogNoConn - Read Log Without a Database Connection” on page 256
- “db2ReadLogNoConnInit - Initialize Read Log Without a Database Connection” on page 260

db2ReadLog - Asynchronous Read Log

Extract log records from the DB2 UDB database logs and the Log Manager for current log state information. This API can only be used with recoverable databases. A database is recoverable if it is configured with *logretain* set to `RECOVERY` or *userexit* set to `ON`.

Authorization:

One of the following:

- *sysadm*
- *dbadm*

Required connection:

Database

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2ReadLog */
/* ... */
SQL_API_RC SQL_API_FN
  db2ReadLog (
    db2Uint32 versionNumber,
    void *pDB2ReadLogStruct,
    struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2ReadLogStruct
{
  db2Uint32                   iCallerAction;
  SQLU_LSN                    *piStartLSN;
  SQLU_LSN                    *piEndLSN;
  char                        *poLogBuffer;
  db2Uint32                   iLogBufferSize;
  db2Uint32                   iFilterOption;
  db2ReadLogInfoStruct        *poReadLogInfo;

typedef SQL_STRUCTURE db2ReadLogInfoStruct
{
  SQLU_LSN                    initialLSN;
  SQLU_LSN                    firstReadLSN;
  SQLU_LSN                    nextStartLSN;
  db2Uint32                   logRecsWritten;
  db2Uint32                   logBytesWritten;
  SQLU_LSN                    firstReusedLSN;
  db2Uint32                   timeOfLSNReuse;
  db2TimeOfLog                currentTimeValue;
} db2ReadLogInfoStruct;

typedef SQL_STRUCTURE db2TimeOfLog
{
  db2Uint32                   seconds;
  db2Uint32                   accuracy;
} db2TimeOfLog;
/* ... */
```

API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter, *pDB2ReadLogStruct*.

pDB2ReadLogStruct
Input. A pointer to the *db2ReadLogStruct*.

pSqlca
Output. A pointer to the *sqlca* structure.

iCallerAction
Input. Specifies the action to be performed.

DB2READLOG_READ
Read the database log from the starting log sequence to the ending log sequence number and return log records within this range.

DB2READLOG_READ_SINGLE
Read a single log record (propagatable or not) identified by the starting log sequence number.

DB2READLOG_QUERY
Query the database log. Results of the query will be sent back via the *db2ReadLogInfoStruct* structure.

piStartLsn
Input. The starting log sequence number specifies the starting relative byte address for the reading of the log. This value must be the start of an actual log record.

piEndLsn
Input. The ending log sequence number specifies the ending relative byte address for the reading of the log. This value must be greater than *startLsn*, and does not need to be the end of an actual log record.

poLogBuffer
Output. The buffer where all the propagatable log records read within the specified range are stored sequentially. This buffer must be large enough to hold a single log record. As a guideline, this buffer should be a minimum of 32 bytes. Its maximum size is dependent on the size of the requested range. Each log record in the buffer is prefixed by a six byte log sequence number (LSN), representing the LSN of the following log record.

iLogBufferSize
Input. Specifies the size, in bytes, of the log buffer.

iFilterOption
Input. Specifies the level of log record filtering to be used when reading the log records. Valid values are:

DB2READLOG_FILTER_OFF
Read all log records in the given LSN range.

DB2READLOG_FILTER_ON
Reads only log records in the given LSN range marked as propagatable. This is the traditional behaviors of the asynchronous log read API.

poReadLogInfo
Output. A structure detailing information regarding the call and the database log.

Usage notes:

If the requested action is to read the log, the caller will provide a log sequence number range and a buffer to hold the log records. This API reads the log sequentially, bounded by the requested LSN range, and returns log records associated with tables having the DATA CAPTURE option CHANGES, and a db2ReadLogInfoStruct structure with the current active log information. If the requested action is query, the API returns an db2ReadLogInfoStruct structure with the current active log information.

To use the Asynchronous Log Reader, first query the database log for a valid starting LSN. Following the query call, the read log information structure (db2ReadLogInfoStruct) will contain a valid starting LSN (in the initialLSN member), to be used on a read call. The value used as the ending LSN on a read can be one of the following:

- A value greater than initialLSN
- `FFFF FFFF FFFF`, which is interpreted by the asynchronous log reader as the end of the current log.

The propagatable log records read within the starting and ending LSN range are returned in the log buffer. A log record does not contain its LSN; it is contained in the buffer before the actual log record. Descriptions of the various DB2 log records returned by db2ReadLog the DB2 UDB Log Records section.

To read the next sequential log record after the initial read, use the nextStartLSN field returned in the db2ReadLogStruct structure. Resubmit the call, with this new starting LSN and a valid ending LSN. The next block of records is then read. An *sqlca* code of SQLU_RLOG_READ_TO_CURRENT means that the log reader has read to the end of the current active log.

Related reference:

- “SQLCA” in the *Administrative API Reference*

Related samples:

- “dbrecov.sqc -- How to recover a database (C)”

- "dbrecov.sqC -- How to recover a database (C++)"

db2HistData

This structure is used to return information after a call to db2HistoryGetEntry.

Table 2. Fields in the db2HistData Structure

Field Name	Data Type	Description
ioHistDataID	char(8)	An 8-byte structure identifier and "eye-catcher" for storage dumps. The only valid value is "`SQLUHINF`". No symbolic definition for this string exists.
oObjectPart	db2Char	The first 14 characters are a time stamp with format *yyyymmddhhnnss*, indicating when the operation was begun. The next 3 characters are a sequence number. Each backup operation can result in multiple entries in this file when the backup image is saved in multiple files or on multiple tapes. The sequence number allows multiple locations to be specified. Restore and load operations have only a single entry in this file, which corresponds to sequence number '001' of the corresponding backup. The time stamp, combined with the sequence number, must be unique.
oEndTime	db2Char	A time stamp with format *yyyymmddhhnnss*, indicating when the operation was completed.
oFirstLog	db2Char	The earliest log file ID (ranging from S0000000 to S9999999): • Required to apply rollforward recovery for an online backup • Required to apply rollforward recovery for an offline backup • Applied after restoring a full database or table space level backup that was current when the load started.
oLastLog	db2Char	The latest log file ID (ranging from S0000000 to S9999999): • Required to apply rollforward recovery for an online backup • Required to apply rollforward recovery to the current point in time for an offline backup • Applied after restoring a full database or table space level backup that was current when the load operation finished (will be the same as *oFirstLog* if roll forward recovery is not applied).

Table 2. Fields in the db2HistData Structure (continued)

Field Name	Data Type	Description
oID	db2Char	Unique backup or table identifier.
oTableQualifier	db2Char	Table qualifier.
oTableName	db2Char	Table name.
oLocation	db2Char	For backups and load copies, this field indicates where the data has been saved. For operations that require multiple entries in the file, the sequence number defined by *oObjectPart* identifies which part of the backup is found in the specified location. For restore and load operations, the location always identifies where the first part of the data restored or loaded (corresponding to sequence '001' for multi-part backups) has been saved. The data in *oLocation* is interpreted differently, depending on *oDeviceType*: • For disk or diskette (D or K), a fully qualified file name • For tape (T), a volume label • For TSM (A), the server name • For user exit or other (U or O), free form text.
oComment	db2Char	Free form text comment.
oCommandText	db2Char	Command text, or DDL.
oLastLSN	SQLU_LSN	Last log sequence number.
oEID	Structure	Unique entry identifier.
poEventSQLCA	Structure	Result *sqlca* of the recorded event.
poTablespace	db2Char	A list of table space names.
ioNumTablespaces	db2Uint32	Number of entries in the *poTablespace* list. Each table space backup contains one or more table spaces. Each table space restore operation replaces one or more table spaces. If this field is not zero (indicating a table space level backup or restore), the next lines in this file contain the name of the table space backed up or restored, represented by an 18-character string. One table space name appears on each line.
oOperation	char	See Table 3 on page 269.
oObject	char	Granularity of the operation: D for full database, P for table space, and T for table.
oOptype	char	See Table 4 on page 270.

Table 2. Fields in the db2HistData Structure (continued)

Field Name	Data Type	Description
oStatus	char	Entry status: A for action, D for deleted (future use), E for expired, I for inactive, N for not yet committed, Y for committed or active, a for active backup, but some datalink servers have not yet completed the backup, and i for inactive backup, but some datalink servers have not yet completed the backup.
oDeviceType	char	Device type. This field determines how the *oLocation* field is interpreted: A for TSM, C for client, D for disk, K for diskette, L for local, O for other (for other vendor device support), P for pipe, Q for cursor, S for server, T for tape, and U for user exit.

Table 3. Valid oOperation Values in the db2HistData Structure

Value	Description	C Definition	COBOL/FORTRAN Definition
A	add table space	DB2HISTORY_OP_ADD_ TABLESPACE	DB2HIST_OP_ADD_ TABLESPACE
B	backup	DB2HISTORY_OP_BACKUP	DB2HIST_OP_BACKUP
C	load copy	DB2HISTORY_OP_LOAD_COPY	DB2HIST_OP_LOAD_COPY
D	dropped table	DB2HISTORY_OP_DROPPED_ TABLE	DB2HIST_OP_DROPPED_TABLE
F	rollforward	DB2HISTORY_OP_ROLLFWD	DB2HIST_OP_ROLLFWD
G	reorganize table	DB2HISTORY_OP_REORG	DB2HIST_OP_REORG
L	load	DB2HISTORY_OP_LOAD	DB2HIST_OP_LOAD
N	rename table space	DB2HISTORY_OP_REN_ TABLESPACE	DB2HIST_OP_REN_ TABLESPACE
O	drop table space	DB2HISTORY_OP_DROP_ TABLESPACE	DB2HIST_OP_DROP_ TABLESPACE
Q	quiesce	DB2HISTORY_OP_QUIESCE	DB2HIST_OP_QUIESCE
R	restore	DB2HISTORY_OP_RESTORE	DB2HIST_OP_RESTORE
T	alter table space	DB2HISTORY_OP_ALT_ TABLESPACE	DB2HIST_OP_ALT_TBS
U	unload	DB2HISTORY_OP_UNLOAD	DB2HIST_OP_UNLOAD

Table 4. Valid oOptype Values in the db2HistData Structure

oOperation	oOptype	Description	C/COBOL/FORTRAN Definition
B	F	offline	DB2HISTORY_OPTYPE_OFFLINE
	N	online	DB2HISTORY_OPTYPE_ONLINE
	I	incremental offline	DB2HISTORY_OPTYPE_INCR_ OFFLINE
	O	incremental online	DB2HISTORY_OPTYPE_INCR_ ONLINE
	D	delta offline	DB2HISTORY_OPTYPE_DELTA_ OFFLINE
	E	delta online	DB2HISTORY_OPTYPE_DELTA_ ONLINE
F	E	end of logs	DB2HISTORY_OPTYPE_EOL
	P	point in time	DB2HISTORY_OPTYPE_PIT
G	F	offline	DB2HISTORY_OPTYPE_OFFLINE
	N	online	DB2HISTORY_OPTYPE_ONLINE
L	I	insert	DB2HISTORY_OPTYPE_INSERT
	R	replace	DB2HISTORY_OPTYPE_REPLACE
Q	S	quiesce share	DB2HISTORY_OPTYPE_SHARE
	U	quiesce update	DB2HISTORY_OPTYPE_UPDATE
	X	quiesce exclusive	DB2HISTORY_OPTYPE_EXCL
	Z	quiesce reset	DB2HISTORY_OPTYPE_RESET
R	F	offline	DB2HISTORY_OPTYPE_OFFLINE
	N	online	DB2HISTORY_OPTYPE_ONLINE
	I	incremental offline	DB2HISTORY_OPTYPE_INCR_ OFFLINE
	O	incremental online	DB2HISTORY_OPTYPE_INCR_ ONLINE
T	C	add containers	DB2HISTORY_OPTYPE_ADD_CONT
	R	rebalance	DB2HISTORY_OPTYPE_REB

Table 5. Fields in the db2Char Structure

Field Name	Data Type	Description
pioData	char	A pointer to a character data buffer. If NULL, no data will be returned.
iLength	db2Uint32	Input. The size of the *pioData* buffer.
oLength	db2Uint32	Output. The number of valid characters of data in the *pioData* buffer.

Table 6. Fields in the db2HistoryEID Structure

Field Name	Data Type	Description
ioNode	SQL_PDB_NODE_TYPE	Node number.
ioHID	db2Uint32	Local history file entry ID.

Language syntax:

C Structure

```
/* File: db2ApiDf.h */
/* ... */
typedef SQL_STRUCTURE db2HistoryData
{
  char ioHistDataID[8];
  db2Char oObjectPart;
  db2Char oEndTime;
  db2Char oFirstLog;
  db2Char oLastLog;
  db2Char oID;
  db2Char oTableQualifier;
  db2Char oTableName;
  db2Char oLocation;
  db2Char oComment;
  db2Char oCommandText;
  SQLU_LSN oLastLSN;
  db2HistoryEID oEID;
  struct sqlca * poEventSQLCA;
  db2Char * poTablespace;
  db2Uint32 ioNumTablespaces;
  char oOperation;
  char oObject;
  char oOptype;
  char oStatus;
  char oDeviceType
} db2HistoryData;

typedef SQL_STRUCTURE db2Char
{
  char * pioData;
  db2Uint32 ioLength
} db2Char;

typedef SQL_STRUCTURE db2HistoryEID
{
  SQL_PDB_NODE_TYPE ioNode;
  db2Uint32 ioHID
} db2HistoryEID;
/* ... */
```

Related reference:

- “db2HistoryGetEntry - Get Next History File Entry” on page 242
- “SQLCA” in the *Administrative API Reference*

SQLU-LSN

This union, used by the db2ReadLog API, contains the definition of the log sequence number. A log sequence number (LSN) represents a relative byte address within the database log. All log records are identified by this number. It represents the log record's byte offset from the beginning of the database log.

Table 7. Fields in the SQLU-LSN Union

Field Name	Data Type	Description
lsnChar	Array of UNSIGNED CHAR	Specifies the 6-member character array log sequence number.
lsnWord	Array of UNSIGNED SHORT	Specifies the 3-member short array log sequence number.

Language syntax:

C Structure

```
typedef union SQLU_LSN
{
unsigned char  lsnChar  [6] ;
unsigned short lsnWord  [3] ;
} SQLU_LSN;
```

Related reference:

- "db2ReadLog - Asynchronous Read Log" on page 263

Appendix E. Recovery Sample Program

Sample Program with Embedded SQL (dbrecov.sqc)

The following sample program shows how to use DB2 backup and restore APIs to:

- Back up a database
- Restore the database
- Rollforward recover the database

Note: The dbrecov sample files can be found in sqllib/samples/c and sqllib/samples/cpp directory.

```
/****************************************************************************
** Licensed Materials - Property of IBM
** Governed under the terms of the IBM Public License
**
** (C) COPYRIGHT International Business Machines Corp. 2002
** All Rights Reserved.
**
** US Government Users Restricted Rights - Use, duplication or
** disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
*****************************************************************************
**
** SOURCE FILE NAME: dbrecov.sqc
**
** SAMPLE: How to recover a database
**
** DB2 API USED:
**         db2HistoryCloseScan -- Close Recovery History File Scan
**         db2HistoryGetEntry -- Get Next Recovery History File Entry
**         db2HistoryOpenScan -- Open Recovery History File Scan
**         db2HistoryUpdate -- Update Recovery History File
**         db2Prune -- Prune Recovery History File
**         db2CfgGet -- Get Configuration
**         db2CfgSet -- Set Configuration
**         sqlbmtsq -- Tablespace Query
**         sqlbstsc -- Set Tablespace Containers
**         sqlbtcq -- Tablespace Container Query
**         sqlecrea -- Create Database
**         sqledrpd -- Drop Database
**         sqlefmem -- Free Memory
**         db2Backup -- Backup Database
**         db2Restore -- Restore Database
**         db2ReadLog -- Asynchronous Read Log
**         db2ReadLogNoConn -- No Db Connection Read Log
**         db2Rollforward -- Rollforward Database
**
** SQL STATEMENTS USED:
```

```
**        ALTER TABLE
**        COMMIT
**        DELETE
**        INSERT
**        ROLLBACK
**
** OUTPUT FILE: dbrecov.out (available in the online documentation)
**
** For detailed information about database backup and recovery, see the
** "Data Recovery and High Availability Guide and Reference". This manual
** will help you to determine which database and table space recovery methods
** are best suited to your business environment.
**
** For more information about the sample programs, see the README file.
**
** For more information about programming in C, see the
** "Programming in C and C++" section of the "Application Development Guide".
**
** For more information about building C applications, see the
** section for your compiler in the "Building Applications" chapter
** for your platform in the "Application Development Guide".
**
** For more information about SQL, see the "SQL Reference".
**
** For more information on DB2 APIs, see the Administrative API Reference.
**
** For the latest information on programming, compiling, and running DB2
** applications, refer to the DB2 application development website at
**     http://www.software.ibm.com/data/db2/udb/ad
****************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sqlenv.h>
#include <sqlutil.h>
#include <db2ApiDf.h>
#include "utilemb.h"

int DbRecoveryHistoryFilePrune(char *, char *, char *);
int DbBackupAndRestore(char *, char *, char *, char *, char *);
int DbBackupAndRedirectedRestore(char *, char *, char *, char *, char *);
int DbBackupRestoreAndRollforward(char *, char *, char *, char *, char *);
int DbLogRecordsForCurrentConnectionRead(char *, char *, char *, char *);
int DbRecoveryHistoryFileRead(char *);
int DbFirstRecoveryHistoryFileEntryUpdate(char *, char *, char *);
int DbReadLogRecordsNoConn(char *);

/* support function called by main() */
int ServerWorkingPathGet(char *, char *);

/* support function called by DbBackupAndRedirectedRestore() */
int InaccessableContainersRedefine(char *);

/* support function called by DbBackupAndRedirectedRestore() and
   DbBackupRestoreAndRollforward() */
```

```
int DbDrop(char *);

/* support function called by DbLogRecordsForCurrentConnectionRead() */
int LogBufferDisplay(char *, sqluint32);
int LogRecordDisplay(char *, sqluint32, sqluint16, sqluint16);
int SimpleLogRecordDisplay(sqluint16, sqluint16, char *, sqluint32);
int ComplexLogRecordDisplay(sqluint16, sqluint16, char *, sqluint32,
                            sqluint8, char *, sqluint32);
int LogSubRecordDisplay(char *, sqluint16);
int UserDataDisplay(char *, sqluint16);

/* support functions called by DbRecoveryHistoryFileRead() and
   DbFirstRecoveryHistoryFileEntryUpdate() */
int HistoryEntryDataFieldsAlloc(struct db2HistoryData *);
int HistoryEntryDisplay(struct db2HistoryData);
int HistoryEntryDataFieldsFree(struct db2HistoryData *);

/* DbCreate will create a new database on the server with the server's
   code page.
   Use this function only if you want to restore a remote database.
   This support function is being called by DbBackupAndRedirectedRestore()
   and DbBackupRestoreAndRollforward().  */
int DbCreate(char *, char *);

int main(int argc, char *argv[])
{
  int rc = 0;
  char nodeName[SQL_INSTNAME_SZ + 1];
  char serverWorkingPath[SQL_PATH_SZ + 1];
  char restoredDbAlias[SQL_ALIAS_SZ + 1];
  char redirectedRestoredDbAlias[SQL_ALIAS_SZ + 1];
  char rolledForwardDbAlias[SQL_ALIAS_SZ + 1];
  sqluint16 savedLogRetainValue;
  char dbAlias[SQL_ALIAS_SZ + 1];
  char user[USERID_SZ + 1];
  char pswd[PSWD_SZ + 1];

  /* check the command line arguments */
  rc = CmdLineArgsCheck3(argc, argv, dbAlias, nodeName, user, pswd);
  if (rc != 0)
  {
    return rc;
  }

  printf("\nTHIS SAMPLE SHOWS HOW TO RECOVER A DATABASE.\n");

  strcpy(restoredDbAlias, dbAlias);
  strcpy(redirectedRestoredDbAlias, "RRDB");
  strcpy(rolledForwardDbAlias, "RFDB");

  /* attach to a local or remote instance */
  rc = InstanceAttach(nodeName, user, pswd);
  if (rc != 0)
  {
    return rc;
```

```
}

printf("\nUSE THE DB2 API:\n");
printf("  db2CfgGet -- Get Configuration\n");
printf("TO GET THE DATABASE CONFIGURATION AND DETERMINE\n");
printf("THE SERVER WORKING PATH.\n");

/* get the server working path */
rc = ServerWorkingPathGet(dbAlias, serverWorkingPath);
if (rc != 0)
{
  return rc;
}

printf("\nNOTE: Backup images will be created on the server\n");
printf("      in the directory %s,\n", serverWorkingPath);
printf("      and will not be deleted by the program.\n");

/* call the sample functions */
rc = DbRecoveryHistoryFilePrune(dbAlias, user, pswd);

rc = DbBackupAndRestore(dbAlias,
                        restoredDbAlias,
                        user,
                        pswd,
                        serverWorkingPath);

rc = DbBackupAndRedirectedRestore(dbAlias,
                                  redirectedRestoredDbAlias,
                                  user,
                                  pswd,
                                  serverWorkingPath);

rc = DbBackupRestoreAndRollforward(dbAlias,
                                   rolledForwardDbAlias,
                                   user,
                                   pswd,
                                   serverWorkingPath);

rc = DbLogRecordsForCurrentConnectionRead(dbAlias,
                                          user,
                                          pswd,
                                          serverWorkingPath);

rc = DbRecoveryHistoryFileRead(dbAlias);

rc = DbFirstRecoveryHistoryFileEntryUpdate(dbAlias, user, pswd);

rc = DbRecoveryHistoryFilePrune(dbAlias, user, pswd);

/* detach from the local or remote instance */
rc = InstanceDetach(nodeName);
if (rc != 0)
{
  return rc;
  }

  rc = DbReadLogRecordsNoConn(dbAlias);

  return 0;
} /* end main */

int ServerWorkingPathGet(char dbAlias[], char serverWorkingPath[])
{
  int rc = 0;
  struct sqlca sqlca;
  char serverLogPath[SQL_PATH_SZ + 1];
  db2CfgParam cfgParameters[1];
  db2Cfg cfgStruct;
  int len;

  /* initialize cfgParameters */
  /* SQLF_DBTN_LOGPATH is a token of the non-updatable database configuration
     parameter 'logpath'; it is used to get the server log path */
  cfgParameters[0].flags = 0;
  cfgParameters[0].token = SQLF_DBTN_LOGPATH;
  cfgParameters[0].ptrvalue =
    (char *)malloc((SQL_PATH_SZ + 1) * sizeof(char));

  /* initialize cfgStruct */
  cfgStruct.numItems = 1;
  cfgStruct.paramArray = cfgParameters;
  cfgStruct.flags = db2CfgDatabase;
  cfgStruct.dbname = dbAlias;

  /* get database configuration */
  /* the API db2CfgGet returns the values of individual entries in a
     database configuration file */
  db2CfgGet(db2Version810, (void *)&cfgStruct, &sqlca);
  DB2_API_CHECK("server log path -- get");

  strcpy(serverLogPath, cfgParameters[0].ptrvalue);
  free(cfgParameters[0].ptrvalue);

  /* choose the server working path; if, for example, serverLogPath =
     "C:\DB2\NODE0001\....", we'll keep "C:\DB2" for the serverWorkingPath
     variable; backup images created in this sample will be placed under
     the 'serverWorkingPath' directory */
  len = (int)(strstr(serverLogPath, "NODE") - serverLogPath - 1);
  memcpy(serverWorkingPath, serverLogPath, len);
  serverWorkingPath[len] = '\0';

  return 0;
} /* ServerWorkingPathGet */

int DbCreate(char existingDbAlias[], char newDbAlias[])
{
  struct sqlca sqlca;
  char dbName[SQL_DBNAME_SZ + 1];
  char dbLocalAlias[SQL_ALIAS_SZ + 1];
char dbPath[SQL_PATH_SZ + 1];
struct sqledbdesc dbDescriptor;
struct sqledbcountryinfo countryInfo;
db2CfgParam cfgParameters[2];
db2Cfg cfgStruct;

printf("\n  Create '%s' empty database with the same code set as '%s'
       database.\n", newDbAlias, existingDbAlias);

/* initialize cfgParameters */
cfgParameters[0].flags = 0;
cfgParameters[0].token = SQLF_DBTN_TERRITORY;
cfgParameters[0].ptrvalue = (char *)malloc(10 * sizeof(char));
memset(cfgParameters[0].ptrvalue, '\0', 10);
cfgParameters[1].flags = 0;
cfgParameters[1].token = SQLF_DBTN_CODESET;
cfgParameters[1].ptrvalue = (char *)malloc(20 * sizeof(char));
memset(cfgParameters[1].ptrvalue, '\0', 20);

/* initialize cfgStruct */
cfgStruct.numItems = 2;
cfgStruct.paramArray = cfgParameters;
cfgStruct.flags = db2CfgDatabase;
cfgStruct.dbname = existingDbAlias;

/* get database configuration */
db2CfgGet(db2Version810, (void *)&cfgStruct, &sqlca);
DB2_API_CHECK("server log path -- get");

/* create a new database */
strcpy(dbName, newDbAlias);
strcpy(dbLocalAlias, newDbAlias);
strcpy(dbPath, "");

strcpy(dbDescriptor.sqldbdid, SQLE_DBDESC_2);
dbDescriptor.sqldbccp = 0;
dbDescriptor.sqldbcss = SQL_CS_NONE;

strcpy(dbDescriptor.sqldbcmt, "");
dbDescriptor.sqldbsgp = 0;
dbDescriptor.sqldbnsg = 10;
dbDescriptor.sqltsext = -1;
dbDescriptor.sqlcatts = NULL;
dbDescriptor.sqlusrts = NULL;
dbDescriptor.sqltmpts = NULL;

strcpy(countryInfo.sqldbcodeset, (char *)cfgParameters[0].ptrvalue);
strcpy(countryInfo.sqldblocale, (char *)cfgParameters[1].ptrvalue);

/* create database */
sqlecrea(dbName,
         dbLocalAlias,
         dbPath,
         &dbDescriptor,
         &countryInfo,
           '\0',
           NULL,
           &sqlca);
  DB2_API_CHECK("Database -- Create");

  /* free the allocated memory */
  free(cfgParameters[0].ptrvalue);
  free(cfgParameters[1].ptrvalue);

  return 0;
} /* DbCreate */

int DbDrop(char dbAlias[])
{
  struct sqlca sqlca;

  printf("\n  Drop the '%s' database.\n", dbAlias);

  /* drop and uncatalog the database */
  sqledrpd(dbAlias, &sqlca);
  DB2_API_CHECK("Database -- Drop");

  return 0;
} /* DbDrop */

int DbBackupAndRestore(char dbAlias[],
                       char restoredDbAlias[],
                       char user[],
                       char pswd[],
                       char serverWorkingPath[])
{
  int rc = 0;
  struct sqlca sqlca;
  db2CfgParam cfgParameters[1];
  db2Cfg cfgStruct;
  unsigned short logretain;
  char restoreTimestamp[SQLU_TIME_STAMP_LEN + 1];

  db2BackupStruct  backupStruct;
  db2TablespaceStruct   tablespaceStruct;
  db2MediaListStruct mediaListStruct;
  db2Uint32  backupImageSize;
  db2RestoreStruct restoreStruct;
  db2TablespaceStruct   rtablespaceStruct;
  db2MediaListStruct rmediaListStruct;

  printf("\n**************************************\n");
  printf("*** BACK UP AND RESTORE A DATABASE ***\n");
  printf("**************************************\n");
  printf("\nUSE THE DB2 APIs:\n");
  printf("  db2CfgSet -- Set Configuration\n");
  printf("  db2Backup -- Backup Database\n");
  printf("  db2Restore -- Restore Database\n");
  printf("TO BACK UP AND RESTORE A DATABASE.\n");
printf("\n  Update \'%s\' database configuration:\n", dbAlias);
printf("      - Disable the database configuration parameter LOGRETAIN\n");
printf("        i.e., set LOGRETAIN = OFF/NO\n");

/* initialize cfgParameters */
/* SQLF_DBTN_LOG_RETAIN is a token of the updatable database configuration
   parameter 'logretain'; it is used to update the database configuration
   file */
cfgParameters[0].flags = 0;
cfgParameters[0].token = SQLF_DBTN_LOG_RETAIN;
cfgParameters[0].ptrvalue = (char *)&logretain;

/* disable the database configuration parameter 'logretain' */
logretain = SQLF_LOGRETAIN_DISABLE;

/* initialize cfgStruct */
cfgStruct.numItems = 1;
cfgStruct.paramArray = cfgParameters;
cfgStruct.flags = db2CfgDatabase | db2CfgDelayed;
cfgStruct.dbname = dbAlias;

/* set database configuration */
db2CfgSet(db2Version810, (void *)&cfgStruct, &sqlca);
DB2_API_CHECK("Db Log Retain -- Disable");

/*******************************/
/*    BACK UP THE DATABASE    */
/*******************************/
printf("\n  Backing up the '%s' database...\n", dbAlias);

tablespaceStruct.tablespaces = NULL;
tablespaceStruct.numTablespaces = 0;

mediaListStruct.locations = &serverWorkingPath;
mediaListStruct.numLocations = 1;
mediaListStruct.locationType = SQLU_LOCAL_MEDIA;

backupStruct.piDBAlias = dbAlias;
backupStruct.piTablespaceList = &tablespaceStruct;
backupStruct.piMediaList = &mediaListStruct;
backupStruct.piUsername = user;
backupStruct.piPassword = pswd;
backupStruct.piVendorOptions = NULL;
backupStruct.iVendorOptionsSize = 0;
backupStruct.iCallerAction = DB2BACKUP_BACKUP;
backupStruct.iBufferSize = 16; /*  16 x 4KB */
backupStruct.iNumBuffers = 1;
backupStruct.iParallelism =1;
backupStruct.iOptions = DB2BACKUP_OFFLINE | DB2BACKUP_DB;

/* The API db2Backup creates a backup copy of a database.
   This API automatically establishes a connection to the specified database.
   (This API can also be used to create a backup copy of a table space). */
db2Backup (db2Version810, &backupStruct, &sqlca);
```

```
DB2_API_CHECK("Database -- Backup");

while (sqlca.sqlcode != 0)
{
  /* continue the backup operation */

  /* depending on the sqlca.sqlcode value, user action may be */
  /* required, such as mounting a new tape */

  printf("\n  Continuing the backup operation...\n");

  backupStruct.iCallerAction = DB2BACKUP_CONTINUE;

  db2Backup (db2Version810, &backupStruct, &sqlca);

  DB2_API_CHECK("Database -- Backup");
}

printf("  Backup finished.\n");
printf("    - backup image size      : %d MB\n", backupStruct.oBackupSize);
printf("    - backup image path      : %s\n", mediaListStruct.locations[0]);

printf("    - backup image time stamp: %s\n", backupStruct.oTimestamp);

/******************************/
/*    RESTORE THE DATABASE    */
/******************************/

strcpy(restoreTimestamp, backupStruct.oTimestamp);


printf("\n  Restoring a database ...\n");
printf("    - source image alias     : %s\n", dbAlias);
printf("    - source image time stamp: %s\n", restoreTimestamp);
printf("    - target database        : %s\n", restoredDbAlias);

rtablespaceStruct.tablespaces = NULL;
rtablespaceStruct.numTablespaces = 0;

rmediaListStruct.locations = &serverWorkingPath;
rmediaListStruct.numLocations = 1;
rmediaListStruct.locationType = SQLU_LOCAL_MEDIA;

restoreStruct.piSourceDBAlias = dbAlias;
restoreStruct.piTargetDBAlias = restoredDbAlias;

restoreStruct.piTimestamp = restoreTimestamp;
restoreStruct.piTargetDBPath = NULL;
restoreStruct.piReportFile = NULL;
restoreStruct.piTablespaceList = &rtablespaceStruct;
restoreStruct.piMediaList = &rmediaListStruct;
restoreStruct.piUsername = user;
restoreStruct.piPassword = pswd;
restoreStruct.piNewLogPath = NULL;
restoreStruct.piVendorOptions = NULL;
```

```
  restoreStruct.iVendorOptionsSize = 0;
  restoreStruct.iParallelism = 1;
  restoreStruct.iBufferSize = 1024; /*  1024 x 4KB */;
  restoreStruct.iNumBuffers = 1;
  restoreStruct.iCallerAction = DB2RESTORE_RESTORE;
  restoreStruct.iOptions = DB2RESTORE_OFFLINE | DB2RESTORE_DB |
  DB2RESTORE_NODATALINK | DB2RESTORE_NOROLLFWD;

  /* The API db2Restore is used to restore a database that has been backed
     up using the API db2Backup. */
  db2Restore (db2Version810, &restoreStruct, &sqlca);

  EXPECTED_WARN_CHECK("database restore -- start");

  while (sqlca.sqlcode != 0)
  {
    /* continue the restore operation */
    printf("\n  Continuing the restore operation...\n");

    /* depending on the sqlca.sqlcode value, user action may be
       required, such as mounting a new tape */

    restoreStruct.iCallerAction = DB2RESTORE_CONTINUE;

    /* restore the database */
    db2Restore (db2Version810, &restoreStruct, &sqlca);

    DB2_API_CHECK("database restore -- continue");
  }

  printf("\n  Restore finished.\n");

  return 0;
} /* DbBackupAndRestore */

int DbBackupAndRedirectedRestore(char dbAlias[],
                                 char restoredDbAlias[],
                                 char user[],
                                 char pswd[],
                                 char serverWorkingPath[])
{
  int rc = 0;
  struct sqlca sqlca;
  db2CfgParam cfgParameters[1];
  db2Cfg cfgStruct;
  unsigned short logretain;

  char restoreTimestamp[SQLU_TIME_STAMP_LEN + 1];

  db2BackupStruct  backupStruct;
  db2TablespaceStruct   tablespaceStruct;
  db2MediaListStruct mediaListStruct;
  db2Uint32  backupImageSize;
  db2RestoreStruct restoreStruct;
  db2TablespaceStruct   rtablespaceStruct;
```

```
db2MediaListStruct rmediaListStruct;

printf("\n*************************\n");
printf("*** REDIRECTED RESTORE ***\n");
printf("*************************\n");
printf("\nUSE THE DB2 APIs:\n");
printf("  db2CfgSet -- Upate Configuration\n");
printf("  db2Backup -- Backup Database\n");
printf("  sqlecrea -- Create Database\n");
printf("  db2Restore -- Restore Database\n");
printf("  sqlbmtsq -- Tablespace Query\n");
printf("  sqlbtcq -- Tablespace Container Query\n");
printf("  sqlbstsc -- Set Tablespace Containers\n");
printf("  sqlefmem -- Free Memory\n");
printf("  sqledrpd -- Drop Database\n");
printf("TO BACK UP AND DO A REDIRECTED RESTORE OF A DATABASE.\n");

printf("\n  Update \'%s\' database configuration:\n", dbAlias);
printf("    - Disable the database configuration parameter LOGRETAIN \n");
printf("        i.e., set LOGRETAIN = OFF/NO\n");

/* initialize cfgParameters */
/* SQLF_DBTN_LOG_RETAIN is a token of the updatable database configuration
   parameter 'logretain'; it is used to update the database configuration
   file */
cfgParameters[0].flags = 0;
cfgParameters[0].token = SQLF_DBTN_LOG_RETAIN;
cfgParameters[0].ptrvalue = (char *)&logretain;

/* disable the database configuration parameter 'logretain' */
logretain = SQLF_LOGRETAIN_DISABLE;

/* initialize cfgStruct */
cfgStruct.numItems = 1;
cfgStruct.paramArray = cfgParameters;
cfgStruct.flags = db2CfgDatabase | db2CfgDelayed;
cfgStruct.dbname = dbAlias;

/* get database configuration */
db2CfgSet(db2Version810, (void *)&cfgStruct, &sqlca);
DB2_API_CHECK("Db Log Retain -- Disable");

/*******************************/
/*    BACK UP THE DATABASE    */
/*******************************/
printf("\n  Backing up the '%s' database...\n", dbAlias);

tablespaceStruct.tablespaces = NULL;
tablespaceStruct.numTablespaces = 0;

mediaListStruct.locations = &serverWorkingPath;
mediaListStruct.numLocations = 1;
mediaListStruct.locationType = SQLU_LOCAL_MEDIA;

backupStruct.piDBAlias = dbAlias;
```

```
  backupStruct.piTablespaceList = &tablespaceStruct;
  backupStruct.piMediaList = &mediaListStruct;
  backupStruct.piUsername = user;
  backupStruct.piPassword = pswd;
  backupStruct.piVendorOptions = NULL;
  backupStruct.iVendorOptionsSize = 0;
  backupStruct.iCallerAction = DB2BACKUP_BACKUP;
  backupStruct.iBufferSize = 16; /*  16 x 4KB */
  backupStruct.iNumBuffers = 1;
  backupStruct.iParallelism =1;
  backupStruct.iOptions = DB2BACKUP_OFFLINE | DB2BACKUP_DB;

  /* The API db2Backup creates a backup copy of a database.
     This API automatically establishes a connection to the
     specified database,
     (This API can also be used to create a backup copy of a table
     space). */
  db2Backup (db2Version810, &backupStruct, &sqlca);

  DB2_API_CHECK("Database -- Backup");

  while (sqlca.sqlcode != 0)
  {
    /* continue the backup operation */

    /* depending on the sqlca.sqlcode value, user action may be
       required, such as mounting a new tape */

    printf("\n  Continuing the backup operation...\n");

    backupStruct.iCallerAction = DB2BACKUP_CONTINUE;

    /* back up the database */
    db2Backup (db2Version810, &backupStruct, &sqlca);
  }

  printf("  Backup finished.\n");
  printf("    - backup image size      : %d MB\n",
         backupStruct.oBackupSize);
  printf("    - backup image path      : %s\n",
         mediaListStruct.locations[0]);

  printf("    - backup image time stamp: %s\n", backupStruct.oTimestamp);

  /* To restore a remote database, you will first need to create an
     empty database if the client's code page is different from the
     server's code page.
     If this is the case, uncomment the call to DbCreate(). It will
     create an empty database on the server with the server's
     code page. */

/*
  rc = DbCreate(dbAlias, restoredDbAlias);
  if (rc != 0)
  {
```

```
  return rc;
  }
*/

  /*****************************/
  /*    RESTORE THE DATABASE    */
  /*****************************/

  strcpy(restoreTimestamp, backupStruct.oTimestamp);

  rtablespaceStruct.tablespaces = NULL;
  rtablespaceStruct.numTablespaces = 0;

  rmediaListStruct.locations = &serverWorkingPath;
  rmediaListStruct.numLocations = 1;
  rmediaListStruct.locationType = SQLU_LOCAL_MEDIA;

  restoreStruct.piSourceDBAlias = dbAlias;
  restoreStruct.piTargetDBAlias = restoredDbAlias;
  restoreStruct.piTimestamp = restoreTimestamp;
  restoreStruct.piTargetDBPath = NULL;
  restoreStruct.piReportFile = NULL;
  restoreStruct.piTablespaceList = &rtablespaceStruct;
  restoreStruct.piMediaList = &rmediaListStruct;
  restoreStruct.piUsername = user;
  restoreStruct.piPassword = pswd;
  restoreStruct.piNewLogPath = NULL;
  restoreStruct.piVendorOptions = NULL;
  restoreStruct.iVendorOptionsSize = 0;
  restoreStruct.iParallelism = 1;
  restoreStruct.iBufferSize = 1024; /*  1024 x 4KB */;
  restoreStruct.iNumBuffers = 1;
  restoreStruct.iOptions = DB2RESTORE_OFFLINE | DB2RESTORE_DB |
  DB2RESTORE_NODATALINK | DB2RESTORE_NOROLLFWD;

  printf("\n  Restoring a database ...\n");
  printf("    - source image alias     : %s\n", dbAlias);
  printf("    - source image time stamp: %s\n", restoreTimestamp);
  printf("    - target database        : %s\n", restoredDbAlias);

  restoreStruct.iCallerAction = DB2RESTORE_RESTORE_STORDEF;

  /* The API db2Restore is used to restore a database that has been backed
     up using the API db2Backup. */
  db2Restore(db2Version810, &restoreStruct, &sqlca);

  EXPECTED_WARN_CHECK("database restore -- start");

  while (sqlca.sqlcode != 0)
  {
    /* continue the restore operation */
    printf("\n  Continuing the restore operation...\n");

    /* depending on the sqlca.sqlcode value, user action may be
       required, such as mounting a new tape */
    if (sqlca.sqlcode == SQLUD_INACCESSABLE_CONTAINER)
    {
      /* redefine the table space container layout */
      printf("\n  Find and redefine inaccessable containers.\n");
      rc = InaccessableContainersRedefine(serverWorkingPath);
      if (rc != 0)
      {
        return rc;
      }
    }

    restoreStruct.iCallerAction = DB2RESTORE_CONTINUE;

    /* restore the database */
    db2Restore (db2Version810, &restoreStruct, &sqlca);

    DB2_API_CHECK("database restore -- continue");
  }

  printf("\n  Restore finished.\n");

  /* drop the restored database */
  rc = DbDrop(restoredDbAlias);

  return 0;
} /* DbBackupAndRedirectedRestore */

int InaccessableContainersRedefine(char serverWorkingPath[])
{
  int rc = 0;
  struct sqlca sqlca;
  sqluint32 numTablespaces;
  struct SQLB_TBSPQRY_DATA **ppTablespaces;
  sqluint32 numContainers;
  struct SQLB_TBSCONTQRY_DATA *pContainers;
  int tspNb;
  int contNb;
  char pathSep[2];

  /* The API sqlbmtsq provides a one-call interface to the table space query
     data. The query data for all table spaces in the database is returned
     in an array. */
  sqlbmtsq(&sqlca,
           &numTablespaces,
           &ppTablespaces,
           SQLB_RESERVED1,
           SQLB_RESERVED2);
  DB2_API_CHECK("tablespaces -- get");

  /* refedine the inaccessable containers */
  for (tspNb = 0; tspNb < numTablespaces; tspNb++)
  {
    /* The API sqlbtcq provides a one-call interface to the table space
       container query data. The query data for all the containers in a table
```

```
   space, or for all containers in all table spaces, is returned in an
   array. */
sqlbtcq(&sqlca, ppTablespaces[tspNb]->id, &numContainers, &pContainers);
DB2_API_CHECK("tablespace containers -- get");

for (contNb = 0; contNb < numContainers; contNb++)
{
  if (!pContainers[contNb].ok)
  {
    /* redefine inaccessable container */
    printf("\n    Redefine inaccessable container:\n");
    printf("      - table space name: %s\n",
           ppTablespaces[tspNb]->name);
    printf("      - default container name: %s\n",
           pContainers[contNb].name);
    if (strstr(pContainers[contNb].name, "/"))
    { /* UNIX */
      strcpy(pathSep, "/");
    }
    else
    { /* Intel */
      strcpy(pathSep, "\\");
    }
    switch (pContainers[contNb].contType)
    {
      case SQLB_CONT_PATH:
        printf("      - container type: path\n");

        sprintf(pContainers[contNb].name, "%s%sSQLT%04d.%d",
                                          serverWorkingPath, pathSep,
                                          ppTablespaces[tspNb]->id,
                                          pContainers[contNb].id);
        printf("      - new container name: %s\n",
               pContainers[contNb].name);
        break;
      case SQLB_CONT_DISK:
      case SQLB_CONT_FILE:
      default:
        printf("    Unknown container type.\n");
        break;
    }
  }
}

/* The API sqlbstsc is used to set or redefine table space containers
   while performing a 'redirected' restore of the database. */
sqlbstsc(&sqlca,
         SQLB_SET_CONT_FINAL_STATE,
         ppTablespaces[tspNb]->id,
         numContainers,
         pContainers);
DB2_API_CHECK("tablespace containers -- redefine");

/* The API sqlefmem is used here to free memory allocated by DB2 for use
   with the API sqlbtcq (Tablespace Container Query). */
```

```
    sqlefmem(&sqlca, pContainers);
    DB2_API_CHECK("tablespace containers memory -- free");
  }

  /* The API sqlefmem is used here to free memory allocated by DB2 for
     use with the API sqlbmtsq (Tablespace Query). */
  sqlefmem(&sqlca, ppTablespaces);
  DB2_API_CHECK("tablespaces memory -- free");

  return 0;
} /* InaccessableContainersRedefine */

int DbBackupRestoreAndRollforward(char dbAlias[],
                                  char rolledForwardDbAlias[],
                                  char user[],
                                  char pswd[],
                                  char serverWorkingPath[])
{
  int rc = 0;
  struct sqlca sqlca;
  db2CfgParam cfgParameters[1];
  db2Cfg cfgStruct;
  unsigned short logretain;

  char restoreTimestamp[SQLU_TIME_STAMP_LEN + 1];

  db2BackupStruct  backupStruct;
  db2TablespaceStruct   tablespaceStruct;
  db2MediaListStruct mediaListStruct;
  db2Uint32  backupImageSize;
  db2RestoreStruct restoreStruct;
  db2TablespaceStruct   rtablespaceStruct;
  db2MediaListStruct rmediaListStruct;

  db2RfwdInputStruct   rfwdInput;
  db2RfwdOutputStruct  rfwdOutput;
  db2RollforwardStruct  rfwdStruct;

  char rollforwardAppId[SQLU_APPLID_LEN + 1];
  sqlint32 numReplies;
  struct sqlurf_info nodeInfo;

  printf("\n****************************\n");
  printf("*** ROLLFORWARD RECOVERY ***\n");
  printf("****************************\n");
  printf("\nUSE THE DB2 APIs:\n");
  printf("  db2CfgSet -- Set Configuration\n");
  printf("  db2Backup -- Backup Database\n");
  printf("  sqlecrea -- Create Database\n");
  printf("  db2Restore -- Restore Database\n");
  printf("  db2Rollforward -- Rollforward Database\n");
  printf("  sqledrpd -- Drop Database\n");
  printf("TO BACK UP, RESTORE, AND ROLL A DATABASE FORWARD. \n");

  printf("\n  Update \'%s\' database configuration:\n", dbAlias);
printf("    - Enable the configuration parameter LOGRETAIN \n");
printf("        i.e., set LOGRETAIN = RECOVERY/YES\n");

/* initialize cfgParameters */
cfgParameters[0].flags = 0;
cfgParameters[0].token = SQLF_DBTN_LOG_RETAIN;
cfgParameters[0].ptrvalue = (char *)&logretain;

/* enable the configuration parameter 'logretain' */
logretain = SQLF_LOGRETAIN_RECOVERY;

/* initialize cfgStruct */
cfgStruct.numItems = 1;
cfgStruct.paramArray = cfgParameters;
cfgStruct.flags = db2CfgDatabase | db2CfgDelayed;
cfgStruct.dbname = dbAlias;

/* get database configuration */
db2CfgSet(db2Version810, (void *)&cfgStruct, &sqlca);
DB2_API_CHECK("Db Log Retain -- Enable");

/* start the backup operation */
printf("\n  Backing up the '%s' database...\n", dbAlias);

tablespaceStruct.tablespaces = NULL;
tablespaceStruct.numTablespaces = 0;

mediaListStruct.locations = &serverWorkingPath;
mediaListStruct.numLocations = 1;
mediaListStruct.locationType = SQLU_LOCAL_MEDIA;

backupStruct.piDBAlias = dbAlias;
backupStruct.piTablespaceList = &tablespaceStruct;
backupStruct.piMediaList = &mediaListStruct;
backupStruct.piUsername = user;
backupStruct.piPassword = pswd;
backupStruct.piVendorOptions = NULL;
backupStruct.iVendorOptionsSize = 0;
backupStruct.iCallerAction = DB2BACKUP_BACKUP;
backupStruct.iBufferSize = 16; /*  16 x 4KB */
backupStruct.iNumBuffers = 1;
backupStruct.iParallelism =1;
backupStruct.iOptions = DB2BACKUP_OFFLINE | DB2BACKUP_DB;

/* The API db2Backup creates a backup copy of a database.
   This API automatically establishes a connection to the specified database.
   (This API can also be used to create a backup copy of a table space). */
db2Backup (db2Version810, &backupStruct, &sqlca);

DB2_API_CHECK("Database -- Backup");

while (sqlca.sqlcode != 0)
{
  /* continue the backup operation */
  printf("\n  Continuing the backup operation...\n");
```

```
    /* depending on the sqlca.sqlcode value, user action may be
       required, such as mounting a new tape. */

    backupStruct.iCallerAction = DB2BACKUP_CONTINUE;

    /* back up the database */
    db2Backup (db2Version810, &backupStruct, &sqlca);

    DB2_API_CHECK("Database -- Backup");
  }

  printf("  Backup finished.\n");
  printf("    - backup image size      : %d MB\n",
         backupStruct.oBackupSize);
  printf("    - backup image path      : %s\n",
         mediaListStruct.locations[0]);

  printf("    - backup image time stamp: %s\n", backupStruct.oTimestamp);

  /* To restore a remote database, you will first need to create an
empty database
     if the client's code page is different from the server's code page.
     If this is the case, uncomment the call to DbCreate(). It will create
     an empty database on the server with the server's code page. */

/*
  rc = DbCreate(dbAlias, rolledForwardDbAlias);
  if (rc != 0)
  {
    return rc;
  }
*/

  /******************************/
  /*    RESTORE THE DATABASE    */
  /******************************/

  strcpy(restoreTimestamp, backupStruct.oTimestamp);

  rtablespaceStruct.tablespaces = NULL;
  rtablespaceStruct.numTablespaces = 0;

  rmediaListStruct.locations = &serverWorkingPath;
  rmediaListStruct.numLocations = 1;
  rmediaListStruct.locationType = SQLU_LOCAL_MEDIA;

  restoreStruct.piSourceDBAlias = dbAlias;
  restoreStruct.piTargetDBAlias = rolledForwardDbAlias;
  restoreStruct.piTimestamp = restoreTimestamp;
  restoreStruct.piTargetDBPath = NULL;
  restoreStruct.piReportFile = NULL;
  restoreStruct.piTablespaceList = &rtablespaceStruct;
  restoreStruct.piMediaList = &rmediaListStruct;
  restoreStruct.piUsername = user;
restoreStruct.piPassword = pswd;
restoreStruct.piNewLogPath = NULL;
restoreStruct.piVendorOptions = NULL;
restoreStruct.iVendorOptionsSize = 0;
restoreStruct.iParallelism = 1;
restoreStruct.iBufferSize = 1024; /*  1024 x 4KB */;
restoreStruct.iNumBuffers = 1;
restoreStruct.iCallerAction = DB2RESTORE_RESTORE;
restoreStruct.iOptions = DB2RESTORE_OFFLINE | DB2RESTORE_DB |
DB2RESTORE_NODATALINK | DB2RESTORE_ROLLFWD;

printf("\n  Restoring a database ...\n");
printf("    - source image alias     : %s\n", dbAlias);
printf("    - source image time stamp: %s\n", restoreTimestamp);
printf("    - target database        : %s\n", rolledForwardDbAlias);

/* The API db2Restore is used to restore a database that has been backed
   up using the API db2Backup. */
db2Restore (db2Version810, &restoreStruct, &sqlca);

DB2_API_CHECK("database restore -- start");

while (sqlca.sqlcode != 0)
{
  /* continue the restore operation */
  printf("\n  Continuing the restore operation...\n");

  /* Depending on the sqlca.sqlcode value, user action may be
     required, such as mounting a new tape. */

  restoreStruct.iCallerAction = DB2RESTORE_CONTINUE;

  /* restore the database */
  db2Restore (db2Version810, &restoreStruct, &sqlca);

  DB2_API_CHECK("database restore -- continue");
}

printf("\n  Restore finished.\n");


/******************************/
/*    ROLLFORWARD RECOVERY    */
/******************************/

printf("\n  Rolling '%s' database forward ...\n", rolledForwardDbAlias);

rfwdInput.version = SQLUM_RFWD_VERSION;
rfwdInput.pDbAlias = rolledForwardDbAlias;
rfwdInput.CallerAction = SQLUM_ROLLFWD_STOP;
rfwdInput.pStopTime = SQLUM_INFINITY_TIMESTAMP;
rfwdInput.pUserName = user;
rfwdInput.pPassword = pswd;
rfwdInput.pOverflowLogPath = serverWorkingPath;
rfwdInput.NumChngLgOvrflw = 0;
```

```
  rfwdInput.pChngLogOvrflw = NULL;
  rfwdInput.ConnectMode = SQLUM_OFFLINE;
  rfwdInput.pTablespaceList = NULL;
  rfwdInput.AllNodeFlag = SQLURF_ALL_NODES;
  rfwdInput.NumNodes = 0;
  rfwdInput.pNodeList = NULL;
  rfwdInput.pDroppedTblID = NULL;
  rfwdInput.pExportDir = NULL;
  rfwdInput.NumNodeInfo = 1;
  rfwdInput.RollforwardFlags = 0;

  rfwdOutput.pApplicationId = rollforwardAppId;
  rfwdOutput.pNumReplies = &numReplies;
  rfwdOutput.pNodeInfo = &nodeInfo;

  rfwdStruct.roll_input = &rfwdInput;
  rfwdStruct.roll_output = &rfwdOutput;

  /* rollforward database */
  /* The API db2Rollforward rollforward recovers a database by
     applying transactions recorded in the database log files. */
  db2Rollforward(db2Version810, &rfwdStruct, &sqlca);

  DB2_API_CHECK("rollforward -- start");

  printf("  Rollforward finished.\n");

  /* drop the restored database */
  rc = DbDrop(rolledForwardDbAlias);

  return 0;
} /* DbBackupRestoreAndRollforward */

int DbLogRecordsForCurrentConnectionRead(char dbAlias[],
                                         char user[],
                                         char pswd[],
                                         char serverWorkingPath[])
{
  int rc = 0;
  struct sqlca sqlca;
  db2CfgParam cfgParameters[1];
  db2Cfg cfgStruct;
  unsigned short logretain;

  db2BackupStruct  backupStruct;
  db2TablespaceStruct   tablespaceStruct;
  db2MediaListStruct mediaListStruct;
  db2Uint32  backupImageSize;
  db2RestoreStruct restoreStruct;
  db2TablespaceStruct   rtablespaceStruct;
  db2MediaListStruct rmediaListStruct;

  SQLU_LSN startLSN;
  SQLU_LSN endLSN;
  char *logBuffer;
```

```
sqluint32 logBufferSize;
db2ReadLogInfoStruct readLogInfo;
db2ReadLogStruct     readLogInput;
int i;

printf("\n*****************************\n");
printf("*** ASYNCHRONOUS READ LOG ***\n");
printf("*****************************\n");
printf("\nUSE THE DB2 APIs:\n");
printf("  db2CfgSet -- Set Configuration\n");
printf("  db2Backup -- Backup Database\n");
printf("  db2ReadLog -- Asynchronous Read Log\n");
printf("AND THE SQL STATEMENTS:\n");
printf("  CONNECT\n");
printf("  ALTER TABLE\n");
printf("  COMMIT\n");
printf("  INSERT\n");
printf("  DELETE\n");
printf("  ROLLBACK\n");
printf("  CONNECT RESET\n");
printf("TO READ LOG RECORDS FOR THE CURRENT CONNECTION.\n");

printf("\n  Update \'%s\' database configuration:\n", dbAlias);
printf("    - Enable the database configuration parameter LOGRETAIN \n");
printf("        i.e., set LOGRETAIN = RECOVERY/YES\n");

/* initialize cfgParameters */
cfgParameters[0].flags = 0;
cfgParameters[0].token = SQLF_DBTN_LOG_RETAIN;
cfgParameters[0].ptrvalue = (char *)&logretain;

/* enable LOGRETAIN */
logretain = SQLF_LOGRETAIN_RECOVERY;

/* initialize cfgStruct */
cfgStruct.numItems = 1;
cfgStruct.paramArray = cfgParameters;
cfgStruct.flags = db2CfgDatabase | db2CfgDelayed;
cfgStruct.dbname = dbAlias;

/* get database configuration */
db2CfgSet(db2Version810, (void *)&cfgStruct, &sqlca);
DB2_API_CHECK("Db Log Retain -- Enable");

/* start the backup operation */
printf("\n  Backing up the '%s' database...\n", dbAlias);

tablespaceStruct.tablespaces = NULL;
tablespaceStruct.numTablespaces = 0;

mediaListStruct.locations = &serverWorkingPath;
mediaListStruct.numLocations = 1;
mediaListStruct.locationType = SQLU_LOCAL_MEDIA;

backupStruct.piDBAlias = dbAlias;
```

```
  backupStruct.piTablespaceList = &tablespaceStruct;
  backupStruct.piMediaList = &mediaListStruct;
  backupStruct.piUsername = user;
  backupStruct.piPassword = pswd;
  backupStruct.piVendorOptions = NULL;
  backupStruct.iVendorOptionsSize = 0;
  backupStruct.iCallerAction = DB2BACKUP_BACKUP;
  backupStruct.iBufferSize = 16; /*  16 x 4KB */
  backupStruct.iNumBuffers = 1;
  backupStruct.iParallelism =1;
  backupStruct.iOptions = DB2BACKUP_OFFLINE | DB2BACKUP_DB;

  /* The API db2Backup creates a backup copy of a database.
     This API automatically establishes a connection to the specified database.
     (This API can also be used to create a backup copy of a table space). */
  db2Backup (db2Version810, &backupStruct, &sqlca);

  DB2_API_CHECK("Database -- Backup");

  while (sqlca.sqlcode != 0)
  {
    /* continue the backup operation */
    printf("\n  Continuing the backup operation...\n");

    /* Depending on the sqlca.sqlcode value, user action may be
       required, such as mounting a new tape. */

    backupStruct.iCallerAction = DB2BACKUP_CONTINUE;

    /* back up the database */
    db2Backup (db2Version810, &backupStruct, &sqlca);

    DB2_API_CHECK("Database -- Backup");
  }

  printf("  Backup finished.\n");
  printf("    - backup image size      : %d MB\n", backupStruct.oBackupSize);
  printf("    - backup image path      : %s\n", mediaListStruct.locations[0]);

  printf("    - backup image time stamp: %s\n", backupStruct.oTimestamp);

  /* connect to the database */
  rc = DbConn(dbAlias, user, pswd);
  if (rc != 0)
  {
    return rc;
  }

  /* invoke SQL statements to fill database log */
  printf("\n  Invoke the following SQL statements:\n"
         "    ALTER TABLE emp_resume DATA CAPTURE CHANGES;\n"
         "    COMMIT;\n"
         "    INSERT INTO emp_resume\n"
         "      VALUES('000777', 'ascii', 'This is a new resume.');\n"
         "            ('777777', 'ascii', 'This is another new resume');\n"
       "    COMMIT;\n"
       "    DELETE FROM emp_resume WHERE empno = '000777';\n"
       "    DELETE FROM emp_resume WHERE empno = '777777';\n"
       "    COMMIT;\n"
       "    DELETE FROM emp_resume WHERE empno = '000140';\n"
       "    ROLLBACK;\n"
       "    ALTER TABLE emp_resume DATA CAPTURE NONE;\n"
       "    COMMIT;\n");

EXEC SQL ALTER TABLE emp_resume DATA CAPTURE CHANGES;
EMB_SQL_CHECK("SQL statement 1 -- invoke");

EXEC SQL COMMIT;
EMB_SQL_CHECK("SQL statement 2 -- invoke");

EXEC SQL INSERT INTO emp_resume
  VALUES('000777', 'ascii', 'This is a new resume.'),
        ('777777', 'ascii', 'This is another new resume');
EMB_SQL_CHECK("SQL statement 3 -- invoke");

EXEC SQL COMMIT;
EMB_SQL_CHECK("SQL statement 4 -- invoke");

EXEC SQL DELETE FROM emp_resume WHERE empno = '000777';
EMB_SQL_CHECK("SQL statement 5 -- invoke");

EXEC SQL DELETE FROM emp_resume WHERE empno = '777777';
EMB_SQL_CHECK("SQL statement 6 -- invoke");

EXEC SQL COMMIT;
EMB_SQL_CHECK("SQL statement 7 -- invoke");

EXEC SQL DELETE FROM emp_resume WHERE empno = '000140';
EMB_SQL_CHECK("SQL statement 8 -- invoke");

EXEC SQL ROLLBACK;
EMB_SQL_CHECK("SQL statement 9 -- invoke");

EXEC SQL ALTER TABLE emp_resume DATA CAPTURE NONE;
EMB_SQL_CHECK("SQL statement 10 -- invoke");

EXEC SQL COMMIT;
EMB_SQL_CHECK("SQL statement 11 -- invoke");

printf("\n  Start reading database log.\n");

logBuffer = NULL;
logBufferSize = 0;

/* The API db2ReadLog (Asynchronous Read Log) is used to extract records
   from the database logs, and to query the log manager for current
   log state information.
   This API can only be used on recoverable databases. */

/* Query the log manager for current log state information. */
```

```
readLogInput.iCallerAction  = DB2READLOG_QUERY;
readLogInput.piStartLSN     = NULL;
readLogInput.piEndLSN       = NULL;
readLogInput.poLogBuffer    = NULL;
readLogInput.iLogBufferSize = 0;
readLogInput.iFilterOption  = DB2READLOG_FILTER_ON;
readLogInput.poReadLogInfo  = &readLogInfo;

rc = db2ReadLog(db2Version810,
                &readLogInput,
                &sqlca);

DB2_API_CHECK("database log info -- get");

logBufferSize = 64 * 1024;
logBuffer = (char *)malloc(logBufferSize);

memcpy(&startLSN, &(readLogInfo.initialLSN), sizeof(startLSN));
memcpy(&endLSN, &(readLogInfo.nextStartLSN), sizeof(endLSN));

/* Extract a log record from the database logs, and
   read the first log sequence asynchronously. */
readLogInput.iCallerAction  = DB2READLOG_READ;
readLogInput.piStartLSN     = &startLSN;
readLogInput.piEndLSN       = &endLSN;
readLogInput.poLogBuffer    = logBuffer;
readLogInput.iLogBufferSize = logBufferSize;
readLogInput.iFilterOption  = DB2READLOG_FILTER_ON;
readLogInput.poReadLogInfo  = &readLogInfo;

rc = db2ReadLog(db2Version810,
                &readLogInput,
                &sqlca);
if (sqlca.sqlcode != SQLU_RLOG_READ_TO_CURRENT)
{
  DB2_API_CHECK("database logs -- read");
}
else
{
  if (readLogInfo.logRecsWritten == 0)
  {
    printf("\n  Database log empty.\n");
  }
}

/* display log buffer */
rc = LogBufferDisplay(logBuffer, readLogInfo.logRecsWritten);

while (sqlca.sqlcode != SQLU_RLOG_READ_TO_CURRENT)
{
  /* read the next log sequence */

  memcpy(&startLSN, &(readLogInfo.nextStartLSN), sizeof(startLSN));

  /* Extract a log record from the database logs, and
```

```
     read the next log sequence asynchronously. */
     rc = db2ReadLog(db2Version810,
                     &readLogInput,
                     &sqlca);
    if (sqlca.sqlcode != SQLU_RLOG_READ_TO_CURRENT)
    {
      DB2_API_CHECK("database logs -- read");
    }

    /* display log buffer */
    rc = LogBufferDisplay(logBuffer, readLogInfo.logRecsWritten);
  }

  /* free the log buffer */
  free(logBuffer);

  /* disconnect from the database */
  rc = DbDisconn(dbAlias);
  if (rc != 0)
  {
    return rc;
  }

  return 0;
} /* DbLogRecordsForCurrentConnectionRead */

int DbReadLogRecordsNoConn(char dbAlias[])
{
  int       rc = 0;
  struct    sqlca sqlca;
  char      logPath[SQL_PATH_SZ + 1];
  db2CfgParam cfgParameters[1];
  db2Cfg    cfgStruct;
  char      nodeName[] = "NODE0000\0";
  db2Uint32 readLogMemSize = 4 * 4096;
  char      *readLogMemory = NULL;
  struct db2ReadLogNoConnInitStruct readLogInit;
  struct db2ReadLogNoConnInfoStruct readLogInfo;
  struct db2ReadLogNoConnStruct readLogInput;
  SQLU_LSN  startLSN;
  SQLU_LSN  endLSN;
  char      *logBuffer = NULL;
  db2Uint32 logBufferSize = 0;
  struct db2ReadLogNoConnTermStruct readLogTerm;

  printf("\n*********************************\n");
  printf("*** NO DB CONNECTION READ LOG ***\n");
  printf("*********************************\n");
  printf("\nUSE THE DB2 APIs:\n");
  printf("  db2ReadLogNoConnInit -- Initialize No Db Connection Read Log\n");
  printf("  db2ReadLogNoConn -- No Db Connection Read Log\n");
  printf("  db2ReadLogNoConnTerm -- Terminate No Db Connection Read Log\n");
  printf("TO READ LOG RECORDS FROM A DATABASE LOG DIRECTORY.\n");

  /* Determine the logpath to read log files from */
```

```
cfgParameters[0].flags = 0;
cfgParameters[0].token = SQLF_DBTN_LOGPATH;
cfgParameters[0].ptrvalue =
  (char *)malloc((SQL_PATH_SZ + 1) * sizeof(char));

/* Initialize cfgStruct */
cfgStruct.numItems = 1;
cfgStruct.paramArray = cfgParameters;
cfgStruct.flags = db2CfgDatabase;
cfgStruct.dbname = dbAlias;

db2CfgGet(db2Version810, (void *)&cfgStruct, &sqlca);
DB2_API_CHECK("log path -- get");

strcpy(logPath, cfgParameters[0].ptrvalue);
free(cfgParameters[0].ptrvalue);

/* First we must allocate memory for the API's control blocks and log
   buffer */
readLogMemory = (char*)malloc(readLogMemSize);

/* Invoke the initialization API to set up the control blocks */
readLogInit.iFilterOption       = DB2READLOG_FILTER_ON;
readLogInit.piLogFilePath       = logPath;
readLogInit.piOverflowLogPath   = NULL;
readLogInit.iRetrieveLogs       = DB2READLOGNOCONN_RETRIEVE_OFF;
readLogInit.piDatabaseName      = dbAlias;
readLogInit.piNodeName          = nodeName;
readLogInit.iReadLogMemoryLimit = readLogMemSize;
readLogInit.poReadLogMemPtr     = &readLogMemory;

rc = db2ReadLogNoConnInit(db2Version810,
                          &readLogInit,
                          &sqlca);
if (sqlca.sqlcode != SQLU_RLOG_LSNS_REUSED)
{
  DB2_API_CHECK("database logs no db conn -- initialization");
}

/* Query for the current log information */
readLogInput.iCallerAction   = DB2READLOG_QUERY;
readLogInput.piStartLSN      = NULL;
readLogInput.piEndLSN        = NULL;
readLogInput.poLogBuffer     = NULL;
readLogInput.iLogBufferSize  = 0;
readLogInput.piReadLogMemPtr = readLogMemory;
readLogInput.poReadLogInfo   = &readLogInfo;

rc = db2ReadLogNoConn(db2Version810,
                      &readLogInput,
                      &sqlca);
if (sqlca.sqlcode != 0)
{
  DB2_API_CHECK("database logs no db conn -- query");
}
```

```
/* Read some log records */
logBufferSize = 64 * 1024;
logBuffer = (char *)malloc(logBufferSize);

memcpy(&startLSN, &(readLogInfo.nextStartLSN), sizeof(startLSN));
endLSN.lsnWord[0] = 0xffff;
endLSN.lsnWord[1] = 0xffff;
endLSN.lsnWord[2] = 0xffff;

readLogInput.iCallerAction   = DB2READLOG_READ;
readLogInput.piStartLSN      = &startLSN;
readLogInput.piEndLSN        = &endLSN;
readLogInput.poLogBuffer     = logBuffer;
readLogInput.iLogBufferSize  = logBufferSize;
readLogInput.piReadLogMemPtr = readLogMemory;
readLogInput.poReadLogInfo   = &readLogInfo;

rc = db2ReadLogNoConn(db2Version810,
                      &readLogInput,
                      &sqlca);
if (sqlca.sqlcode != SQLU_RLOG_READ_TO_CURRENT)
{
  DB2_API_CHECK("database logs no db conn -- read");
}
else
{
  if (readLogInfo.logRecsWritten == 0)
  {
    printf("\n  Database log empty.\n");
  }
}

/* Display the log records read */
rc = LogBufferDisplay(logBuffer, readLogInfo.logRecsWritten);

while (sqlca.sqlcode != SQLU_RLOG_READ_TO_CURRENT)
{
  /* read the next log sequence */
  memcpy(&startLSN, &(readLogInfo.nextStartLSN), sizeof(startLSN));

  /* Extract a log record from the database logs, and
   read the next log sequence asynchronously. */
  rc = db2ReadLogNoConn(db2Version810,
                        &readLogInput,
                        &sqlca);
  if (sqlca.sqlcode != SQLU_RLOG_READ_TO_CURRENT)
  {
    DB2_API_CHECK("database logs no db conn -- read");
  }

  /* display log buffer */
  rc = LogBufferDisplay(logBuffer, readLogInfo.logRecsWritten);
}
```

```
  printf("\nRead to end of logs.\n\n");
  free(logBuffer);

  readLogTerm.poReadLogMemPtr = &readLogMemory;

  rc = db2ReadLogNoConnTerm(db2Version810,
                            &readLogTerm,
                            &sqlca);
  if (sqlca.sqlcode != 0)
  {
    DB2_API_CHECK("database logs no db conn -- terminate");
  }

  return 0;
} /* DbReadLogRecordsNoConn */

int LogBufferDisplay(char *logBuffer, sqluint32 numLogRecords)
{
  int rc = 0;
  sqluint32 logRecordNb;
  sqluint32 recordSize;
  sqluint16 recordType;
  sqluint16 recordFlag;
  char *recordBuffer;

  /* initialize recordBuffer */
  recordBuffer = logBuffer + sizeof(SQLU_LSN);

  for (logRecordNb = 0; logRecordNb < numLogRecords; logRecordNb++)
  {
    recordSize = *(sqluint32 *)(recordBuffer);
    recordType = *(sqluint16 *)(recordBuffer + 4);
    recordFlag = *(sqluint16 *)(recordBuffer + 6);

    rc = LogRecordDisplay(recordBuffer, recordSize, recordType, recordFlag);
    /* update recordBuffer */
    recordBuffer = recordBuffer + recordSize + sizeof(SQLU_LSN);
  }

  return 0;
} /* LogBufferDisplay */

int LogRecordDisplay(char *recordBuffer,
                     sqluint32 recordSize,
                     sqluint16 recordType,
                     sqluint16 recordFlag)
{
  int rc = 0;
  sqluint32 logManagerLogRecordHeaderSize;
  char *recordDataBuffer;
  sqluint32 recordDataSize;
  char *recordHeaderBuffer;
  sqluint8 componentIdentifier;
  sqluint32 recordHeaderSize;
```

```
/* determine logManagerLogRecordHeaderSize */
if (recordType == 0x0043)
{ /* compensation */
  if (recordFlag & 0x0002)
  { /* propagatable */
    logManagerLogRecordHeaderSize = 32;
  }
  else
  {
    logManagerLogRecordHeaderSize = 26;
  }
}
else
{ /* non compensation */
  logManagerLogRecordHeaderSize = 20;
}

switch (recordType)
{
  case 0x008A:
  case 0x0084:
  case 0x0041:
    recordDataBuffer = recordBuffer + logManagerLogRecordHeaderSize;
    recordDataSize = recordSize - logManagerLogRecordHeaderSize;
    rc = SimpleLogRecordDisplay(recordType,
                                recordFlag,
                                recordDataBuffer,
                                recordDataSize);
    break;
  case 0x004E:
  case 0x0043:
    recordHeaderBuffer = recordBuffer + logManagerLogRecordHeaderSize;
    componentIdentifier = *(sqluint8 *)recordHeaderBuffer;
    switch (componentIdentifier)
    {
      case 1:
        recordHeaderSize = 6;
        break;
      default:
        printf("    Unknown complex log record: %lu %c %u\n",
               recordSize, recordType, componentIdentifier);
        return 1;
    }
    recordDataBuffer = recordBuffer +
                       logManagerLogRecordHeaderSize +
                       recordHeaderSize;
    recordDataSize = recordSize -
                     logManagerLogRecordHeaderSize -
                     recordHeaderSize;
    rc = ComplexLogRecordDisplay(recordType,
                                 recordFlag,
                                 recordHeaderBuffer,
                                 recordHeaderSize,
                                 componentIdentifier,
                                 recordDataBuffer,
                                   recordDataSize);
      break;
    default:
      printf("    Unknown log record: %lu \"%c\"\n",
             recordSize, (char)recordType);
      break;
  }

  return 0;
} /* LogRecordDisplay */

int SimpleLogRecordDisplay(sqluint16 recordType,
                           sqluint16 recordFlag,
                           char *recordDataBuffer,
                           sqluint32 recordDataSize)
{
  int rc = 0;
  sqluint32 timeTransactionCommited;
  sqluint16 authIdLen;
  char authId[129];

  switch (recordType)
  {
    case 138:
      printf("\n    Record type: Local pending list\n");
      timeTransactionCommited = *(sqluint32 *)(recordDataBuffer);
      authIdLen = *(sqluint16 *)(recordDataBuffer + 4);
      memcpy(authId, (char *)(recordDataBuffer + 6), authIdLen);
      authId[authIdLen] = '\0';
      printf("      %s: %lu\n",
             "UTC transaction committed (in seconds since 01/01/70)",
             timeTransactionCommited);
      printf("      authorization ID of the application: %s\n", authId);
      break;
    case 132:
      printf("\n    Record type: Normal commit\n");
      timeTransactionCommited = *(sqluint32 *)(recordDataBuffer);
      authIdLen = (sqluint16) (recordDataSize - 4);
      memcpy(authId, (char *)(recordDataBuffer + 4), authIdLen);
      authId[authIdLen] = '\0';
      printf("      %s: %lu\n",
             "UTC transaction committed (in seconds since 01/01/70)",
             timeTransactionCommited);
      printf("      authorization ID of the application: %s\n", authId);
      break;
    case 65:
      printf("\n    Record type: Normal abort\n");
      authIdLen = (sqluint16) (recordDataSize);
      memcpy(authId, (char *)(recordDataBuffer), authIdLen);
      authId[authIdLen] = '\0';
      printf("      authorization ID of the application: %s\n", authId);
      break;
    default:
      printf("    Unknown simple log record: %d %lu\n",
             recordType, recordDataSize);
      break;
  }

  return 0;
} /* SimpleLogRecordDisplay */

int ComplexLogRecordDisplay(sqluint16 recordType,
                            sqluint16 recordFlag,
                            char *recordHeaderBuffer,
                            sqluint32 recordHeaderSize,
                            sqluint8 componentIdentifier,
                            char *recordDataBuffer,
                            sqluint32 recordDataSize)
{
  int rc = 0;
  sqluint8 functionIdentifier;
  /* for insert, delete, undo delete */
  sqluint32 RID;
  sqluint16 subRecordLen;
  sqluint16 subRecordOffset;
  char *subRecordBuffer;
  /* for update */
  sqluint32 newRID;
  sqluint16 newSubRecordLen;
  sqluint16 newSubRecordOffset;
  char *newSubRecordBuffer;
  sqluint32 oldRID;
  sqluint16 oldSubRecordLen;
  sqluint16 oldSubRecordOffset;
  char *oldSubRecordBuffer;
  /* for alter table attributes */
  sqluint32 alterBitMask;
  sqluint32 alterBitValues;

  switch ((char)recordType)
  {
    case 'N':
      printf("\n    Record type: Normal\n");
      break;
    case 'C':
      printf("\n    Record type: Compensation\n");
      break;
    default:
      printf("\n    Unknown complex log record type: %c\n", recordType);
      break;
  }

  switch (componentIdentifier)
  {
    case 1:
      printf("      component ID: DMS log record\n");
      break;
    default:
      printf("      unknown component ID: %d\n", componentIdentifier);
      break;
```

```
}

functionIdentifier = *(sqluint8 *)(recordHeaderBuffer + 1);
switch (functionIdentifier)
{
  case 106:
    printf("      function ID: Delete Record\n");
    RID = *(sqluint32 *)(recordDataBuffer + 2);
    subRecordLen = *(sqluint16 *)(recordDataBuffer + 6);
    subRecordOffset = *(sqluint16 *)(recordDataBuffer + 10);
    printf("        RID: %lu\n", RID);
    printf("        subrecord length: %u\n", subRecordLen);
    printf("        subrecord offset: %u\n", subRecordOffset);
    subRecordBuffer = recordDataBuffer + 12;
    rc = LogSubRecordDisplay(subRecordBuffer, subRecordLen);
    break;
  case 111:
    printf("      function ID: Undo Delete Record\n");
    RID = *(sqluint32 *)(recordDataBuffer + 2);
    subRecordLen = *(sqluint16 *)(recordDataBuffer + 6);
    subRecordOffset = *(sqluint16 *)(recordDataBuffer + 10);
    printf("        RID: %lu\n", RID);
    printf("        subrecord length: %u\n", subRecordLen);
    printf("        subrecord offset: %u\n", subRecordOffset);
    subRecordBuffer = recordDataBuffer + 12;
    rc = LogSubRecordDisplay(subRecordBuffer, subRecordLen);
    break;
  case 118:
    printf("      function ID: Insert Record\n");
    RID = *(sqluint32 *)(recordDataBuffer + 2);
    subRecordLen = *(sqluint16 *)(recordDataBuffer + 6);
    subRecordOffset = *(sqluint16 *)(recordDataBuffer + 10);
    printf("        RID: %lu\n", RID);
    printf("        subrecord length: %u\n", subRecordLen);
    printf("        subrecord offset: %u\n", subRecordOffset);
    subRecordBuffer = recordDataBuffer + 12;
    rc = LogSubRecordDisplay(subRecordBuffer, subRecordLen);
    break;
  case 120:
    printf("      function ID: Update Record\n");
    oldRID = *(sqluint32 *)(recordDataBuffer + 2);
    oldSubRecordLen = *(sqluint16 *)(recordDataBuffer + 6);
    oldSubRecordOffset = *(sqluint16 *)(recordDataBuffer + 10);
    newRID = *(sqluint32 *)(recordDataBuffer +
                            12 +
                            oldSubRecordLen +
                            recordHeaderSize +
                            2);
    newSubRecordLen = *(sqluint16 *)(recordDataBuffer +
                                     12 +
                                     oldSubRecordLen +
                                     recordHeaderSize +
                                     6);
    newSubRecordOffset = *(sqluint16 *)(recordDataBuffer +
                                        12 +
                                      oldSubRecordLen +
                                      recordHeaderSize +
                                      10);
  printf("        oldRID: %lu\n", oldRID);
  printf("        old subrecord length: %u\n", oldSubRecordLen);
  printf("        old subrecord offset: %u\n", oldSubRecordOffset);
  oldSubRecordBuffer = recordDataBuffer + 12;
  rc = LogSubRecordDisplay(oldSubRecordBuffer, oldSubRecordLen);
  printf("        newRID: %lu\n", newRID);
  printf("        new subrecord length: %u\n", newSubRecordLen);
  printf("        new subrecord offset: %u\n", newSubRecordOffset);
  newSubRecordBuffer = recordDataBuffer +
                       12 +
                       oldSubRecordLen +
                       recordHeaderSize +
                       12;
  rc = LogSubRecordDisplay(newSubRecordBuffer, newSubRecordLen);
  break;
case 124:
  printf("      function ID:  Alter Table Attribute\n");
  alterBitMask = *(sqluint32 *)(recordDataBuffer + 2);
  alterBitValues = *(sqluint32 *)(recordDataBuffer + 6);
  if (alterBitMask & 0x00000001)
  {
    /* Alter the value of the 'propagation' attribute: */
    printf("        Propagation attribute is changed to: ");
    if (alterBitValues & 0x00000001)
    {
      printf("ON\n");
    }
    else
    {
      printf("OFF\n");
    }
  }
  if (alterBitMask & 0x00000002)
  {
    /* Alter the value of the 'pending' attribute: */
    printf("        Pending attribute is changed to: ");
    if (alterBitValues & 0x00000002)
    {
      printf("ON\n");
    }
    else
    {
      printf("OFF\n");
    }
  }
  if (alterBitMask & 0x00010000)
  {
    /* Alter the value of the 'append mode' attribute: */
    printf("        Append Mode attribute is changed to: ");
    if (alterBitValues & 0x00010000)
    {
      printf("ON\n");
```

```
        }
        else
        {
          printf("OFF\n");
        }
      }
      if (alterBitMask & 0x00200000)
      {
        /* Alter the value of the 'LF Propagation' attribute: */
        printf("        LF Propagation attribute is changed to: ");
        if (alterBitValues & 0x00200000)
        {
          printf("ON\n");
        }
        else
        {
          printf("OFF\n");
        }
      }
      if (alterBitMask & 0x00400000)
      {
        /* Alter the value of the 'LOB Propagation' attribute: */
        printf("        LOB Propagation attribute is changed to: ");
        if (alterBitValues & 0x00400000)
        {
          printf("ON\n");
        }
        else
        {
          printf("OFF\n");
        }
      }
      break;
    default:
      printf("      unknown function identifier: %u\n",
             functionIdentifier);
      break;
  }

  return 0;
} /* ComplexLogRecordDisplay */

int LogSubRecordDisplay(char *recordBuffer, sqluint16 recordSize)
{
  int rc = 0;
  sqluint8 recordType;
  sqluint8 updatableRecordType;
  sqluint16 userDataFixedLength;
  char *userDataBuffer;
  sqluint16 userDataSize;

  recordType = *(sqluint8 *)(recordBuffer);
  if ((recordType != 0) &&
      (recordType != 4) &&
      (recordType != 16))
  {
    printf("        Unknown subrecord type: %x\n", recordType);
  }
  else if (recordType == 4)
  {
    printf("        subrecord type: Special control\n");
  }
  else
  {
    /* recordType == 0 or recordType == 16
     * record Type 0 indicates a normal record
     * record Type 16, for the purposes of this program, should be treated
     * as type 0
     */
    printf("        subrecord type: Updatable, ");
    updatableRecordType = *(sqluint8 *)(recordBuffer + 4);
    if (updatableRecordType != 1)
    {
      printf("Internal control\n");
    }
    else
    {
      printf("Formatted user data\n");
      userDataFixedLength = *(sqluint16 *)(recordBuffer + 6);
      printf("        user data fixed length: %u\n",
             userDataFixedLength);
      userDataBuffer = recordBuffer + 8;
      userDataSize = recordSize - 8;
      rc = UserDataDisplay(userDataBuffer, userDataSize);
    }
  }

  return 0;
} /* LogSubRecordDisplay */

int UserDataDisplay(char *dataBuffer, sqluint16 dataSize)
{
  int rc = 0;

  sqluint16 line, col;

  printf("        user data:\n");

  for (line = 0; line * 10 < dataSize; line = line + 1)
  {
    printf("        ");
    for (col = 0; col < 10; col = col + 1)
    {
      if (line * 10 + col < dataSize)
      {
        printf("%02X ", dataBuffer[line * 10 + col]);
      }
      else
      {
        printf("   ");
      }
    }
    printf("*");
    for (col = 0; col < 10; col = col + 1)
    {
      if (line * 10 + col < dataSize)
      {
        if (isalpha(dataBuffer[line * 10 + col]) ||
            isdigit(dataBuffer[line * 10 + col]))
        {
          printf("%c", dataBuffer[line * 10 + col]);
        }
        else
        {
          printf(".");
        }
      }
      else
      {
        printf(" ");
      }
    }
    printf("*");
    printf("\n");
  }

  return 0;
} /* UserDataDisplay */

int DbRecoveryHistoryFileRead(char dbAlias[])
{
  int rc = 0;
  struct sqlca sqlca;
  struct db2HistoryOpenStruct dbHistoryOpenParam;
  sqluint32 numEntries;
  sqluint16 recoveryHistoryFileHandle;
  sqluint32 entryNb;
  struct db2HistoryGetEntryStruct dbHistoryEntryGetParam;
  struct db2HistoryData histEntryData;

  printf("\n*********************************************\n");
  printf("*** READ A DATABASE RECOVERY HISTORY FILE ***\n");
  printf("*********************************************\n");
  printf("\nUSE THE DB2 APIs:\n");
  printf("  db2HistoryOpenScan -- Open Recovery History File Scan\n");
  printf("  db2HistoryGetEntry -- Get Next Recovery History File Entry\n");
  printf("  db2HistoryCloseScan -- Close Recovery History File Scan\n");
  printf("TO READ A DATABASE RECOVERY HISTORY FILE.\n");

  /* initialize the data structures */
  dbHistoryOpenParam.piDatabaseAlias = dbAlias;
  dbHistoryOpenParam.piTimestamp = NULL;
  dbHistoryOpenParam.piObjectName = NULL;
  dbHistoryOpenParam.iCallerAction = DB2HISTORY_LIST_HISTORY;
  dbHistoryEntryGetParam.pioHistData = &histEntryData;
  dbHistoryEntryGetParam.iCallerAction = DB2HISTORY_GET_ALL;
  rc = HistoryEntryDataFieldsAlloc(&histEntryData);
  if (rc != 0)
  {
    return rc;
  }

  /*******************************************/
  /* OPEN THE DATABASE RECOVERY HISTORY FILE */
  /*******************************************/
  printf("\n  Open recovery history file for '%s' database.\n", dbAlias);

  /* open the recovery history file to scan */
  db2HistoryOpenScan(db2Version810, &dbHistoryOpenParam, &sqlca);
  DB2_API_CHECK("database recovery history file -- open");

  numEntries = dbHistoryOpenParam.oNumRows;

  /* dbHistoryOpenParam.oHandle returns the handle for scan access */
  recoveryHistoryFileHandle = dbHistoryOpenParam.oHandle;
  dbHistoryEntryGetParam.iHandle = recoveryHistoryFileHandle;

  /**********************************************/
  /* READ AN ENTRY IN THE RECOVERY HISTORY FILE */
  /**********************************************/
  for (entryNb = 0; entryNb < numEntries; entryNb = entryNb + 1)
  {
    printf("\n  Read entry number %u.\n", entryNb);

    /* get the next entry from the recovery history file */
    db2HistoryGetEntry(db2Version810, &dbHistoryEntryGetParam, &sqlca);
    DB2_API_CHECK("database recovery history file entry -- read")

    /* display the entries in the recovery history file */
    printf("\n  Display entry number %u.\n", entryNb);
    rc = HistoryEntryDisplay(histEntryData);
  }

  /********************************************/
  /* CLOSE THE DATABASE RECOVERY HISTORY FILE */
  /********************************************/
  printf("\n  Close recovery history file for '%s' database.\n", dbAlias);

  /* The API db2HistoryCloseScan ends the recovery history file scan and
     frees DB2 resources required for the scan. */
  db2HistoryCloseScan(db2Version810, &recoveryHistoryFileHandle, &sqlca);
  DB2_API_CHECK("database recovery history file -- close");

  /* free the allocated memory */
  rc = HistoryEntryDataFieldsFree(&histEntryData);

  return 0;
} /* DbRecoveryHistoryFileRead */
```

```
int HistoryEntryDataFieldsAlloc(struct db2HistoryData *pHistEntryData)
{
  int rc = 0;
  sqluint32 tsNb;

  strcpy(pHistEntryData->ioHistDataID, "SQLUHINF");

  pHistEntryData->oObjectPart.pioData = malloc(17 + 1);
  pHistEntryData->oObjectPart.iLength = 17 + 1;

  pHistEntryData->oEndTime.pioData = malloc(12 + 1);
  pHistEntryData->oEndTime.iLength = 12 + 1;

  pHistEntryData->oFirstLog.pioData = malloc(8 + 1);
  pHistEntryData->oFirstLog.iLength = 8 + 1;

  pHistEntryData->oLastLog.pioData = malloc(8 + 1);
  pHistEntryData->oLastLog.iLength = 8 + 1;

  pHistEntryData->oID.pioData = malloc(128 + 1);
  pHistEntryData->oID.iLength = 128 + 1;

  pHistEntryData->oTableQualifier.pioData = malloc(128 + 1);
  pHistEntryData->oTableQualifier.iLength = 128 + 1;

  pHistEntryData->oTableName.pioData = malloc(128 + 1);
  pHistEntryData->oTableName.iLength = 128 + 1;

  pHistEntryData->oLocation.pioData = malloc(128 + 1);
  pHistEntryData->oLocation.iLength = 128 + 1;

  pHistEntryData->oComment.pioData = malloc(128 + 1);
  pHistEntryData->oComment.iLength = 128 + 1;

  pHistEntryData->oCommandText.pioData = malloc(128 + 1);
  pHistEntryData->oCommandText.iLength = 128 + 1;

  pHistEntryData->poEventSQLCA =
    (struct sqlca *)malloc(sizeof(struct sqlca));

  pHistEntryData->poTablespace = (db2Char *)malloc(3 * sizeof(db2Char));
  for (tsNb = 0; tsNb < 3; tsNb = tsNb + 1)
  {
    pHistEntryData->poTablespace[tsNb].pioData = malloc(18 + 1);
    pHistEntryData->poTablespace[tsNb].iLength = 18 + 1;
  }

  pHistEntryData->iNumTablespaces = 3;

  return 0;
} /* HistoryEntryDataFieldsAlloc */

int HistoryEntryDisplay(struct db2HistoryData histEntryData)
{
  int rc = 0;
```

```
char buf[129];
sqluint32 tsNb;

memcpy(buf, histEntryData.oObjectPart.pioData,
            histEntryData.oObjectPart.oLength);
buf[histEntryData.oObjectPart.oLength] = '\0';
printf("    object part: %s\n", buf);

memcpy(buf, histEntryData.oEndTime.pioData,
            histEntryData.oEndTime.oLength);
buf[histEntryData.oEndTime.oLength] = '\0';
printf("    end time: %s\n", buf);

memcpy(buf, histEntryData.oFirstLog.pioData,
            histEntryData.oFirstLog.oLength);
buf[histEntryData.oFirstLog.oLength] = '\0';
printf("    first log: %s\n", buf);

memcpy(buf, histEntryData.oLastLog.pioData,
            histEntryData.oLastLog.oLength);
buf[histEntryData.oLastLog.oLength] = '\0';
printf("    last log: %s\n", buf);

memcpy(buf, histEntryData.oID.pioData, histEntryData.oID.oLength);
buf[histEntryData.oID.oLength] = '\0';
printf("    ID: %s\n", buf);

memcpy(buf, histEntryData.oTableQualifier.pioData,
            histEntryData.oTableQualifier.oLength);
buf[histEntryData.oTableQualifier.oLength] = '\0';
printf("    table qualifier: %s\n", buf);

memcpy(buf, histEntryData.oTableName.pioData,
            histEntryData.oTableName.oLength);
buf[histEntryData.oTableName.oLength] = '\0';
printf("    table name: %s\n", buf);

memcpy(buf, histEntryData.oLocation.pioData,
            histEntryData.oLocation.oLength);
buf[histEntryData.oLocation.oLength] = '\0';
printf("    location: %s\n", buf);

memcpy(buf, histEntryData.oComment.pioData,
            histEntryData.oComment.oLength);
buf[histEntryData.oComment.oLength] = '\0';
printf("    comment: %s\n", buf);

memcpy(buf, histEntryData.oCommandText.pioData,
            histEntryData.oCommandText.oLength);
buf[histEntryData.oCommandText.oLength] = '\0';
printf("    command text: %s\n", buf);
printf("    history file entry ID: %u\n", histEntryData.oEID.ioHID);
printf("    table spaces:\n");

for (tsNb = 0; tsNb < histEntryData.oNumTablespaces; tsNb = tsNb + 1)
```

```
  {
    memcpy(buf, histEntryData.poTablespace[tsNb].pioData,
                histEntryData.poTablespace[tsNb].oLength);
    buf[histEntryData.poTablespace[tsNb].oLength] = '\0';
    printf("      %s\n", buf);
  }

  printf("    type of operation: %c\n", histEntryData.oOperation);
  printf("    granularity of the operation: %c\n", histEntryData.oObject);
  printf("    operation type: %c\n", histEntryData.oOptype);
  printf("    entry status: %c\n", histEntryData.oStatus);
  printf("    device type: %c\n", histEntryData.oDeviceType);
  printf("    SQLCA:\n");
  printf("      sqlcode: %ld\n", histEntryData.poEventSQLCA->sqlcode);
  memcpy(buf, histEntryData.poEventSQLCA->sqlstate, 5);
  buf[5] = '\0';
  printf("      sqlstate: %s\n", buf);
  memcpy(buf, histEntryData.poEventSQLCA->sqlerrmc,
              histEntryData.poEventSQLCA->sqlerrml);
  buf[histEntryData.poEventSQLCA->sqlerrml] = '\0';
  printf("      message: %s\n", buf);

  return 0;
} /* HistoryEntryDisplay */

int HistoryEntryDataFieldsFree(struct db2HistoryData *pHistEntryData)
{
  int rc = 0;
  sqluint32 tsNb;

  free(pHistEntryData->oObjectPart.pioData);
  free(pHistEntryData->oEndTime.pioData);
  free(pHistEntryData->oFirstLog.pioData);
  free(pHistEntryData->oLastLog.pioData);
  free(pHistEntryData->oID.pioData);
  free(pHistEntryData->oTableQualifier.pioData);
  free(pHistEntryData->oTableName.pioData);
  free(pHistEntryData->oLocation.pioData);
  free(pHistEntryData->oComment.pioData);
  free(pHistEntryData->oCommandText.pioData);
  free(pHistEntryData->poEventSQLCA);

  for (tsNb = 0; tsNb < 3; tsNb = tsNb + 1)
  {
    free(pHistEntryData->poTablespace[tsNb].pioData);
  }

  free(pHistEntryData->poTablespace);

  return 0;
} /* HistoryEntryDataFieldsFree */

int DbFirstRecoveryHistoryFileEntryUpdate(char dbAlias[],
                                          char user[],
                                          char pswd[])
```

```
{
  int rc = 0;
  struct sqlca sqlca;
  struct db2HistoryOpenStruct dbHistoryOpenParam;
  sqluint16 recoveryHistoryFileHandle;
  struct db2HistoryGetEntryStruct dbHistoryEntryGetParam;
  struct db2HistoryData histEntryData;
  char newLocation[DB2HISTORY_LOCATION_SZ + 1];
  char newComment[DB2HISTORY_COMMENT_SZ + 1];
  struct db2HistoryUpdateStruct dbHistoryUpdateParam;

  printf("\n***************************************************\n");
  printf("*** UPDATE A DATABASE RECOVERY HISTORY FILE ENTRY ***\n");
  printf("*****************************************************\n");
  printf("\nUSE THE DB2 APIs:\n");
  printf("  db2HistoryOpenScan -- Open Recovery History File Scan\n");
  printf("  db2HistoryGetEntry -- Get Next Recovery History File Entry\n");
  printf("  db2HistoryUpdate -- Update Recovery History File\n");
  printf("  db2HistoryCloseScan -- Close Recovery History File Scan\n");
  printf("TO UPDATE A DATABASE RECOVERY HISTORY FILE ENTRY.\n");

  /* initialize data structures */
  dbHistoryOpenParam.piDatabaseAlias = dbAlias;
  dbHistoryOpenParam.piTimestamp = NULL;
  dbHistoryOpenParam.piObjectName = NULL;
  dbHistoryOpenParam.iCallerAction = DB2HISTORY_LIST_HISTORY;
  dbHistoryEntryGetParam.pioHistData = &histEntryData;
  dbHistoryEntryGetParam.iCallerAction = DB2HISTORY_GET_ALL;
  rc = HistoryEntryDataFieldsAlloc(&histEntryData);
  if (rc != 0)
  {
    return rc;
  }

  /*******************************************/
  /* OPEN THE DATABASE RECOVERY HISTORY FILE */
  /*******************************************/
  printf("\n  Open the recovery history file for '%s' database.\n", dbAlias);

  /* The API db2HistoryOpenScan starts a recovery history file scan */
  db2HistoryOpenScan(db2Version810, &dbHistoryOpenParam, &sqlca);
  DB2_API_CHECK("database recovery history file -- open");

  /* dbHistoryOpenParam.oHandle returns the handle for scan access */
  recoveryHistoryFileHandle = dbHistoryOpenParam.oHandle;
  dbHistoryEntryGetParam.iHandle = recoveryHistoryFileHandle;

  /*****************************************************/
  /* READ THE FIRST ENTRY IN THE RECOVERY HISTORY FILE */
  /*****************************************************/
  printf("\n  Read the first entry in the recovery history file.\n");

  /* The API db2HistoryGetEntry gets the next entry from the recovery
     history file. */
  db2HistoryGetEntry(db2Version810, &dbHistoryEntryGetParam, &sqlca);
```

```
DB2_API_CHECK("first recovery history file entry -- read");
printf("\n  Display the first entry.\n");

/* HistoryEntryDisplay is a support function used to display the entries
   in the recovery history file. */
rc = HistoryEntryDisplay(histEntryData);

/* update the first history file entry */
rc = DbConn(dbAlias, user, pswd);
if (rc != 0)
{
  return rc;
}

strcpy(newLocation, "this is the NEW LOCATION");
strcpy(newComment, "this is the NEW COMMENT");
printf("\n  Update the first entry in the history file:\n");
printf("    new location = '%s'\n", newLocation);
printf("    new comment = '%s'\n", newComment);
dbHistoryUpdateParam.piNewLocation = newLocation;
dbHistoryUpdateParam.piNewDeviceType = NULL;
dbHistoryUpdateParam.piNewComment = newComment;
dbHistoryUpdateParam.iEID.ioNode = histEntryData.oEID.ioNode;
dbHistoryUpdateParam.iEID.ioHID = histEntryData.oEID.ioHID;

/* The API db2HistoryUpdate can be used to update the location,
   device type, or comment in a history file entry. */

/* Call this API to update the location and comment of the first
   entry in the history file: */
db2HistoryUpdate(db2Version810, &dbHistoryUpdateParam, &sqlca);
DB2_API_CHECK("first history file entry -- update");

rc = DbDisconn(dbAlias);
if (rc != 0)
{
  return rc;
}

/*********************************************/
/* CLOSE THE DATABASE RECOVERY HISTORY FILE */
/*********************************************/
printf("\n  Close recovery history file for '%s' database.\n", dbAlias);

/* The API db2HistoryCloseScan ends the recovery history file scan and
   frees DB2 resources required for the scan. */
db2HistoryCloseScan(db2Version810, &recoveryHistoryFileHandle, &sqlca);
DB2_API_CHECK("database recovery history file -- close");


/***********************************************/
/* RE-OPEN THE DATABASE RECOVERY HISTORY FILE */
/***********************************************/
printf("\n  Open the recovery history file for '%s' database.\n", dbAlias);
```

```
  /* starts a recovery history file scan */
  db2HistoryOpenScan(db2Version810, &dbHistoryOpenParam, &sqlca);
  DB2_API_CHECK("database recovery history file -- open");

  recoveryHistoryFileHandle = dbHistoryOpenParam.oHandle;

  dbHistoryEntryGetParam.iHandle = recoveryHistoryFileHandle;
  printf("\n  Read the first recovery history file entry.\n");

  /**********************************************************************/
  /* READ THE FIRST ENTRY IN THE RECOVERY HISTORY FILE AFTER MODIFICATION */
  /**********************************************************************/
  db2HistoryGetEntry(db2Version810, &dbHistoryEntryGetParam, &sqlca);
  DB2_API_CHECK("first recovery history file entry -- read");

  printf("\n  Display the first entry.\n");
  rc = HistoryEntryDisplay(histEntryData);

  /*******************************************/
  /* CLOSE THE DATABASE RECOVERY HISTORY FILE */
  /*******************************************/
  printf("\n  Close the recovery history file for '%s' database.\n",
         dbAlias);

  /* ends the recovery history file scan */
  db2HistoryCloseScan(db2Version810, &recoveryHistoryFileHandle, &sqlca);
  DB2_API_CHECK("database recovery history file -- close");

  /* free the allocated memory */
  rc = HistoryEntryDataFieldsFree(&histEntryData);

  return 0;
} /* DbFirstRecoveryHistoryFileEntryUpdate */

int DbRecoveryHistoryFilePrune(char dbAlias[], char user[], char pswd[])
{
  int rc = 0;
  struct sqlca sqlca;
  struct db2PruneStruct histPruneParam;
  char timeStampPart[14 + 1];

  printf("\n***************************************\n");
  printf("*** PRUNE THE RECOVERY HISTORY FILE ***\n");
  printf("***************************************\n");
  printf("\nUSE THE DB2 API:\n");
  printf("  db2Prune -- Prune Recovery History File\n");
  printf("AND THE SQL STATEMENTS:\n");
  printf("  CONNECT\n");
  printf("  CONNECT RESET\n");
  printf("TO PRUNE THE RECOVERY HISTORY FILE.\n");

  /* Connect to the database: */
  rc = DbConn(dbAlias, user, pswd);
  if (rc != 0)
  {
    return rc;
  }

  /* Prune the recovery history file: */
  printf("\n  Prune the recovery history file for '%s' database.\n",
         dbAlias);

  /* timeStampPart is a pointer to a string specifying a time stamp or
     log sequence number. Time stamp is used here to select records for
     deletion. All entries equal to or less than the time stamp will be
     deleted. */
  histPruneParam.piString = timeStampPart;
  strcpy(timeStampPart, "2010"); /* year 2010 */

  /* The action DB2PRUNE_ACTION_HISTORY removes history file entries: */
  histPruneParam.iAction = DB2PRUNE_ACTION_HISTORY;

  /* The option DB2PRUNE_OPTION_FORCE forces the removal of the last backup: */
  histPruneParam.iOptions = DB2PRUNE_OPTION_FORCE;

  /* db2Prune can be called to delete entries from the recovery history file
     or log files from the active log path. Here we call it to delete
     entries from the recovery history file.
     You must have SYSADM, SYSCTRL, SYSMAINT, or DBADM authority to prune
     the recovery history file. */
  db2Prune(db2Version810, &histPruneParam, &sqlca);
  DB2_API_CHECK("recovery history file -- prune");

  /* Disconnect from the database: */
  rc = DbDisconn(dbAlias);
  if (rc != 0)
  {
    return rc;
  }

  return 0;
} /* DbRecoveryHistoryFilePrune */
```

Appendix F. Tivoli Storage Manager

When calling the BACKUP DATABASE or RESTORE DATABASE commands, you can specify that you want to use the Tivoli Storage Manager (TSM) product to manage database or table space backup or restore operation. The minimum required level of TSM client API is Version 4.2.0, except on a 64-bit Solaris system which requires TSM client API Version 4.2.1.

Configuring a Tivoli Storage Manager Client

Before the database manager can use the TSM option, the following steps may be required to configure the TSM environment:

1. A functioning TSM client and server must be installed and configured. In addition, the TSM client API must be installed.
2. Set the environment variables used by the TSM client API:

   DSMI_DIR Identifies the user-defined directory path where the API trusted agent file (`dsmtca`) is located.

   DSMI_CONFIG
   Identifies the user-defined directory path to the `dsm.opt` file, which contains the TSM user options. Unlike the other two variables, this variable should contain a fully qualified path and file name.

   DSMI_LOG Identifies the user-defined directory path where the error log (`dsierror.log`) will be created.

   Note: In a multi-partition database environment these settings must be specified in the `sqllib/userprofile` directory.
3. If any changes are made to these environment variables and the database manager is running, you should:
   - Stop the database manager using the **db2stop** command.
   - Start the database manager using the **db2start** command.
4. Depending on the server's configuration, a Tivoli client may require a password to interface with a TSM server. If the TSM environment is configured to use `PASSWORDACCESS=generate`, the Tivoli client needs to have its password established.

   The executable file `dsmapipw` is installed in the `sqllib/adsm` directory of the instance owner. This executable allows you to establish and reset the TSM password.

To execute the `dsmapipw` command, you must be logged in as the local administrator or "root" user. When this command is executed, you will be prompted for the following information:

- *Old password*, which is the current password for the TSM node, as recognized by the TSM server. The first time you execute this command, this password will be the one provided by the TSM administrator at the time your node was registered on the TSM server.
- *New password*, which is the new password for the TSM node, stored at the TSM server. (You will be prompted twice for the new password, to check for input errors.)

Note: Users who invoke the BACKUP DATABASE or RESTORE DATABASE commands do not need to know this password. You only need to run the `dsmapipw` command to establish a password for the initial connection, and after the password has been reset on the TSM server.

Considerations for Using Tivoli Storage Manager

To use specific features within TSM, you may be required to give the fully qualified path name of the object using the feature. (Remember that on Windows operating systems, the `\` will be used instead of `/`.) The fully qualified path name of:

- A full database backup object is:
  `/<database>/NODEnnnn/FULL_BACKUP.timestamp.seq_no`
- An incremental database backup object is:
  `/<database>/NODEnnnn/DB_INCR_BACKUP.timestamp.seq_no`
- An incremental delta database backup object is_
  `/<database>/NODEnnnn/DB_DELTA_BACKUP.timestamp.seq_no`
- A full table space backup object is:
  `/<database>/NODEnnnn/TSP_BACKUP.timestamp.seq_no`
- An incremental table space backup object is:
  `/<database>/NODEnnnn/TSP_INCR_BACKUP.timestamp.seq_no`
- An incremental delta table space backup object is:
  `/<database>/NODEnnnn/TSP_DELTA_BACKUP.timestamp.seq_no`

where `<database>` is the database alias name, and `NODEnnnn` is the node number. The names shown in uppercase characters must be entered as shown.

- In the case where you have multiple backup images using the same database alias name, the time stamp and sequence number become the distinguishing part of a fully qualified name. You will need to query TSM to determine which backup version to use.
- Individual backup images are pooled into file spaces that TSM manages. Individual backup images can only be manipulated through the TSM APIs, or through **db2adutl** which uses these APIs.

- The TSM server will time out a session if the Tivoli client does not respond within the period of time specified by the `COMMTIMEOUT` parameter in the server's configuration file. Three factors can contribute to a timeout problem:
  - The `COMMTIMEOUT` parameter may be set too low at the TSM server. For example, during a restore operation, a timeout can occur if large DMS table spaces are being created. The recommended value for this parameter is 6000 seconds.
  - The DB2 backup or restore buffer may be too large.
  - Database activity during an online backup operation may be too high.
- Use multiple sessions to increase throughput (only if sufficient hardware is available on the TSM server).

Related concepts:

- "Managing Log Files" on page 45
- "Tivoli Space Manager Hierarchical Storage Manager (AIX)" in the *Quick Beginnings for Data Links Manager*

Related reference:

- "db2adutl - Work with TSM Archived Images" on page 209

Appendix G. User Exit for Database Recovery

You can develop a *user exit program* to automate log file archiving and retrieval. Before invoking a user exit program for log file archiving or retrieval, ensure that the *userexit* database configuration parameter has been set to `YES`. This also enables your database for rollforward recovery.

When a user exit program is invoked, the database manager passes control to the executable file, `db2uext2`. The database manager passes parameters to `db2uext2` and, on completion, the program passes a return code back to the database manager. Because the database manager handles a limited set of return conditions, the user exit program should be able to handle error conditions (see “Error Handling” on page 325). And because only one user exit program can be invoked within a database manager instance, it must have a section for each of the operations it may be asked to perform.

The following topics are covered:

- “Sample User Exit Programs”
- “Calling Format” on page 324
- “Error Handling” on page 325

Sample User Exit Programs

Sample user exit programs are provided for all supported platforms. You can modify these programs to suit your particular requirements. The sample programs are well commented with information that will help you to use them most effectively.

You should be aware that user exit programs must *copy* log files from the active log path to the archive log path. Do not remove log files from the active log path. (This could cause problems during database recovery.) DB2® removes archived log files from the active log path when these log files are no longer needed for recovery.

Following is a description of the sample user exit programs that are shipped with DB2.

- **UNIX® based systems**

  The user exit sample programs for DB2 for UNIX based systems are found in the `sqllib/samples/c` subdirectory. Although the samples provided are coded in C, your user exit program can be written in a different programming language.

  Your user exit program must be an executable file whose name is `db2uext2`.

There are four sample user exit programs for UNIX based systems:

- `db2uext2.ctsm`

  This sample uses Tivoli[®] Storage Manager to archive and retrieve database log files.
- `db2uext2.ctape`

  This sample uses tape media to archive and retrieve database log files .
- `db2uext2.cdisk`

  This sample uses the operating system COPY command and disk media to archive and retrieve database log files.
- `db2uxt2.cxbsa`

  This sample works with the XBSA Draft 0.8 published by the X/Open group. It can be used to archive and retrieve database log files. This sample is only supported on AIX.

- **Windows[®] operating systems**

  The user exit sample programs for DB2 for Windows operating systems are found in the `sqllib\samples\c` subdirectory. Although the samples provided are coded in C, your user exit program can be written in a different programming language.

  Your user exit program must be an executable file whose name is `db2uext2`.

  There are two sample user exit programs for Windows operating systems:

  - `db2uext2.ctsm`

    This sample uses Tivoli Storage Manager to archive and retrieve database log files.
  - `db2uext2.cdisk`

    This sample uses the operating system COPY command and disk media to archive and retrieve database log files.

Calling Format

When the database manager calls a user exit program, it passes a set of parameters (of data type CHAR) to the program. The calling format is dependent on your operating system:

```
db2uext2 -OS<os> -RL<db2rel> -RQ<request> -DB<dbname>
-NN<nodenum> -LP<logpath> -LN<logname> -AP<tsmpasswd>
-SP<startpage> -LS<logsize>
```

os Specifies the platform on which the instance is running. Valid values are: `AIX`[®], `Solaris`, `HP-UX`, `SCO`, `Linux`, and `NT`.

db2rel Specifies the DB2 release level. For example, `SQL07020`.

request Specifies a request type. Valid values are: `ARCHIVE` and `RETRIEVE`.

dbname Specifies a database name.

nodenum Specifies the local node number, such as `5`, for example.

logpath Specifies the fully qualified path to the log files. The path must contain the trailing path separator. For example, `/u/database/log/path/`, or `d:\logpath\`.

logname Specifies the name of the log file that is to be archived or retrieved, such as `S0000123.LOG`, for example.

tsmpasswd Specifies the TSM password. (If a value for the database configuration parameter *tsm_password* has previously been specified, that value is passed to the user exit program.)

startpage Specifies the number of 4-KB offset pages of the device at which the log extent starts.

logsize Specifies the size of the log extent, in 4-KB pages. This parameter is only valid if a raw device is used for logging.

Error Handling

Your user exit program should be designed to provide specific and meaningful return codes, so that the database manager can interpret them correctly. Because the user exit program is called by the underlying operating system command processor, the operating system itself could return error codes. And because these error codes are not remapped, use the operating system message help utility to obtain information about them.

Table 8 shows the codes that can be returned by a user exit program, and describes how these codes are interpreted by the database manager. If a return code is not listed in the table, it is treated as if its value were 32.

Table 8. User Exit Program Return Codes. Applies to archiving and retrieval operations only.

Return Code	Explanation
0	Successful.
4	Temporary resource error encountered.[a]
8	Operator intervention is required.[a]
12	Hardware error.[b]
16	Error with the user exit program or a software function used by the program.[b]
20	Error with one or more of the parameters passed to the user exit program. Verify that the user exit program is correctly processing the specified parameters.[b]
24	The user exit program was not found. [b]

Table 8. User Exit Program Return Codes (continued). Applies to archiving and retrieval operations only.

Return Code	Explanation
28	Error caused by an input/output (I/O) failure, or by the operating system.[b]
32	The user exit program was terminated by the user.[b]
255	Error caused by the user exit program not being able to load the library file for the executable.[c]

[a] For archiving or retrieval requests, a return code of 4 or 8 causes a retry in five minutes. If the user exit program continues to return 4 or 8 on retrieve requests for the same log file, DB2 will continue to retry until successful. (This applies to rollforward operations, or calls to the **db2ReadLog** API, which is used by the replication utility.)

[b] User exit requests are suspended for five minutes. During this time, all requests are ignored, including the request that caused the error condition. Following this five-minute suspension, the next request is processed. If this request is processed without error, processing of new user exit requests continues, and DB2 reissues the archive request that failed or was suspended previously. If a return code greater than 8 is generated during the retry, requests are suspended for an additional five minutes. The five-minute suspensions continue until the problem is corrected, or the database is stopped and restarted. Once all applications have disconnected from the database, DB2 issues an archive request for any log file that may not have been successfully archived previously. If the user exit program fails to archive log files, your disk may become filled with log files, and performance may be degraded. Once the disk becomes full, the database manager will not accept further application requests for database updates. If the user exit program was called to retrieve log files, rollforward recovery is suspended, but not stopped, unless the ROLLFORWARD STOP option was specified. If the STOP option was not specified, you can correct the problem and resume recovery.

[c] If the user exit program returns error code 255, it is likely that the program cannot load the library file for the executable. To verify this, manually invoke the user exit program. More information is displayed.

Note: During archiving and retrieval operations, an alert message is issued for all return codes except 0, and 4. The alert message contains the return code from the user exit program, and a copy of the input parameters that were provided to the user exit program.

Appendix H. Backup and Restore APIs for Vendor Products

Backup and Restore APIs for Vendor Products

DB2 provides interfaces that can be used by third-party media management products to store and retrieve data for backup and restore operations. This function is designed to augment the backup and restore data targets of diskette, disk, tape, and Tivoli Storage Manager, that are supported as a standard part of DB2.

These third-party media management products will be referred to as vendor products in the remainder of this appendix.

DB2 defines a set of function prototypes that provide a general purpose data interface to backup and restore that can be used by many vendors. These functions are to be provided by the vendor in a shared library on UNIX based systems, or DLL on the Windows operating system. When the functions are invoked by DB2, the shared library or DLL specified by the calling backup or restore routine is loaded and the functions provided by the vendor are called to perform the required tasks.

This appendix is divided into four parts:

- Operational overview of DB2's interaction with vendor products.
- Detailed descriptions of DB2's vendor APIs.
- Details on invoking backup and restore using vendor products.
- Information on the data structures used in the API calls.

Operational Overview

Five functions are defined to interface DB2 and the vendor product:

- sqluvint - Initialize and Link to Device
- sqluvget - Reading Data from Device
- sqluvput - Writing Data to Device
- sqluvend - Unlink the Device
- sqluvdel - Delete Committed Session

DB2 will call these functions, and they should be provided by the vendor product in a shared library on UNIX based systems, or in a DLL on the Windows operating system.

Note: The shared library or DLL code will be run as part of the database engine code. Therefore, it must be reentrant and thoroughly debugged. An errant function may compromise data integrity of the database.

The sequence of functions that DB2 will call during a specific backup or restore operation depends on:

- The number of sessions that will be utilized.
- Whether it is a backup or a restore operation.
- The PROMPTING mode that is specified on the backup or restore operation.
- The characteristics of the device on which the data is stored.
- The errors that may be encountered during the operation.

Number of Sessions

DB2 supports the backup and restore of database objects using one or more data streams or sessions. A backup or restore using three sessions would require three physical or logical devices to be available. When vendor device support is being used, it is the vendor's functions that are responsible for managing the interface to each physical or logical device. DB2 simply sends or receives data buffers to or from the vendor provided functions.

The number of sessions to be used is specified as a parameter by the application that calls the backup or restore database function. This value is provided in the INIT-INPUT structure used by **sqluvint** .

DB2 will continue to initialize sessions until the specified number is reached, or it receives an SQLUV_MAX_LINK_GRANT warning return code from an **sqluvint** call. In order to warn DB2 that it has reached the maximum number of sessions that it can support, the vendor product will require code to track the number of active sessions. Failure to warn DB2 could lead to a DB2 initialize session request that fails, resulting in a termination of all sessions and the failure of the entire backup or restore operation.

When the operation is backup, DB2 writes a media header record at the beginning of each session. The record contains information that DB2 uses to identify the session during a restore operation. DB2 uniquely identifies each session by appending a sequence number to the name of the backup image. The number starts at one for the first session, and is incremented by one each time another session is initiated with an **sqluvint** call for a backup or a restore operation.

When the backup operation completes successfully, DB2 writes a media trailer to the last session it closes. This trailer includes information that tells DB2 how many sessions were used to perform the backup operation. During a restore operation, this information is used to ensure all the sessions, or data streams, have been restored.

Operation with No Errors, Warnings or Prompting

For backup, the following sequence of calls is issued by DB2 for *each* session.

```
sqluvint, action = SQLUV_WRITE
```

followed by 1 to n

```
sqluvput
```

followed by 1

```
sqluvend, action = SQLUV_COMMIT
```

When DB2 issues an **sqluvend** call (action SQLUV_COMMIT), it expects the vendor product to appropriately save the output data. A return code of SQLUV_OK to DB2 indicates success.

The DB2-INFO structure, used on the **sqluvint** call, contains the information required to identify the backup. A sequence number is supplied. The vendor product may choose to save this information. DB2 will use it during restore to identify the backup that will be restored.

For restore, the sequence of calls for each session is:

```
sqluvint, action = SQLUV_READ
```

followed by 1 to n

```
sqluvget
```

followed by 1

```
sqluvend, action = SQLUV_COMMIT
```

The information in the DB2-INFO structure used on the **sqluvint** call will contain the information required to identify the backup. A sequence number is not supplied. DB2 expects that all backup objects (session outputs committed during a backup) will be returned. The first backup object returned is the object generated with sequence number 1, and all other objects are restored in no specific order. DB2 checks the media tail to ensure that all objects have been processed.

Note: Not all vendor products will keep a record of the names of the backup objects. This is most likely when the backups are being done to tapes, or other media of limited capacity. During the initialization of restore sessions, the identification information can be utilized to stage the necessary backup objects so that they are available when required; this may be most useful when juke boxes or robotic systems are used to store the backups. DB2 will always check the media header (first record in each session's output) to ensure that the correct data is being restored.

PROMPTING Mode

When a backup or a restore operation is initiated, two prompting modes are possible:

- WITHOUT PROMPTING or NOINTERRUPT, where there is no opportunity for the vendor product to write messages to the user, or for the user to respond to them.
- PROMPTING or INTERRUPT, where the user can receive and respond to messages from the vendor product.

For PROMPTING mode, backup and restore define three possible user responses:

- Continue

  The operation of reading or writing data to the device will resume.
- Device terminate

  The device will receive no additional data, and the session is terminated.
- Terminate

  The entire backup or restore operation is terminated.

The use of the PROMPTING and WITHOUT PROMPTING modes is discussed in the sections that follow.

Device Characteristics

For purposes of the vendor device support APIs, two general types of devices are defined:

- Limited capacity devices requiring user action to change the media; for example, a tape drive, diskette, or CDROM drive.
- Very large capacity devices, where normal operations do not require the user to handle media; for example, a juke box, or an intelligent robotic media handling device.

A limited capacity device may require that the user be prompted to load additional media during the backup or restore operation. Generally DB2 is not sensitive to the order in which the media is loaded for either backup or restore operations. It also provides facilities to pass vendor media handling messages to the user. This prompting requires that the backup or restore operation be initiated with PROMPTING on. The media handling message text is specified in the description field of the return code structure.

If PROMPTING is on, and DB2 receives an SQLUV_ENDOFMEDIA or an SQLUV_ENDOFMEDIA_NO_DATA return code from a **sqluvput** (write) or a **sqluvget** (read) call, DB2:

- Marks the last buffer sent to the session to be resent, if the call was **sqluvput**. It will be put to a session later.
- Calls the session with **sqluvend** (action = SQLUV_COMMIT). If successful (SQLUV_OK return code), DB2:
  - Sends a vendor media handling message to the user from the return code structure that signaled the end-of-media condition.
  - Prompts the user for a continue, device terminate, or terminate response.
- If the response is *continue*, DB2 initializes another session using the **sqluvint** call, and if successful, begins writing data to or reading data from the session. To uniquely identify the session when writing, DB2 increments the sequence number. The sequence number is available in the DB2-INFO structure used with **sqluvint**, and is in the media header record, which is the first data record sent to the session.

  DB2 will not start more sessions than requested when a backup or a restore operation is started, or indicated by the vendor product with a SQLUV_MAX_LINK_GRANT warning on an **sqluvint** call.
- If the response is *device terminate*, DB2 does not attempt to initialize another session, and the number of active sessions is reduced by one. DB2 does not allow all sessions to be terminated by device terminate responses; at least one session must be kept active until the backup or the restore operation completes.
- If the response is *terminate*, DB2 terminates the backup or the restore operation. For more information on exactly what DB2 does to terminate the sessions, see "If Error Conditions Are Returned to DB2" on page 332.

Because backup or restore performance is often dependent on the number of devices being used, it is important that parallelism be maintained. For backup operations, users are encouraged to respond with a `continue`, unless they know that the remaining active sessions will hold the data that is still to be written out. For restore operations, users are also encouraged to respond with a `continue` until all media have been processed.

If the backup or the restore mode is WITHOUT PROMPTING, and DB2 receives an SQLUV_ENDOFMEDIA or an SQLUV_ENDOFMEDIA_NO_DATA return code from a session, it will terminate the session and not attempt to open another session. If all sessions return end-of-media to DB2 before the backup or the restore operation is complete, the operation will fail. Because of this, WITHOUT PROMPTING should be used carefully with limited capacity devices; it does, however, make sense to operate in this mode with very large capacity devices.

It is possible for the vendor product to hide media mounting and switching actions from DB2, so that the device appears to have infinite capacity. Some very large capacity devices operate in this mode. In these cases, it is critical that all the data that was backed up be returned to DB2 in the same order when a restore operation is in progress. Failure to do so could result in missing data, but DB2 assumes a successful restore operation, because it has no way of detecting the missing data.

DB2 writes data to the vendor product with the assumption that each buffer will be contained on one and only one media (for example, a tape). It is possible for the vendor product to split these buffers across multiple media without DB2's knowledge. In this case, the order in which the media is processed during a restore operation is critical, because the vendor product will be responsible for returning reconstructed buffers from the multiple media to DB2. Failure to do so will result in a failed restore operation.

If Error Conditions Are Returned to DB2

When performing a backup or a restore operation, DB2 expects that all sessions will complete successfully; otherwise, the entire backup or restore operation fails. A session signals successful completion to DB2 with an SQLUV_OK return code on the **sqluvend** call, action = SQLUV_COMMIT.

If unrecoverable errors are encountered, the session is terminated by DB2. These can be DB2 errors, or errors returned to DB2 from the vendor product. Because all sessions must commit successfully to have a complete backup or restore operation, the failure of one causes DB2 to terminate the other sessions associated with the operation.

If the vendor product responds to a call from DB2 with an unrecoverable return code, the vendor product can optionally provide additional information, using message text placed in the description field of the RETURN-CODE structure. This message text is presented to the user, along with the DB2 information, so that corrective action can be taken.

There will be backup scenarios in which a session has committed successfully, and another session associated with the backup operation experiences an unrecoverable error. Because all sessions must complete successfully before a backup operation is considered successful, DB2 must delete the output data in the committed sessions: DB2 issues a **sqluvdel** call to request deletion of the object. This call is not considered an I/O session, and is responsible for initializing and terminating any connection that may be necessary to delete the backup object.

The DB2-INFO structure will not contain a sequence number; **sqluvdel** will delete all backup objects that match the remaining parameters in the DB2-INFO structure.

Warning Conditions

It is possible for DB2 to receive warning return codes from the vendor product; for example, if a device is not ready, or some other correctable condition has occurred. This is true for both read and write operations.

On **sqluvput** and **sqluvget** calls, the vendor can set the return code to SQLUV_WARNING, and optionally provide additional information, using message text placed in the description field of the RETURN-CODE structure. This message text is presented to the user so that corrective action can be taken. The user can respond in one of three ways: continue, device terminate, or terminate:

- If the response is *continue*, DB2 attempts to rewrite the buffer using **sqluvput** during a backup operation. During a restore operation, DB2 issues an **sqluvget** call to read the next buffer.
- If the response is *device terminate* or *terminate*, DB2 terminates the entire backup or restore operation in the same way that it would respond after an unrecoverable error (for example, it will terminate active sessions and delete committed sessions).

Operational Hints and Tips

This section provides some hints and tips for building vendor products.

History File

The history file can be used as an aid in database recovery operations. It is associated with each database, and is automatically updated with each backup or restore operation. Information in the file can be viewed, updated, or pruned through the following facilities:

- Control Center
- Command line processor (CLP)
  - LIST HISTORY command
  - UPDATE HISTORY FILE command
  - PRUNE HISTORY command
- APIs
  - db2HistoryOpenScan
  - db2HistoryGetEntry
  - db2HistoryCloseScan
  - db2HistoryUpdate
  - db2Prune

For information about the layout of the file, see db2HistData.

When a backup operation completes, one or more records is written to the file. If the output of the backup operation was directed to vendor devices, the DEVICE field in the history record contains a 0, and the LOCATION field contains either:

- The vendor file name specified when the backup operation was invoked.
- The name of the shared library, if no vendor file name was specified.

For more information about specifying this option, see "Invoking a Backup or a Restore Operation Using Vendor Products".

The LOCATION field can be updated using the Control Center, the CLP, or an API. The location of backup information can be updated if limited capacity devices (for example, removable media) have been used to hold the backup image, and the media is physically moved to a different (perhaps off-site) storage location. If this is the case, the history file can be used to help locate a backup image if a recovery operation becomes necessary.

Invoking a Backup or a Restore Operation Using Vendor Products

Vendor products can be specified when invoking the DB2 backup or the DB2 restore utility from:

- The Control Center
- The command line processor (CLP)
- An application programming interface (API).

The Control Center

The Control Center is the graphical user interface for database administration that is shipped with DB2.

To specify	The Control Center input variable for backup or restore operations
Use of vendor device and library name	Is *Use Library*. Specify the library name (on UNIX based systems) or the DLL name (on the Windows operating system).
Number of sessions	Is *Sessions*.
Vendor options	Is not supported.
Vendor file name	Is not supported.
Transfer buffer size	Is (for backup) *Size of each Buffer*, and (for restore) not applicable.

The Command Line Processor (CLP)

The command line processor (CLP) can be used to invoke the DB2 BACKUP DATABASE or the RESTORE DATABASE command.

To specify	The command line processor parameter	
	for backup is	**for restore is**
Use of vendor device and library name	*library-name*	*shared-library*
Number of sessions	*num-sessions*	*num-sessions*
Vendor options	not supported	not supported
Vendor file name	not supported	not supported
Transfer buffer size	*buffer-size*	*buffer-size*

Application Programming Interface (API)

Two API function calls support backup and restore operations: **db2Backup** for backup and **db2Restore** for restore.

To specify	The API parameter (for both db2Backup and db2Restore) is
Use of vendor device and library name	as follows: In structure *sqlu_media_list*, specify a media type of SQLU_OTHER_MEDIA, and then in structure *sqlu_vendor*, specify a shared library or DLL in *shr_lib*.
Number of sessions	as follows: In structure *sqlu_media_list*, specify *sessions*.
Vendor options	*PVendorOptions*
Vendor file name	as follows: In structure *sqlu_media_list*, specify a media type of SQLU_OTHER_MEDIA, and then in structure *sqlu_vendor*, specify a file name in *filename*.
Transfer buffer size	*BufferSize*

Related reference:

- "sqluvint - Initialize and Link to Device" on page 336
- "sqluvget - Reading Data from Device" on page 339
- "sqluvput - Writing Data to Device" on page 341
- "sqluvend - Unlink the Device and Release its Resources" on page 343
- "sqluvdel - Delete Committed Session" on page 346
- "DB2-INFO" on page 347
- "VENDOR-INFO" on page 350
- "INIT-INPUT" on page 351

- “INIT-OUTPUT” on page 353
- “DATA” on page 353
- “RETURN-CODE” on page 354

sqluvint - Initialize and Link to Device

This function is called to provide information for initialization and establishment of a logical link between DB2 and the vendor device.

Authorization:

One of the following:

- *sysadm*
- *dbadm*

Required connection:

Database

API include file:

sql.h

C API syntax:

```
/* File: sqluvend.h */
/* API: Initialize and Link to Device */
/* ... */
int sqluvint (
  struct Init_input   *,
  struct Init_output  *,
  struct Return_code  *);
/* ... */
```

API parameters:

Init_input
Input. Structure that contains information provided by DB2 to establish a logical link with the vendor device.

Init_output
Output. Structure that contains the output returned by the vendor device.

Return_code
Output. Structure that contains the return code to be passed to DB2, and a brief text explanation.

Usage notes:

For each media I/O session, DB2 will call this function to obtain a device handle. If for any reason, the vendor function encounters an error during initialization, it will indicate it via a return code. If the return code indicates an error, DB2 may choose to terminate the operation by calling the **sqluvend** function. Details on possible return codes, and the DB2 reaction to each of these, is contained in the return codes table (see Table 9 on page 338).

The INIT-INPUT structure contains elements that can be used by the vendor product to determine if the backup or restore can proceed:

- size_HI_order and size_LOW_order

  This is the estimated size of the backup. They can be used to determine if the vendor devices can handle the size of the backup image. They can be used to estimate the quantity of removable media that will be required to hold the backup. It might be beneficial to fail at the first **sqluvint** call if problems are anticipated.
- req_sessions

  The number of user requested sessions can be used in conjunction with the estimated size and the prompting level to determine if the backup or restore operation is possible.
- prompt_lvl

  The prompting level indicates to the vendor if it is possible to prompt for actions such as changing removable media (for example, put another tape in the tape drive). This might suggest that the operation cannot proceed since there will be no way to prompt the user.

  If the prompting level is WITHOUT PROMPTING and the quantity of removable media is greater than the number of sessions requested, DB2 will not be able to complete the operation successfully.

DB2 names the backup being written or the restore to be read via fields in the DB2-INFO structure. In the case of an action = SQLUV_READ, the vendor product must check for the existence of the named object. If it cannot be found, the return code should be set to SQLUV_OBJ_NOT_FOUND so that DB2 will take the appropriate action.

After initialization is completed successfully, DB2 will continue by issuing other data transfer functions, but may terminate the session at any time with an **sqluvend** call.

Return codes:

Table 9. Valid Return Codes for sqluvint and Resulting DB2 Action

Literal in Header File	Description	Probable Next Call	Other Comments
SQLUV_OK	Operation successful.	sqluvput, sqluvget (see comments)	If action = SQLUV_WRITE, the next call will be sqluvput (to BACKUP data). If action = SQLUV_READ, verify the existence of the named object prior to returning SQLUV_OK; the next call will be sqluvget to RESTORE data.
SQLUV_LINK_EXIST	Session activated previously.	no further calls	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend call will not be received, since the session was never established.
SQLUV_COMM_ ERROR	Communication error with device.	no further calls	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend call will not be received, since the session was never established.
SQLUV_INV_VERSION	The DB2 and vendor products are incompatible.	no further calls	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend call will not be received, since the session was never established.
SQLUV_INV_ACTION	Invalid action is requested. This could also be used to indicate that the combination of parameters results in an operation which is not possible.	no further calls	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend call will not be received, since the session was never established.
SQLUV_NO_DEV_ AVAIL	No device is available for use at the moment.	no further calls	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend call will not be received, since the session was never established.
SQLUV_OBJ_NOT_ FOUND	Object specified cannot be found. This should be used when the action on the sqluvint call is ’R’ (read) and the requested object cannot be found based on the criteria specified in the DB2-INFO structure.	no further calls	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend call will not be received, since the session was never established.
SQLUV_OBJS_FOUND	More than 1 object matches the specified criteria. This will result when the action on the sqluvint call is ’R’ (read) and more than one object matches the criteria in the DB2-INFO structure.	no further calls	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend call will not be received, since the session was never established.
SQLUV_INV_USERID	Invalid userid specified.	no further calls	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend call will not be received, since the session was never established.
SQLUV_INV_ PASSWORD	Invalid password provided.	no further calls	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend call will not be received, since the session was never established.

Table 9. Valid Return Codes for sqluvint and Resulting DB2 Action (continued)

Literal in Header File	Description	Probable Next Call	Other Comments
SQLUV_INV_OPTIONS	Invalid options encountered in the vendor options field.	no further calls	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend call will not be received, since the session was never established.
SQLUV_INIT_FAILED	Initialization failed and the session is to be terminated.	no further calls	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend call will not be received, since the session was never established.
SQLUV_DEV_ERROR	Device error.	no further calls	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend call will not be received, since the session was never established.
SQLUV_MAX_LINK_ GRANT	Max number of links established.	sqluvput, sqluvget (see comments)	This is treated as a warning by DB2. The warning tells DB2 not to open additional sessions with the vendor product, because the maximum number of sessions it can support has been reached (note: this could be due to device availability). If action = SQLUV_WRITE (BACKUP), the next call will be sqluvput. If action = SQLUV_READ, verify the existence of the named object prior to returning SQLUV_MAX_LINK_GRANT; the next call will be sqluvget to RESTORE data.
SQLUV_IO_ERROR	I/O error.	no further calls	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend call will not be received, since the session was never established.
SQLUV_NOT_ ENOUGH_SPACE	There is not enough space to store the entire backup image; the size estimate is provided as a 64-bit value in bytes.	no further calls	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend call will not be received, since the session was never established.

sqluvget - Reading Data from Device

After initialization, this function can be called to read data from the device.

Authorization:

One of the following:

- *sysadm*
- *dbadm*

Required connection:

Database

API include file:

sqluvend.h

sqluvget - Reading Data from Device

C API syntax:

```
/* File: sqluvend.h */
/* API: Reading Data from Device */
/* ... */
int sqluvget (
  void * pVendorCB,
  struct Data        *,
  struct Return_code  *);
/* ... */

typedef struct Data
}
   sqlint32  obj_num;
   sqlint32  buff_size;
   sqlint32  actual_buff_size;
   void      *dataptr;
   void      *reserve;
{ Data;
```

API parameters:

pVendorCB
Input. Pointer to space allocated for the DATA structure (including the data buffer) and Return_code.

Data Input/output. A pointer to the *data* structure.

Return_code
Output. The return code from the API call.

obj_num
Specifies which backup object should be retrieved.

buff_size
Specifies the buffer size to be used.

actual_buff_size
Specifies the actual bytes read or written. This value should be set to output to indicate how many bytes of data were actually read.

dataptr
A pointer to the data buffer.

reserve
Reserved for future use.

Usage notes:

This function is used by the restore utility.

Return codes:

Table 10. Valid Return Codes for sqluvget and Resulting DB2 Action

Literal in Header File	Description	Probable Next Call	Other Comments
SQLUV_OK	Operation successful.	sqluvget	DB2 processes the data
SQLUV_COMM_ERROR	Communication error with device.	sqluvend, action = SQLU_ABORT[a]	The session will be terminated.
SQLUV_INV_ACTION	Invalid action is requested.	sqluvend, action = SQLU_ABORT[a]	The session will be terminated.
SQLUV_INV_DEV_HANDLE	Invalid device handle.	sqluvend, action = SQLU_ABORT[a]	The session will be terminated.
SQLUV_INV_BUFF_SIZE	Invalid buffer size specified.	sqluvend, action = SQLU_ABORT[a]	The session will be terminated.
SQLUV_DEV_ERROR	Device error.	sqluvend, action = SQLU_ABORT[a]	The session will be terminated.
SQLUV_WARNING	Warning. This should not be used to indicate end-of-media to DB2; use SQLUV_ENDOFMEDIA or SQLUV_ENDOFMEDIA_NO_ DATA for this purpose. However, device not ready conditions can be indicated using this return code.	sqluvget, or sqluvend, action = SQLU_ABORT	
SQLUV_LINK_NOT_EXIST	No link currently exists.	sqluvend, action = SQLU_ABORT[a]	The session will be terminated.
SQLUV_MORE_DATA	Operation successful; more data available.	sqluvget	
SQLUV_ENDOFMEDIA_NO_ DATA	End of media and 0 bytes read (for example, end of tape).	sqluvend	
SQLUV_ENDOFMEDIA	End of media and > 0 bytes read, (for example, end of tape).	sqluvend	DB2 processes the data, and then handles the end-of-media condition.
SQLUV_IO_ERROR	I/O error.	sqluvend, action = SQLU_ABORT[a]	The session will be terminated.

Next call:

[a] If the next call is an sqluvend, action = SQLU_ABORT, this session and all other active sessions will be terminated.

sqluvput - Writing Data to Device

After initialization, this function can be used to write data to the device.

Authorization:

One of the following:
- *sysadm*
- *dbadm*

Required connection:

sqluvput - Writing Data to Device

Database

API include file:

sqluvend.h

C API syntax:

```
/* File: sqluvend.h */
/* API: Writing Data to Device */
/* ... */
int sqluvput (
  void * pVendorCB,
  struct Data  *,
  struct Return_code  *);
/* ... */

typedef struct Data
}
   sqlint32  obj_num;
   sqlint32  buff_size;
   sqlint32  actual_buff_size;
   void      *dataptr;
   void      *reserve;
{ Data;
```

API parameters:

pVendorCB
Input. Pointer to space allocated for the DATA structure (including the data buffer) and Return_code.

Data Output. Data buffer filled with data to be written out.

Return_code
Output. The return code from the API call.

obj_num
Specifies which backup object should be retrieved.

buff_size
Specifies the buffer size to be used.

actual_buff_size
Specifies the actual bytes read or written. This value should be set to indicate how many bytes of data were actually read.

dataptr
A pointer to the data buffer.

reserve
Reserved for future use.

Usage notes:

This function is used by the backup utility.

Return codes:

Table 11. Valid Return Codes for sqluvput and Resulting DB2 Action

Literal in Header File	Description	Probable Next Call	Other Comments
SQLUV_OK	Operation successful.	sqluvput or sqluvend, if complete (for example, DB2 has no more data)	Inform other processes of successful operation.
SQLUV_COMM_ERROR	Communication error with device.	sqluvend, action = SQLU_ABORT[a]	The session will be terminated.
SQLUV_INV_ACTION	Invalid action is requested.	sqluvend, action = SQLU_ABORT[a]	The session will be terminated.
SQLUV_INV_DEV_HANDLE	Invalid device handle.	sqluvend, action = SQLU_ABORT[a]	The session will be terminated.
SQLUV_INV_BUFF_SIZE	Invalid buffer size specified.	sqluvend, action = SQLU_ABORT[a]	The session will be terminated.
SQLUV_ENDOFMEDIA	End of media reached, for example, end of tape.	sqluvend	
SQLUV_DATA_RESEND	Device requested to have buffer sent again.	sqluvput	DB2 will retransmit the last buffer. This will only be done once.
SQLUV_DEV_ERROR	Device error.	sqluvend, action = SQLU_ABORT[a]	The session will be terminated.
SQLUV_WARNING	Warning. This should not be used to indicate end-of-media to DB2; use SQLUV_ENDOFMEDIA for this purpose. However, device not ready conditions can be indicated using this return code.	sqluvput	
SQLUV_LINK_NOT_EXIST	No link currently exists.	sqluvend, action = SQLU_ABORT[a]	The session will be terminated.
SQLUV_IO_ERROR	I/O error.	sqluvend, action = SQLU_ABORT[a]	The session will be terminated.
Next call: [a] If the next call is an sqluvend, action = SQLU_ABORT, this session and all other active sessions will be terminated. Committed sessions are deleted with an sqluvint, sqluvdel, and sqluvend sequence of calls.			

sqluvend - Unlink the Device and Release its Resources

Ends or unlinks the device, and frees all of its related resources. The vendor must free or release unused resources (for example, allocated space and file handles) before returning to DB2.

Authorization:

One of the following:

- *sysadm*
- *dbadm*

Required connection:

Database

API include file:

sql.h

C API syntax:

```
/* File: sqluvend.h */
/* API: Unlink the Device and Release its Resources */
/* ... */
int sqluvend (
  sqlint32 action,
  void * pVendorCB,
  struct Init_output  *,
  struct Return_code  *);
/* ... */
```

API parameters:

action Input. Used to commit or abort the session:

- SQLUV_COMMIT (0 = to commit)
- SQLUV_ABORT (1 = to abort)

pVendorCB
Input. Pointer to the Init_output structure.

Init_output
Output. Space for Init_output de-allocated. The data has been committed to stable storage for a backup if action is to commit. The data is purged for a backup if the action is to abort.

Return code
Output. The return code from the API call.

Usage notes:

This function is called for each session that has been opened. There are two possible action codes:

- Commit

  Output of data to this session, or the reading of data from the session, is complete.

For a write (backup) session, if the vendor returns to DB2 with a return code of SQLUV_OK, DB2 assumes that the output data has been appropriately saved by the vendor product, and can be accessed if referenced in a later **sqluvint** call.

For a read (restore) session, if the vendor returns to DB2 with a return code of SQLUV_OK, the data should not be deleted, because it may be needed again.

If the vendor returns SQLUV_COMMIT_FAILED, DB2 assumes that there are problems with the entire backup or restore operation. All active sessions are terminated by **sqluvend** calls with action = SQLUV_ABORT. For a backup operation, committed sessions receive a **sqluvint**, **sqluvdel**, and **sqluvend** sequence of calls.

- Abort

  A problem has been encountered by DB2, and there will be no more reading or writing of data to the session.

  For a write (backup) session, the vendor should delete the partial output dataset, and use a SQLUV_OK return code if the partial output is deleted. DB2 assumes that there are problems with the entire backup. All active sessions are terminated by **sqluvend** calls with action = SQLUV_ABORT, and committed sessions receive a **sqluvint**, **sqluvdel**, and **sqluvend** sequence of calls.

  For a read (restore) session, the vendor should not delete the data (because it may be needed again), but should clean up and return to DB2 with a SQLUV_OK return code. DB2 terminates all the restore sessions by **sqluvend** calls with action = SQLUV_ABORT. If the vendor returns SQLUV_ABORT_FAILED to DB2, the caller is not notified of this error, because DB2 returns the first fatal failure and ignores subsequent failures. In this case, for DB2 to have called **sqluvend** with action = SQLUV_ABORT, an initial fatal error must have occurred.

Return codes:

Table 12. Valid Return Codes for sqluvend and Resulting DB2 Action

Literal in Header File	Description	Probable Next Call	Other Comments
SQLUV_OK	Operation successful.	no further calls	Free all memory allocated for this session and terminate.
SQLUV_COMMIT_FAILED	Commit request failed.	no further calls	Free all memory allocated for this session and terminate.
SQLUV_ABORT_FAILED	Abort request failed.	no further calls	

sqluvdel - Delete Committed Session

Deletes committed sessions.

Authorization:

One of the following:

- *sysadm*
- *dbadm*

Required connection:

Database

API include file:

sqluvend.h

C API syntax:

```
/* File: sqluvend.h */
/* API: Delete Committed Session */
/* ... */
int sqluvdel (
  struct Init_input   *,
  struct Init_output  *,
  struct Return_code  *);
/* ... */
```

API parameters:

Init_input
Input. Space allocated for Init_input and Return_code.

Return_code
Output. Return code from the API call. The object pointed to by the Init_input structure is deleted.

Usage notes:

If multiple sessions are opened, and some sessions are committed, but one of them fails, this function is called to delete the committed sessions. No sequence number is specified; **sqluvdel** is responsible for finding all of the objects that were created during a particular backup operation, and deleting them. Information in the INIT-INPUT structure is used to identify the output data to be deleted. The call to **sqluvdel** is responsible for establishing any connection or session that is required to delete a backup object from the vendor device. If the return code from this call is SQLUV_DELETE_FAILED, DB2 does not notify the caller, because DB2 returns the first fatal failure and ignores subsequent failures. In this case, for DB2 to have called **sqluvdel**, an initial fatal error must have occurred.

Return codes:

Table 13. Valid Return Codes for sqluvdel and Resulting DB2 Action

Literal in Header File	Description	Probable Next Call	Other Comments
SQLUV_OK	Operation successful.	no further calls	
SQLUV_DELETE_FAILED	Delete request failed.	no further calls	

DB2-INFO

This structure contains information identifying DB2 to the vendor device.

Table 14. Fields in the DB2-INFO Structure. All fields are NULL-terminated strings.

Field Name	Data Type	Description
DB2_id	char	An identifier for the DB2 product. Maximum length of the string it points to is 8 characters.
version	char	The current version of the DB2 product. Maximum length of the string it points to is 8 characters.
release	char	The current release of the DB2 product. Set to NULL if it is insignificant. Maximum length of the string it points to is 8 characters.
level	char	The current level of the DB2 product. Set to NULL if it is insignificant. Maximum length of the string it points to is 8 characters.
action	char	Specifies the action to be taken. Maximum length of the string it points to is 1 character.
filename	char	The file name used to identify the backup image. If it is NULL, the *server_id, db2instance, dbname*, and *timestamp* will uniquely identify the backup image. Maximum length of the string it points to is 255 characters.
server_id	char	A unique name identifying the server where the database resides. Maximum length of the string it points to is 8 characters.
db2instance	char	The db2instance ID. This is the user ID invoking the command. Maximum length of the string it points to is 8 characters.

Table 14. Fields in the DB2-INFO Structure (continued). All fields are NULL-terminated strings.

Field Name	Data Type	Description

type	char	Specifies the type of backup being taken or the type of restore being performed. The following are possible values:
		When action is SQLUV_WRITE:
		`0 - full database backup`
		`3 - table space level backup`
		When action is SQLUV_READ:
		`0 - full restore`
		`3 - online table space restore`
		`4 - table space restore`
		`5 - history file restore`

dbname	char	The name of the database to be backed up or restored. Maximum length of the string it points to is 8 characters.
alias	char	The alias of the database to be backed up or restored. Maximum length of the string it points to is 8 characters.
timestamp	char	The time stamp used to identify the backup image. Maximum length of the string it points to is 26 characters.
sequence	char	Specifies the file extension for the backup image. For write operations, the value for the first session is 1 and each time another session is initiated with an sqluvint call, the value is incremented by 1. For read operations, the value is always zero. Maximum length of the string it points to is 3 characters.
obj_list	struct sqlu_gen_list	Reseverd for future use.
max_bytes_per_txn	sqlint32	Specifies to the vendor in bytes, the transfer buffer size specified by the user.
image_filename	char	Reserved for future use.
reserve	void	Reserved for future use.
nodename	char	Name of the node at which the backup was generated.
password	char	Password for the node at which the backup was generated.
owner	char	ID of the backup originator.
mcNameP	char	Management class.
nodeNum	SQL_PDB_NODE_TYPE	Node number. Numbers greater than 255 are supported by the vendor interface.

The *filename*, or *server_id*, *db2instance*, *type*, *dbname* and *timestamp* uniquely identifies the backup image. The sequence number, specified by *sequence*, identifies the file extension. When a backup image is to be restored, the same values must be specified to retrieve the backup image. Depending on the vendor product, if *filename* is used, the other parameters may be set to NULL, and vice versa.

Language syntax:

C Structure

```
/* File: sqluvend.h */
/* ... */
typedef struct DB2_info
{
  char                 *DB2_id;
  char                 *version;
  char                 *release;
  char                 *level;
  char                 *action;
  char                 *filename;
  char                 *server_id;
  char                 *db2instance;
  char                 *type;
  char                 *dbname;
  char                 *alias;
  char                 *timestamp;
  char                 *sequence;
  struct sqlu_gen_list  *obj_list;
  long                 max_bytes_per_txn;
  char                 *image_filename;
  void                 *reserve;
  char                 *nodename;
  char                 *password;
  char                 *owner;
  char                 *mcNameP;
  SQL_PDB_NODE_TYPE    nodeNum;
} DB2_info;
/* ... */
```

VENDOR-INFO

This structure contains information identifying the vendor and version of the device.

Table 15. Fields in the VENDOR-INFO Structure. All fields are NULL-terminated strings.

Field Name	Data Type	Description
vendor_id	char	An identifier for the vendor. Maximum length of the string it points to is 64 characters.
version	char	The current version of the vendor product. Maximum length of the string it points to is 8 characters.
release	char	The current release of the vendor product. Set to NULL if it is insignificant. Maximum length of the string it points to is 8 characters.

Table 15. Fields in the VENDOR-INFO Structure (continued). All fields are NULL-terminated strings.

Field Name	Data Type	Description
level	char	The current level of the vendor product. Set to NULL if it is insignificant. Maximum length of the string it points to is 8 characters.
server_id	char	A unique name identifying the server where the database resides. Maximum length of the string it points to is 8 characters.
max_bytes_per_txn	sqlint32	The maximum supported transfer buffer size. Specified by the vendor, in bytes. This is used only if the return code from the vendor initialize function is SQLUV_BUFF_SIZE, indicating that an invalid buffer size was specified.
num_objects_in_backup	sqlint32	The number of sessions that were used to make a complete backup. This is used to determine when all backup images have been processed during a restore operation.
reserve	void	Reserved for future use.

Language syntax:

C Structure

```
typedef struct Vendor_info
{
  char       *vendor_id;
  char       *version;
  char       *release;
  char       *level;
  char       *server_id;
  sqlint32   max_bytes_per_txn;
  sqlint32   num_objects_in_backup;
  void       *reserve;
} Vendor_info;
```

INIT-INPUT

This structure contains information provided by DB2 to set up and to establish a logical link with the vendor device.

Table 16. Fields in the INIT-INPUT Structure. All fields are NULL-terminated strings.

Field Name	Data Type	Description
DB2_session	struct DB2_info	A description of the session from the perspective of DB2.

Table 16. Fields in the INIT-INPUT Structure (continued). All fields are NULL-terminated strings.

Field Name	Data Type	Description
size_options	unsigned short	The length of the options field. When using the DB2 backup or restore function, the data in this field is passed directly from the *VendorOptionsSize* parameter.
size_HI_order	sqluint32	High order 32 bits of DB size estimate in bytes; total size is 64 bits.
size_LOW_order	sqluint32	Low order 32 bits of DB size estimate in bytes; total size is 64 bits.
options	void	This information is passed from the application when the backup or the restore function is invoked. This data structure must be flat; in other words, no level of indirection is supported. Byte-reversal is not done, and the code page for this data is not checked. When using the DB2 backup or restore function, the data in this field is passed directly from the *pVendorOptions* parameter.
reserve	void	Reserved for future use.
prompt_lvl	char	Prompting level requested by the user when a backup or a restore operation was invoked. Maximum length of the string it points to is 1 character.
num_sessions	unsigned short	Number of sessions requested by the user when a backup or a restore operation was invoked.

Language syntax:

C Structure

```
typedef struct Init_input
{
  struct DB2_info  *DB2_session;
  unsigned short   size_options;
  sqluint32        size_HI_order;
  sqluint32        size_LOW_order;
  void             *options;
  void             *reserve;
  char             *prompt_lvl;
  unsigned short   num_sessions;
} Init_input;
```

INIT-OUTPUT

This structure contains the output returned by the vendor device.

Table 17. Fields in the INIT-OUTPUT Structure

Field Name	Data Type	Description
vendor_session	struct Vendor_info	Contains information to identify the vendor to DB2.
pVendorCB	void	Vendor control block.
reserve	void	Reserved for future use.

Language syntax:

C Structure

```
typedef struct Init_output
{
  struct Vendor_info  *vendor_session;
  void                *pVendorCB;
  void                *reserve;
} Init_output;
```

DATA

This structure contains data transferred between DB2 and the vendor device.

Table 18. Fields in the DATA Structure

Field Name	Data Type	Description
obj_num	sqlint32	The sequence number assigned by DB2 during a backup operation.
buff_size	sqlint32	The size of the buffer.

Table 18. Fields in the DATA Structure (continued)

Field Name	Data Type	Description
actual_buf_size	sqlint32	The actual number of bytes sent or received. This must not exceed *buff_size*.
dataptr	void	Pointer to the data buffer. DB2 allocates space for the buffer.
reserve	void	Reserved for future use.

Language syntax:

C Structure

```
typedef struct Data
{
  sqlint32  obj_num;
  sqlint32  buff_size;
  sqlint32  actual_buff_size;
  void      *dataptr;
  void      *reserve;
} Data;
```

RETURN-CODE

This structure contains the return code and a short explanation of the error being returned to DB2.

Table 19. Fields in the RETURN-CODE Structure

Field Name	Data Type	Description
return_code[a]	sqlint32	Return code from the vendor function.
description	char	A short description of the return code.
reserve	void	Reserved for future use.
[a] This is a vendor-specific return code that is not the same as the value returned by various DB2 APIs. See the individual API descriptions for the return codes that are accepted from vendor products.		

Language syntax:

C Structure

```
typedef struct Return_code
{
  sqlint32  return_code,
  char      description[30],
  void      *reserve,
} Return_code;
```

Appendix I. DB2 Universal Database technical information

Overview of DB2 Universal Database technical information

DB2 Universal Database technical information can be obtained in the following formats:

- Books (PDF and hard-copy formats)
- A topic tree (HTML format)
- Help for DB2 tools (HTML format)
- Sample programs (HTML format)
- Command line help
- Tutorials

This section is an overview of the technical information that is provided and how you can access it.

Categories of DB2 technical information

The DB2 technical information is categorized by the following headings:

- Core DB2 information
- Administration information
- Application development information
- Business intelligence information
- DB2 Connect information
- Getting started information
- Tutorial information
- Optional component information
- Release notes

The following tables describe, for each book in the DB2 library, the information needed to order the hard copy, print or view the PDF, or locate the HTML directory for that book. A full description of each of the books in the DB2 library is available from the IBM Publications Center at www.ibm.com/shop/publications/order

The installation directory for the HTML documentation CD differs for each category of information:

```
htmlcdpath/doc/htmlcd/%L/category
```

where:

- *htmlcdpath* is the directory where the HTML CD is installed.
- *%L* is the language identifier. For example, en_US.
- *category* is the category identifier. For example, core for the core DB2 information.

In the PDF file name column in the following tables, the character in the sixth position of the file name indicates the language version of a book. For example, the file name db2d1e80 identifies the English version of the *Administration Guide: Planning* and the file name db2d1g80 identifies the German version of the same book. The following letters are used in the sixth position of the file name to indicate the language version:

Language	**Identifier**
Arabic	w
Brazilian Portuguese	b
Bulgarian	u
Croatian	9
Czech	x
Danish	d
Dutch	q
English	e
Finnish	y
French	f
German	g
Greek	a
Hungarian	h
Italian	i
Japanese	j
Korean	k
Norwegian	n
Polish	p
Portuguese	v
Romanian	8
Russian	r
Simp. Chinese	c
Slovakian	7
Slovenian	l
Spanish	z
Swedish	s
Trad. Chinese	t
Turkish	m

No form number indicates that the book is only available online and does not have a printed version.

Core DB2 information

The information in this category cover DB2 topics that are fundamental to all DB2 users. You will find the information in this category useful whether you are a programmer, a database administrator, or you work with DB2 Connect, DB2 Warehouse Manager, or other DB2 products.

The installation directory for this category is `doc/htmlcd/%L/core`.

Table 20. Core DB2 information

Name	Form Number	PDF File Name
IBM DB2 Universal Database Command Reference	SC09-4828	db2n0x80
IBM DB2 Universal Database Glossary	No form number	db2t0x80
IBM DB2 Universal Database Master Index	SC09-4839	db2w0x80
IBM DB2 Universal Database Message Reference, Volume 1	GC09-4840	db2m1x80
IBM DB2 Universal Database Message Reference, Volume 2	GC09-4841	db2m2x80
IBM DB2 Universal Database What's New	SC09-4848	db2q0x80

Administration information

The information in this category covers those topics required to effectively design, implement, and maintain DB2 databases, data warehouses, and federated systems.

The installation directory for this category is `doc/htmlcd/%L/admin`.

Table 21. Administration information

Name	Form number	PDF file name
IBM DB2 Universal Database Administration Guide: Planning	SC09-4822	db2d1x80
IBM DB2 Universal Database Administration Guide: Implementation	SC09-4820	db2d2x80
IBM DB2 Universal Database Administration Guide: Performance	SC09-4821	db2d3x80
IBM DB2 Universal Database Administrative API Reference	SC09-4824	db2b0x80

Table 21. Administration information (continued)

Name	Form number	PDF file name
IBM DB2 Universal Database Data Movement Utilities Guide and Reference	SC09-4830	db2dmx80
IBM DB2 Universal Database Data Recovery and High Availability Guide and Reference	SC09-4831	db2hax80
IBM DB2 Universal Database Data Warehouse Center Administration Guide	SC27-1123	db2ddx80
IBM DB2 Universal Database Federated Systems Guide	GC27-1224	db2fpx80
IBM DB2 Universal Database Guide to GUI Tools for Administration and Development	SC09-4851	db2atx80
IBM DB2 Universal Database Replication Guide and Reference	SC27-1121	db2e0x80
IBM DB2 Installing and Administering a Satellite Environment	GC09-4823	db2dsx80
IBM DB2 Universal Database SQL Reference, Volume 1	SC09-4844	db2s1x80
IBM DB2 Universal Database SQL Reference, Volume 2	SC09-4845	db2s2x80
IBM DB2 Universal Database System Monitor Guide and Reference	SC09-4847	db2f0x80

Application development information

The information in this category is of special interest to application developers or programmers working with DB2. You will find information about supported languages and compilers, as well as the documentation required to access DB2 using the various supported programming interfaces, such as embedded SQL, ODBC, JDBC, SQLj, and CLI. If you view this information online in HTML you can also access a set of DB2 sample programs in HTML.

The installation directory for this category is `doc/htmlcd/%L/ad`.

Table 22. Application development information

Name	Form number	PDF file name
IBM DB2 Universal Database Application Development Guide: Building and Running Applications	SC09-4825	db2axx80
IBM DB2 Universal Database Application Development Guide: Programming Client Applications	SC09-4826	db2a1x80
IBM DB2 Universal Database Application Development Guide: Programming Server Applications	SC09-4827	db2a2x80
IBM DB2 Universal Database Call Level Interface Guide and Reference, Volume 1	SC09-4849	db2l1x80
IBM DB2 Universal Database Call Level Interface Guide and Reference, Volume 2	SC09-4850	db2l2x80
IBM DB2 Universal Database Data Warehouse Center Application Integration Guide	SC27-1124	db2adx80
IBM DB2 XML Extender Administration and Programming	SC27-1234	db2sxx80

Business intelligence information

The information in this category describes how to use components that enhance the data warehousing and analytical capabilities of DB2 Universal Database.

The installation directory for this category is `doc/htmlcd/%L/wareh`.

Table 23. Business intelligence information

Name	Form number	PDF file name
IBM DB2 Warehouse Manager Information Catalog Center Administration Guide	SC27-1125	db2dix80
IBM DB2 Warehouse Manager Installation Guide	GC27-1122	db2idx80

DB2 Connect information

The information in this category describes how to access host or iSeries data using DB2 Connect Enterprise Edition or DB2 Connect Personal Edition.

The installation directory for this category is `doc/htmlcd/%L/conn`.

Table 24. DB2 Connect information

Name	Form number	PDF file name
APPC, CPI-C, and SNA Sense Codes	No form number	db2apx80
IBM Connectivity Supplement	No form number	db2h1x80
IBM DB2 Connect Quick Beginnings for DB2 Connect Enterprise Edition	GC09-4833	db2c6x80
IBM DB2 Connect Quick Beginnings for DB2 Connect Personal Edition	GC09-4834	db2c1x80
IBM DB2 Connect User's Guide	SC09-4835	db2c0x80

Getting started information

The information in this category is useful when you are installing and configuring servers, clients, and other DB2 products.

The installation directory for this category is `doc/htmlcd/%L/start`.

Table 25. Getting started information

Name	Form number	PDF file name
IBM DB2 Universal Database Quick Beginnings for DB2 Clients	GC09-4832	db2itx80
IBM DB2 Universal Database Quick Beginnings for DB2 Servers	GC09-4836	db2isx80
IBM DB2 Universal Database Quick Beginnings for DB2 Personal Edition	GC09-4838	db2i1x80
IBM DB2 Universal Database Installation and Configuration Supplement	GC09-4837	db2iyx80
IBM DB2 Universal Database Quick Beginnings for DB2 Data Links Manager	GC09-4829	db2z6x80

Tutorial information

Tutorial information introduces DB2 features and teaches how to perform various tasks.

The installation directory for this category is `doc/htmlcd/%L/tutr`.

Table 26. Tutorial information

Name	Form number	PDF file name
Business Intelligence Tutorial: Introduction to the Data Warehouse	No form number	db2tux80
Business Intelligence Tutorial: Extended Lessons in Data Warehousing	No form number	db2tax80
Development Center Tutorial for Video Online using Microsoft Visual Basic	No form number	db2tdx80
Information Catalog Center Tutorial	No form number	db2aix80
Video Central for e-business Tutorial	No form number	db2twx80
Visual Explain Tutorial	No form number	db2tvx80

Optional component information

The information in this category describes how to work with optional DB2 components.

The installation directory for this category is `doc/htmlcd/%L/opt`.

Table 27. Optional component information

Name	Form number	PDF file name
IBM DB2 Life Sciences Data Connect Planning, Installation, and Configuration Guide	GC27-1235	db2lsx80
IBM DB2 Spatial Extender User's Guide and Reference	SC27-1226	db2sbx80
IBM DB2 Universal Database Data Links Manager Administration Guide and Reference	SC27-1221	db2z0x80

Table 27. Optional component information (continued)

Name	Form number	PDF file name
IBM DB2 Universal Database Net Search Extender Administration and Programming Guide **Note:** HTML for this document is not installed from the HTML documentation CD.	SH12-6740	N/A

Release notes

The release notes provide additional information specific to your product's release and FixPak level. They also provides summaries of the documentation updates incorporated in each release and FixPak.

Table 28. Release notes

Name	Form number	PDF file name	HTML directory
DB2 Release Notes	See note.	See note.	doc/prodcd/*%L*/db2ir where *%L* is the language identifier.
DB2 Connect Release Notes	See note.	See note.	doc/prodcd/*%L*/db2cr where *%L* is the language identifier.
DB2 Installation Notes	Available on product CD-ROM only.	Available on product CD-ROM only.	

Note: The HTML version of the release notes is available from the Information Center and on the product CD-ROMs. To view the ASCII file:

- On UNIX-based platforms, see the `Release.Notes` file. This file is located in the `DB2DIR/Readme/%L` directory, where `%L` represents the locale name and `DB2DIR` represents:
  - `/usr/opt/db2_08_01` on AIX
  - `/opt/IBM/db2/V8.1` on all other UNIX operating systems
- On other platforms, see the `RELEASE.TXT` file. This file is located in the directory where the product is installed.

Related tasks:

- "Printing DB2 books from PDF files" on page 365

- “Ordering printed DB2 books” on page 366
- “Accessing online help” on page 366
- “Finding product information by accessing the DB2 Information Center from the administration tools” on page 370
- “Viewing technical documentation online directly from the DB2 HTML Documentation CD” on page 371

Printing DB2 books from PDF files

You can print DB2 books from the PDF files on the *DB2 PDF Documentation* CD. Using Adobe Acrobat Reader, you can print either the entire book or a specific range of pages.

Prerequisites:

Ensure that you have Adobe Acrobat Reader. It is available from the Adobe Web site at www.adobe.com

Procedure:

To print a DB2 book from a PDF file:

1. Insert the *DB2 PDF Documentation* CD. On UNIX operating systems, mount the DB2 PDF Documentation CD. Refer to your *Quick Beginnings* book for details on how to mount a CD on UNIX operating systems.
2. Start Adobe Acrobat Reader.
3. Open the PDF file from one of the following locations:
   - On Windows operating systems:

     `x:\doc\language` directory, where *x* represents the CD-ROM drive letter and *language* represents the two-character territory code that represents your language (for example, EN for English).
   - On UNIX operating systems:

     `/cdrom/doc/%L` directory on the CD-ROM, where */cdrom* represents the mount point of the CD-ROM and *%L* represents the name of the desired locale.

Related tasks:

- “Ordering printed DB2 books” on page 366
- “Finding product information by accessing the DB2 Information Center from the administration tools” on page 370
- “Viewing technical documentation online directly from the DB2 HTML Documentation CD” on page 371

Related reference:

- “Overview of DB2 Universal Database technical information” on page 357

Ordering printed DB2 books

Procedure:

To order printed books:

- Contact your IBM authorized dealer or marketing representative. To find a local IBM representative, check the IBM Worldwide Directory of Contacts at www.ibm.com/shop/planetwide
- Phone `1-800-879-2755` in the United States or `1-800-IBM-4YOU` in Canada.
- Visit the IBM Publications Center at www.ibm.com/shop/publications/order

Related tasks:

- “Printing DB2 books from PDF files” on page 365
- “Finding topics by accessing the DB2 Information Center from a browser” on page 368
- “Viewing technical documentation online directly from the DB2 HTML Documentation CD” on page 371

Related reference:

- “Overview of DB2 Universal Database technical information” on page 357

Accessing online help

The online help that comes with all DB2 components is available in three types:

- Window and notebook help
- Command line help
- SQL statement help

Window and notebook help explain the tasks that you can perform in a window or notebook and describe the controls. This help has two types:

- Help accessible from the **Help** button
- Infopops

The **Help** button gives you access to overview and prerequisite information. The infopops describe the controls in the window or notebook. Window and notebook help are available from DB2 centers and components that have user interfaces.

Command line help includes Command help and Message help. Command help explains the syntax of commands in the command line processor. Message help describes the cause of an error message and describes any action you should take in response to the error.

SQL statement help includes SQL help and SQLSTATE help. DB2 returns an SQLSTATE value for conditions that could be the result of an SQL statement. SQLSTATE help explains the syntax of SQL statements (SQL states and class codes).

Note: SQL help is not available for UNIX operating systems.

Procedure:

To access online help:

- For window and notebook help, click **Help** or click that control, then click **F1**. If the **Automatically display infopops** check box on the **General** page of the **Tool Settings** notebook is selected, you can also see the infopop for a particular control by holding the mouse cursor over the control.
- For command line help, open the command line processor and enter:
  - For Command help:

    ```
    ? command
    ```

    where *command* represents a keyword or the entire command.

  For example, `? catalog` displays help for all the CATALOG commands, while `? catalog database` displays help for the CATALOG DATABASE command.
- For Message help:

  ```
  ? XXXnnnnn
  ```

  where *XXXnnnnn* represents a valid message identifier.

  For example, `? SQL30081` displays help about the SQL30081 message.
- For SQL statement help, open the command line processor and enter:
  - For SQL help:

    ```
    ? sqlstate or ? class code
    ```

    where *sqlstate* represents a valid five-digit SQL state and *class code* represents the first two digits of the SQL state.

    For example, `? 08003` displays help for the 08003 SQL state, while `? 08` displays help for the 08 class code.
  - For SQLSTATE help:

```
help statement
```

where *statement* represents an SQL statement.

For example, `help SELECT` displays help about the SELECT statement.

Related tasks:

- "Finding topics by accessing the DB2 Information Center from a browser" on page 368
- "Viewing technical documentation online directly from the DB2 HTML Documentation CD" on page 371

Finding topics by accessing the DB2 Information Center from a browser

The DB2 Information Center accessed from a browser enables you to access the information you need to take full advantage of DB2 Universal Database and DB2 Connect. The DB2 Information Center also documents major DB2 features and components including replication, data warehousing, metadata, Life Sciences Data Connect, and DB2 extenders.

The DB2 Information Center accessed from a browser is composed of the following major elements:

Navigation tree
The navigation tree is located in the left frame of the browser window. The tree expands and collapses to show and hide topics, the glossary, and the master index in the DB2 Information Center.

Navigation toolbar
The navigation toolbar is located in the top right frame of the browser window. The navigation toolbar contains buttons that enable you to search the DB2 Information Center, hide the navigation tree, and find the currently displayed topic in the navigation tree.

Content frame
The content frame is located in the bottom right frame of the browser window. The content frame displays topics from the DB2 Information Center when you click on a link in the navigation tree, click on a search result, or follow a link from another topic or from the master index.

Prerequisites:

To access the DB2 Information Center from a browser, you must use one of the following browsers:

- Microsoft Explorer, version 5 or later
- Netscape Navigator, version 6.1 or later

Restrictions:

The DB2 Information Center contains only those sets of topics that you chose to install from the *DB2 HTML Documentation CD*. If your Web browser returns a `File not found` error when you try to follow a link to a topic, you must install one or more additional sets of topics *DB2 HTML Documentation CD*.

Procedure:

To find a topic by searching with keywords:

1. In the navigation toolbar, click **Search**.
2. In the top text entry field of the Search window, enter two or more terms related to your area of interest and click **Search**. A list of topics ranked by accuracy displays in the **Results** field.

   Entering more terms increases the precision of your query while reducing the number of topics returned from your query.
3. In the **Results** field, click the title of the topic you want to read. The topic displays in the content frame.

To find a topic in the navigation tree:

1. In the navigation tree, click the book icon of the category of topics related to your area of interest. A list of subcategories displays underneath the icon.
2. Continue to click the book icons until you find the category containing the topics in which you are interested. Categories that link to topics display the category title as an underscored link when you move the cursor over the category title. The navigation tree identifies topics with a page icon.
3. Click the topic link. The topic displays in the content frame.

To find a topic or term in the master index:

1. In the navigation tree, click the “Index” category. The category expands to display a list of links arranged in alphabetical order in the navigation tree.
2. In the navigation tree, click the link corresponding to the first character of the term relating to the topic in which you are interested. A list of terms with that initial character displays in the content frame. Terms that have multiple index entries are identified by a book icon.
3. Click the book icon corresponding to the term in which you are interested. A list of subterms and topics displays below the term you clicked. Topics are identified by page icons with an underscored title.
4. Click on the title of the topic that meets your needs. The topic displays in the content frame.

Related concepts:

- "Accessibility" on page 377
- "DB2 Information Center for topics" on page 379

Related tasks:

- "Finding product information by accessing the DB2 Information Center from the administration tools" on page 370
- "Updating the HTML documentation installed on your machine" on page 372
- "Troubleshooting DB2 documentation search with Netscape 4.x" on page 374
- "Searching the DB2 documentation" on page 375

Related reference:

- "Overview of DB2 Universal Database technical information" on page 357

Finding product information by accessing the DB2 Information Center from the administration tools

The DB2 Information Center provides quick access to DB2 product information and is available on all operating systems for which the DB2 administration tools are available.

The DB2 Information Center accessed from the tools provides six types of information.

Tasks Key tasks you can perform using DB2.

Concepts
Key concepts for DB2.

Reference
DB2 reference information, such as keywords, commands, and APIs.

Troubleshooting
Error messages and information to help you with common DB2 problems.

Samples
Links to HTML listings of the sample programs provided with DB2.

Tutorials
Instructional aid designed to help you learn a DB2 feature.

Prerequisites:

Some links in the DB2 Information Center point to Web sites on the Internet. To display the content for these links, you will first have to connect to the Internet.

Procedure:

To find product information by accessing the DB2 Information Center from the tools:

1. Start the DB2 Information Center in one of the following ways:
   - From the graphical administration tools, click on the **Information Center** icon in the toolbar. You can also select it from the **Help** menu.
   - At the command line, enter **db2ic**.
2. Click the tab of the information type related to the information you are attempting to find.
3. Navigate through the tree and click on the topic in which you are interested. The Information Center will then launch a Web browser to display the information.
4. To find information without browsing the lists, click the **Search** icon to the right of the list.

   Once the Information Center has launched a browser to display the information, you can perform a full-text search by clicking the **Search** icon in the navigation toolbar.

Related concepts:

- "Accessibility" on page 377
- "DB2 Information Center for topics" on page 379

Related tasks:

- "Finding topics by accessing the DB2 Information Center from a browser" on page 368
- "Searching the DB2 documentation" on page 375

Viewing technical documentation online directly from the DB2 HTML Documentation CD

All of the HTML topics that you can install from the *DB2 HTML Documentation CD* can also be read directly from the CD. Therefore, you can view the documentation without having to install it.

Restrictions:

Because the following items are installed from the DB2 product CD and not the *DB2 HTML Documentation CD*, you must install the DB2 product to view these items:

- Tools help
- DB2 Quick Tour
- Release notes

Procedure:

1. Insert the *DB2 HTML Documentation* CD. On UNIX operating systems, mount the *DB2 HTML Documentation CD*. Refer to your *Quick Beginnings* book for details on how to mount a CD on UNIX operating systems.
2. Start your HTML browser and open the appropriate file:
   - For Windows operating systems:
     ```
     e:\Program Files\sqllib\doc\htmlcd\%L\index.htm
     ```

     where *e* represents the CD-ROM drive, and %L is the locale of the documentation that you wish to use, for example, **en_US** for English.
   - For UNIX operating systems:
     ```
     /cdrom/Program Files/sqllib/doc/htmlcd/%L/index.htm
     ```

     where */cdrom/* represents where the CD is mounted, and %L is the locale of the documentation that you wish to use, for example, **en_US** for English.

Related tasks:

- "Finding topics by accessing the DB2 Information Center from a browser" on page 368
- "Copying files from the DB2 HTML Documentation CD to a Web Server" on page 374

Related reference:

- "Overview of DB2 Universal Database technical information" on page 357

Updating the HTML documentation installed on your machine

It is now possible to update the HTML installed from the *DB2 HTML Documentation CD* when updates are made available from IBM. This can be done in one of two ways:

- Using the Information Center (if you have the DB2 administration GUI tools installed).
- By downloading and applying a DB2 HTML documentation FixPak .

Note: This will NOT update the DB2 code; it will only update the HTML documentation installed from the *DB2 HTML Documentation CD*.

Procedure:

To use the Information Center to update your local documentation:

1. Start the DB2 Information Center in one of the following ways:
   - From the graphical administration tools, click on the **Information Center** icon in the toolbar. You can also select it from the **Help** menu.
   - At the command line, enter **db2ic**.
2. Ensure your machine has access to the external Internet; the updater will download the latest documentation FixPak from the IBM server if required.
3. Select **Information Center —> Update Local Documentation** from the menu to start the update.
4. Supply your proxy information (if required) to connect to the external Internet.

The Information Center will download and apply the latest documentation FixPak, if one is available.

To manually download and apply the documentation FixPak :

1. Ensure your machine is connected to the Internet.
2. Open the DB2 support page in your Web browser at: www.ibm.com/software/data/db2/udb/winos2unix/support
3. Follow the link for version 8 and look for the ″Documentation FixPaks″ link.
4. Determine if the version of your local documentation is out of date by comparing the documentation FixPak level to the documentation level you have installed. This current documentation on your machine is at the following level: **DB2 v8.1 GA**.
5. If there is a more recent version of the documentation available then download the FixPak applicable to your operating system. There is one FixPak for all Windows platforms, and one FixPak for all UNIX platforms.
6. Apply the FixPak:
   - For Windows operating systems: The documentation FixPak is a self extracting zip file. Place the downloaded documentation FixPak in an empty directory, and run it. It will create a **setup** command which you can run to install the documentation FixPak.
   - For UNIX operating systems: The documentation FixPak is a compressed tar.Z file. Uncompress and untar the file. It will create a directory named `delta_install` with a script called **installdocfix**. Run this script to install the documentation FixPak.

Related tasks:

- "Copying files from the DB2 HTML Documentation CD to a Web Server" on page 374

Related reference:

- "Overview of DB2 Universal Database technical information" on page 357

Copying files from the DB2 HTML Documentation CD to a Web Server

The entire DB2 information library is delivered to you on the *DB2 HTML Documentation CD*, so you can install the library on a Web server for easier access. Simply copy to your Web server the documentation for the languages that you want.

Procedure:

To copy files from the *DB2 HTML Documentation CD* to a Web server, use the appropriate path:

- For Windows operating systems:

  ```
  E:\Program Files\sqllib\doc\htmlcd\%L\*.*
  ```

  where *E* represents the CD-ROM drive and *%L* represents the language identifier.
- For UNIX operating systems:

  ```
  /cdrom:Program Files/sqllib/doc/htmlcd/%L/*.*
  ```

  where *cdrom* represents the CD-ROM drive and *%L* represents the language identifier.

Related tasks:

- "Searching the DB2 documentation" on page 375

Related reference:

- "Supported DB2 interface languages, locales, and code pages" in the *Quick Beginnings for DB2 Servers*
- "Overview of DB2 Universal Database technical information" on page 357

Troubleshooting DB2 documentation search with Netscape 4.x

Most search problems are related to the Java support provided by web browsers. This task describes possible workarounds.

Procedure:

A common problem with Netscape 4.x involves a missing or misplaced security class. Try the following workaround, especially if you see the following line in the browser Java console:

```
Cannot find class  java/security/InvalidParameterException
```

- On Windows operating systems:

  From the *DB2 HTML Documentation CD*, copy the supplied `x:Program Files\sqllib\doc\htmlcd\locale\InvalidParameterException.class` file to the `java\classes\java\security\` directory relative to your Netscape browser installation, where *x* represents the CD-ROM drive letter and *locale* represents the name of the desired locale.

  Note: You may have to create the `java\security\` subdirectory structure.
- On UNIX operating systems:

  From the *DB2 HTML Documentation CD*, copy the supplied `/cdrom/Program Files/sqllib/doc/htmlcd/locale/InvalidParameterException.class` file to the `java/classes/java/security/` directory relative to your Netscape browser installation, where *cdrom* represents the mount point of the CD-ROM and *locale* represents the name of the desired locale.

  Note: You may have to create the `java/security/` subdirectory structure.

If your Netscape browser still fails to display the search input window, try the following:

- Stop all instances of Netscape browsers to ensure that there is no Netscape code running on the machine. Then open a new instance of the Netscape browser and try to start the search again.
- Purge the browser's cache.
- Try a different version of Netscape, or a different browser.

Related tasks:

- "Searching the DB2 documentation" on page 375

Searching the DB2 documentation

To search DB2's documentation, you need Netscape 6.1 or higher, or Microsoft's Internet Explorer 5 or higher. Ensure that your browser's Java support is enabled.

A pop-up search window opens when you click the search icon in the navigation toolbar of the Information Center accessed from a browser. If you are using the search for the first time it may take a minute or so to load into the search window.

Restrictions:

The following restrictions apply when you use the documentation search:

- Boolean searches are not supported. The boolean search qualifiers *and* and *or* will be ignored in a search. For example, the following searches would produce the same results:
  - servlets *and* beans
  - servlets *or* beans
- Wildcard searches are not supported. A search on *java** will only look for the literal string *java** and would not, for example, find *javadoc*.

In general, you will get better search results if you search for phrases instead of single words.

Procedure:

To search the DB2 documentation:

1. In the navigation toolbar, click **Search**.
2. In the top text entry field of the Search window, enter two or more terms related to your area of interest and click **Search**. A list of topics ranked by accuracy displays in the **Results** field.

   Entering more terms increases the precision of your query while reducing the number of topics returned from your query.
3. In the **Results** field, click the title of the topic you want to read. The topic displays in the content frame.

Note: When you perform a search, the first result is automatically loaded into your browser frame. To view the contents of other search results, click on the result in results lists.

Related tasks:

- "Troubleshooting DB2 documentation search with Netscape 4.x" on page 374

Online DB2 troubleshooting information

With the release of DB2® UDB Version 8, there will no longer be a *Troubleshooting Guide*. The troubleshooting information once contained in this guide has been integrated into the DB2 publications. By doing this, we are able to deliver the most up-to-date information possible. To find information on the troubleshooting utilities and functions of DB2, access the DB2 Information Center from any of the tools.

Refer to the DB2 Online Support site if you are experiencing problems and want help finding possible causes and solutions. The support site contains a large, constantly updated database of DB2 publications, TechNotes, APAR (product problem) records, FixPaks, and other resources. You can use the support site to search through this knowledge base and find possible solutions to your problems.

Access the Online Support site at www.ibm.com/software/data/db2/udb/winos2unix/support, or by clicking the **Online Support** button in the DB2 Information Center. Frequently changing information, such as the listing of internal DB2 error codes, is now also available from this site.

Related concepts:

- "DB2 Information Center for topics" on page 379

Related tasks:

- "Finding product information by accessing the DB2 Information Center from the administration tools" on page 370

Accessibility

Accessibility features help users with physical disabilities, such as restricted mobility or limited vision, to use software products successfully. These are the major accessibility features in DB2® Universal Database Version 8:

- DB2 allows you to operate all features using the keyboard instead of the mouse. See "Keyboard Input and Navigation".
- DB2 enables you customize the size and color of your fonts. See "Accessible Display" on page 378.
- DB2 allows you to receive either visual or audio alert cues. See "Alternative Alert Cues" on page 378.
- DB2 supports accessibility applications that use the Java™ Accessibility API. See "Compatibility with Assistive Technologies" on page 378.
- DB2 comes with documentation that is provided in an accessible format. See "Accessible Documentation" on page 378.

Keyboard Input and Navigation

Keyboard Input

You can operate the DB2 Tools using only the keyboard. You can use keys or key combinations to perform most operations that can also be done using a mouse.

Keyboard Focus

In UNIX-based systems, the position of the keyboard focus is highlighted, indicating which area of the window is active and where your keystrokes will have an effect.

Accessible Display

The DB2 Tools have features that enhance the user interface and improve accessibility for users with low vision. These accessibility enhancements include support for customizable font properties.

Font Settings

The DB2 Tools allow you to select the color, size, and font for the text in menus and dialog windows, using the Tools Settings notebook.

Non-dependence on Color

You do not need to distinguish between colors in order to use any of the functions in this product.

Alternative Alert Cues

You can specify whether you want to receive alerts through audio or visual cues, using the Tools Settings notebook.

Compatibility with Assistive Technologies

The DB2 Tools interface supports the Java Accessibility API enabling use by screen readers and other assistive technologies used by people with disabilities.

Accessible Documentation

Documentation for the DB2 family of products is available in HTML format. This allows you to view documentation according to the display preferences set in your browser. It also allows you to use screen readers and other assistive technologies.

DB2 tutorials

The DB2® tutorials help you learn about various aspects of DB2 Universal Database. The tutorials provide lessons with step-by-step instructions in the areas of developing applications, tuning SQL query performance, working with data warehouses, managing metadata, and developing Web services using DB2.

Before you begin:

Before you can access these tutorials using the links below, you must install the tutorials from the *DB2 HTML Documentation* CD-ROM.

If you do not want to install the tutorials, you can view the HTML versions of the tutorials directly from the *DB2 HTML Documentation CD*. PDF versions of these tutorials are also available on the *DB2 PDF Documentation CD*.

Some tutorial lessons use sample data or code. See each individual tutorial for a description of any prerequisites for its specific tasks.

DB2 Universal Database tutorials:

If you installed the tutorials from the *DB2 HTML Documentation* CD-ROM, you can click on a tutorial title in the following list to view that tutorial.

Business Intelligence Tutorial: Introduction to the Data Warehouse Center
Perform introductory data warehousing tasks using the Data Warehouse Center.

Business Intelligence Tutorial: Extended Lessons in Data Warehousing
Perform advanced data warehousing tasks using the Data Warehouse Center.

Development Center Tutorial for Video Online using Microsoft® Visual Basic
Build various components of an application using the Development Center Add-in for Microsoft Visual Basic.

Information Catalog Center Tutorial
Create and manage an information catalog to locate and use metadata using the Information Catalog Center.

Video Central for e-business Tutorial
Develop and deploy an advanced DB2 Web Services application using WebSphere® products.

Visual Explain Tutorial
Analyze, optimize, and tune SQL statements for better performance using Visual Explain.

DB2 Information Center for topics

The DB2® Information Center gives you access to all of the information you need to take full advantage of DB2 Universal Database™ and DB2 Connect™ in your business. The DB2 Information Center also documents major DB2 features and components including replication, data warehousing, the Information Catalog Center, Life Sciences Data Connect, and DB2 extenders.

The DB2 Information Center accessed from a browser has the following features:

Regularly updated documentation
Keep your topics up-to-date by downloading updated HTML.

Search
Search all of the topics installed on your workstation by clicking **Search** in the navigation toolbar.

Integrated navigation tree
Locate any topic in the DB2 library from a single navigation tree. The navigation tree is organized by information type as follows:

- Tasks provide step-by-step instructions on how to complete a goal.
- Concepts provide an overview of a subject.
- Reference topics provide detailed information about a subject, including statement and command syntax, message help, requirements.

Master index
Access the information in topics and tools help from one master index. The index is organized in alphabetical order by index term.

Master glossary
The master glossary defines terms used in the DB2 Information Center. The glossary is organized in alphabetical order by glossary term.

Related tasks:

- "Finding topics by accessing the DB2 Information Center from a browser" on page 368
- "Finding product information by accessing the DB2 Information Center from the administration tools" on page 370
- "Updating the HTML documentation installed on your machine" on page 372

Appendix J. Notices

IBM may not offer the products, services, or features discussed in this document in all countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described 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:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country/region or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country/region where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions; therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product, and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

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 that has been exchanged, should contact:

IBM Canada Limited
Office of the Lab Director
8200 Warden Avenue
Markham, Ontario
L6G 1C7
CANADA

Such information may be available, subject to appropriate terms and conditions, including in some cases payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems, and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements, or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information 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.

COPYRIGHT LICENSE:

This information may contain sample application programs, in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Each copy or any portion of these sample programs or any derivative work must include a copyright notice as follows:

© (*your company name*) (*year*). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. *_enter the year or years_*. All rights reserved.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both, and have been used in at least one of the documents in the DB2 UDB documentation library.

ACF/VTAM
AISPO
AIX
AIXwindows
AnyNet
APPN
AS/400
BookManager
C Set++
C/370
CICS
Database 2
DataHub
DataJoiner
DataPropagator
DataRefresher
DB2
DB2 Connect
DB2 Extenders
DB2 OLAP Server
DB2 Universal Database
Distributed Relational Database Architecture
DRDA
eServer
Extended Services
FFST
First Failure Support Technology
IBM
IMS
IMS/ESA
iSeries
LAN Distance
MVS
MVS/ESA
MVS/XA
Net.Data
NetView
OS/390
OS/400
PowerPC
pSeries
QBIC
QMF
RACF
RISC System/6000
RS/6000
S/370
SP
SQL/400
SQL/DS
System/370
System/390
SystemView
Tivoli
VisualAge
VM/ESA
VSE/ESA
VTAM
WebExplorer
WebSphere
WIN-OS/2
z/OS
zSeries

The following terms are trademarks or registered trademarks of other companies and have been used in at least one of the documents in the DB2 UDB documentation library:

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel and Pentium are trademarks of Intel Corporation in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

Index

Contacting IBM

In the United States, call one of the following numbers to contact IBM:

- 1-800-237-5511 for customer service
- 1-888-426-4343 to learn about available service options
- 1-800-IBM-4YOU (426-4968) for DB2 marketing and sales

In Canada, call one of the following numbers to contact IBM:

- 1-800-IBM-SERV (1-800-426-7378) for customer service
- 1-800-465-9600 to learn about available service options
- 1-800-IBM-4YOU (1-800-426-4968) for DB2 marketing and sales

To locate an IBM office in your country or region, check IBM's Directory of Worldwide Contacts on the web at www.ibm.com/planetwide

Product information

Information regarding DB2 Universal Database products is available by telephone or by the World Wide Web at www.ibm.com/software/data/db2/udb

This site contains the latest information on the technical library, ordering books, client downloads, newsgroups, FixPaks, news, and links to web resources.

If you live in the U.S.A., then you can call one of the following numbers:

- 1-800-IBM-CALL (1-800-426-2255) to order products or to obtain general information.
- 1-800-879-2755 to order publications.

For information on how to contact IBM outside of the United States, go to the IBM Worldwide page at www.ibm.com/planetwide

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC09-4831-00

Spine information:

IBM® DB2 Universal Database™

Data Recovery and High Availability Guide and Reference

Version 8